SEDORIC 3.0
à NU

SEDORIC et STRATORIC
Versions 3.0 du 01/01/96

Première Partie: Pages 1 à 231

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

Troisième Edition (1998)

Table des matières

AVANT-PROPOS

SEDORIC V3.0, c'est une version majeure, déboguée, opérationnelle et extensible. Mais SEDORIC V3.0, c'est avant tout SEDORIC tout court, c'est à dire un merveilleux petit système d'exploitation, crée par Fabrice Broche et Denis Sebbag. Même s'il s'appuie sur les systèmes antérieurement développés pour l'ORIC (et il serait intéressant de reconstruire les filiations), SEDORIC V1.0 a réellement fait la percée et de loin!

Depuis sa sortie en 1986, il a connu deux évolutions majeures. La première, qui est de taille, concerne notamment l'extension de sa bitmap, réalisée avec brio par Ray McLaughlin (Versions 2.0 et 2.1). Cette extension permet de passer dans les meilleures conditions des disquettes 3" aux disquettes 3,5", c'est à dire de bénéficier d'un doublement de capacité. Les additions et corrections apportées par les versions 2.0 et 2.1 sont résumées en ANNEXE.

La seconde, plus récente puisqu'elle date du 01/01/1996 (Version 3.0, pour le dixième anniversaire, tout un symbole!), est probablement moins spectaculaire, puisqu'elle ne présente que deux nouvelles fonctions, mais répond à un besoin réel: le débogage. En effet, depuis le début, SEDORIC traîne de sérieuses bogues, qui ont perduré en raison de la complexité du code et du manque dramatique de place. L'histoire de la ROM1.1 incompatible avec la ROM1.0 a gelé toute évolution de la version 1.0 de SEDORIC. Le chapitre suivant présente un résumé rapide des nouveautés. Voir aussi en ANNEXE pour plus de détails.

Mais puisque je suis dans une veine historique, je dois dire que SEDORIC V3.0 doit tout à Ray McLaughlin. Il y a quelques années, je me suis mis à décortiquer SEDORIC, parce que je trouvais très pratique 'L'ORIC À NU" de Fabrice Broche (désassemblage des ROM 1.0 et 1.1) et qu'il n'y avait aucune information disponible sur SEDORIC. J'ai bien vite abandonné. Un peu plus tard, Yann Legrand a lancé, dans le CEO-MAG, un appel pour explorer le code de SEDORIC. J'ai repris le collier avec ses encouragements. Il corrigeait mon travail au fur et à mesure que j'avançais, bien péniblement, il faut le dire, car je suis biologiste et pas informaticien! Arrivé aux 4/5 de la fin, j'ai rencontré la gestion de fichiers et j'ai regretté de ne pas avoir étudié les langues orientales au lieu de la biologie! Au bord du suicide, j'ai craqué. C'est Ray qui est venu à mon secours en m'envoyant le morceau qui me manquait. J'ai quand même mis 6 mois à digérer ce morceau. Sans Ray, le livre "SEDORIC À NU" n'aurait jamais vu le jour.

Au cours de mes pérégrinations dans le code de SEDORIC, j'ai appris toutes les subtilités du langage machine. Et SEDORIC est un modèle de code 8 bits concis, efficace et astucieux. On se prend à rêver de ce que ça pourrait donner si les méga-octets de Windows avaient la même densité. Bref, les bogues de SEDORIC ont fini par m'agacer, d'autant plus que les connaissant bien, elles me sautaient sans arrêt aux yeux. J'ai commencé à bricoler un peu, puis à tenir la rubrique "SEDORIC, DO IT YOURSELF" du CEO-MAG, dans laquelle je décrivais mes petites expériences. A la fin, je me suis lancé, j'ai mis tout ça ensemble et c'est devenu la version 3.0

Ce livre représente un énorme travail. Je me suis appuyé sur "SEDORIC À NU" édité par le CEO et épuisé depuis longtemps, ainsi que sur "L'ORIC À NU" de Fabrice Broche, le manuel SEDORIC et 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. J'ai bénéficié de l'aide de nombreux membres du CEO, que je ne citerais pas de peur d'en oublier un. Je dois quand même remercier Laurent, Fabrice et mon complice Claude qui m'ont sollicité, encouragé et aidé constamment.

COMMENT LIRE CE LIVRE

Ce livre n'est pas à lire, mais à utiliser au cas par cas, en fonction de vos problèmes. **Je vous encourage à utiliser en tout premier les ANNEXES** qui contiennent une mine de renseignements, ainsi que la table des matières et les index. Ce livre comporte de nombreuses redites, afin que chaque partie soit compréhensible séparément. J'espère que vous y trouverez les réponses aux questions que vous vous posez. Les parties de commentaires et d'explications générales ont été dégagées des parties de désassemblage, afin que ceux d'entre vous qui sont allergiques au langage machine puissent quand même accéder aux informations susceptibles de les intéresser. Ce livre ne s'adresse pas uniquement aux maniaques du "Langage Machine" et aux bricoleurs, mais aussi très simplement à l'utilisateur "normal" de notre machine favorite, à condition qu'il ne se laisse pas impressionner par le listing de désassemblage.

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

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 "Langage Machine" m'a semblé inutilement compliquée).

Abréviations utilisées:

BRK	pour marquer la présence d'un #00 à la fin de certains messages
CR	Carriage Return (retour en début de ligne)
CRC	Cyclic redundancy check, contrôle de récurrence cyclique
CTRL	contrôle
LF	Line Feed (passage à la ligne suivante)
LL	octet de poids faible (Low)
HH	octet de poids fort (High)

On appelle “page” de mémoire un bloc de 256 octets commençant en HH00 et finissant en HHFF. Par exemple la page #03 va de #0300 à #03FF.

Malgré mes soins, mes relectures et mes vérifications, il est évident qu'un ouvrage de cette importance comporte des erreurs. N'hésitez pas à me faire part de celles que vous trouverez. Vos aurez droit à ma reconnaissance et à la prochaine mise à jour.

NOUVEAUTÉS DE LA VERSION 3.0

Ce livre traite de la version de base 3.006 du 01/01/1996 ainsi que des patches 1 et 2. Cette nouvelle version a été profondément remaniée pour être compatible avec les versions 1.0 et 2.x. Tous les programmes en langage machine peuvent maintenant tourner avec la V3.0.

Cela a été possible en restaurant la table des vecteurs système à sa place d'origine (de #FF43 à FFF9). En contrepartie (la place disponible en RAM overlay étant nulle), j'ai dû déplacer certaines commandes SEDORIC dans une nouvelle BANQUE (n°7). Il s'agit des commandes EXT, STATUS, PROT, UNPROT et SYSTEM qui ne sont jamais utilisées à l'intérieur des programmes BASIC, mais uniquement en mode immédiat.

Autre tribut à la nouveauté, la possibilité de taper les commandes SEDORIC en minuscules a été supprimée. Je me suis longuement expliqué à ce sujet dans le CEO-MAG et je ne vais donc pas recommencer. Sachez seulement que cette possibilité était l'objet de nombreuses bogues.

A part ça, toutes les améliorations de Ray McLaughlin ont été conservées. Elles ont simplement été déplacées dans la RAM overlay. Il faut redire combien elles sont géniales et ont donné du sang neuf à notre système d'exploitation notamment avec la possibilité de tirer parti au maximum des lecteurs de disquettes 3"1/2.

Les BOGUES

Alors, quoi de neuf ? Et bien tout d'abord, toutes les bogues majeures sont maintenant corrigées. Vous pouvez par exemple utiliser la commande LINPUT en toute tranquillité. La routine (fondamentale) "Prendre un caractère au clavier" a été également corrigée. Il en est de même pour de nombreuses bogues secondaires. Les commandes essentielles CLOAD et CSAVE de la ROM dont le fonctionnement était fortement perturbé par SEDORIC sont maintenant opérationnelles.

Les MODIFICATIONS

La commande KEYSAVE a été étendue à la table des commandes pré-définies et sauve donc maintenant la zone #C800 à #C9DD, sans accroissement de taille des fichiers "*.KEY". Ceci permet de pouvoir sauver non seulement les tables "KEYDEF" et "Commandes utilisateur", mais aussi celle des "Commandes pré-définies". Trois jeux de fichier "*.KEY" sont fournis en standard, dont l'ancien de SEDORIC V1.006, le fichier standard de SEDORIC V3.0 et un nouveau fichier destiné aux développer.

La table KEYDEF a été complètement remaniée. Il est maintenant possible d'accéder aux principales commandes SEDORIC avec une combinaison FUNCT+touche et aux principales commandes BASIC avec une combinaison FUNCT+SHIFT+touche. Les caractères ASCII "ê" et "©" ainsi que les commandes SEDORIC sans n° (UNPROT, USING, VISUHIRES, VUSER, WIDTH, WINDOW et !RESTORE) sont maintenant accessibles au clavier.

La table des drives à été mise à jour pour tenir compte des nouveaux lecteurs (82 pistes, double face).

La capacité de formatage maximum des disquettes est passée de 99 à 101 pistes. Ceci ne présente d'intérêt que pour une utilisation avec l'émulateur EUPHORIC qui ne connaît pas la limite physique de 82 pistes des lecteurs 3"1/2. Il est maintenant possible de disposer de 3731 secteurs avec une disquette Master!

Menu détail, quelques messages ont été modifiés afin de pouvoir identifier du premier coup d'oeil la nouvelle version: le numéro de version, ainsi que le copyright et beaucoup plus pratique la mention "_(Master)_" a été remplacée par "_V3_(Mst)_" et "_(Slave_)_" par "_V3_(Slv)_" lors de l'affichage du directory.

Les NOUVELLES COMMANDES

La commande VISUHIRES permet de visualiser les écrans HIRES présents sur une disquette (voir le mode d'emploi dans VISUHIRES.HLP).

La commande CHKSUM est une extension de la commande DIR. Elle indique les adresses de début, fin et exécution des fichiers, ainsi que leur type et la somme de tous les octets qui les constituent. Cette dernière information permet de vérifier que tel fichier est bien lisible et de savoir s'il est identique à tel autre (même checksum) (mode d'emploi dans CHKSUM.HLP).

Un des points les plus importants est que contrairement aux versions précédentes, la version 3.0 n'est pas verrouillée par le manque de place. Bien que limitée, la place libérée permettra d'ajouter encore quelques commandes. Il y aura donc des versions 3.x ultérieures.

En **CONCLUSION**

La liste des modifications apportées peut être consultée dans le fichier SEDORIC3.FIX ainsi que de manière plus extensive en ANNEXE.

En résumé, sachez que les principales commandes qui ont été traitées par Ray ou par moi sont les suivantes: ">", BACKUP, CHKSUM, DKEY, DNAME, DNUM, DSYS, DTRACK, EXT, INIST, INIT, KEYSAVE, LINPUT, LOVE (Prendre un caractère au clavier), PROT, STATUS, SYSTEM, TRACK, UNPROT, VISUHIRES soit 20 commandes!

LA RAM OVERLAY
(Première partie, de C000 à C3FF)

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

ANALYSE DES COMMANDES SEDORIC

Lors de l'initialisation, le NOYAU du code SEDORIC est implanté en **RAM overlay** (C000 à FFFF) et quelques modifications sont apportées aux pages 0, 2, et 4 de la RAM afin de permettre l'analyse des commandes SEDORIC et d'accéder à cette RAM overlay 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 CHRGET, 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 SEDORIC, 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 de SEDORIC.

La routine **CHRGET** (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 sous-programme "exécuter une ligne" de l'interpréteur BASIC. Si cette adresse est CA8A, il s'agit de l'adresse de retour en CA8B appartenant au sous-programme "IF".

La routine d’analyse des commandes SEDORIC 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 CHRGET 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 de SEDORIC 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 SEDORIC on continue sans restaurer l'adresse de retour sur la pile. S'agit-il d'un mot-clé utilisateur? (voir manuel SEDORIC 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 RAM overlay est opérée, puis un saut au sous-programme D3AE (**INTERPRÉTEUR 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 (sous-programme 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 sous-programme 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 (c’est à dire là où le listing de désassemblage traite de l’adresse mémoire C700).

TABLE DES VARIABLES SYSTEME

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
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 SEDORIC)
91/92	longueur de la chaîne
9E/9F	adresse de début des tableaux BASIC, c'est à dire, adresse de **FI**
A0/A1	adresse de fin des tableaux BASIC
C7/C8	adresse du haut de cible pour déplacer un bloc vers le haut
C9/CA	adresse du dernier octet du bloc à déplacer vers le haut
CE/CF	adresse du premier octet du bloc à déplacer vers le haut
D0/D4	ACC1 dont:
D0	longueur de la chaîne
D1/D2	adresse de la chaîne
D3/D4	adresse de la variable
F0/F1	Vecteur Interpréteur (ECB9 ATMOS et 0400 SEDORIC)
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 correspondant au n° logique dans la "Table NL" (cette paire d'octet est l'offset du début du "Channel Buffer" du fichier ouvert correspondant, F3 est nul si fichier est fermé)
	adresse de l'entrée courante dans le "Field Buffer"

	adresse dans le "Channel's own Data Buffer"
	en général, adresse dans **FI** calculée à partir d'un offset AY
F3	longueur de la fiche
F4	flag "?" présent dans le nom de fichier source
F4/F5	nombre total de champs déclarés
	adresse d'un emplacement libre dans le "Field Buffer"
F5	longueur de la chaîne (échange variable alphanumérique/champ)
	index dans le "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 SEDORIC: 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 SEDORIC)
0245/46	Vecteur IRQ (EE22 ATMOS et 0488 SEDORIC)
0248/49	Vecteur NMI (F8B2 ATMOS et 04C4 SEDORIC)
0271	"Couleur" du curseur (#01 ATMOS et #00 SEDORIC)
0274/0275	Clignotement curseur (#0004 ATMOS et #000B SEDORIC)
0276/77	Timer 3 (#6B81 ATMOS et #F6D7 SEDORIC)
02A0	(#FF ATMOS et #05 SEDORIC)
02BE	(#80 ATMOS et #FF SEDORIC)
02F5/F6	Vecteur ! (D336 ATMOS et 0467 SEDORIC)
02FC/FD	Vecteur &() (D336 ATMOS et 0461 SEDORIC)

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). SEDORIC 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 pour PSG, clavier et imprimante

03**00**- 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 correspondante).

03**01**- 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 correspondante). On peut utiliser VIAORA/VIAIRA à la place de VIADRA afin de ne pas modifier les status d'interruption de CA2 et CA1.

03**02**- VIA**DDRB D**irection des **D**onnées **R**egistre port **B**: chacun des 8 bits de ce registre indique si le bit correspondant de VIADRB est en entrée (bit à 0) ou en sortie (bit à 1).

03**03**- VIADDRA idem pour le port A

03**04**- VIA**T1L** LL T1 counter

03**05**- VIA**T1H T**imer **1** HH T1 counter

03**06**- VIA**T1LL** LL T1 latch

03**07**- VIA**T1LH** HH T1 latch

03**08**- VIAT2L **T**imer **2**

03**09**- VIAT2H

03**0A**- VIA**SR S**hift **R**egistre (inutilisable avec l'ORIC)

03**0B**- 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)

03**0C**- 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)

03**0D**- 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

03**0E**- 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

03**0F**- 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

<u>0310 à 031B lecteur de disquette ORIC</u>

0310- Registre de commande (en écriture) et d'état (en lecture) du FDC 1973

0311- Registre de piste du FDC 1973

0312- Registre de secteur du FDC 1973

0313- Registre de données du FDC 1973
0314- Registre de configuration de l'électronique du MICRODISC (IRQ, ROMDIS, sélection lecteur et face...)
En lecture, état de la ligne IRQ du FDC
0315-
0316-
0317-
0318- Ligne DRQ (Data ReQuest) du FDC, lecture seulement
0319-
031A-
031B-

Page #04

04F0/F1	EXEVEC+1	adresse d'exécution
04FB	ROMRAM	flag ROM/RAM overlay 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 #BF

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

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_FAULT_", à 1 = "_READ_FAULT_") commande à destination du FDC
C006-	02		XRWTS (nombre de tentatives possibles en cas de secteur non trouvé)
C007-	08		idem (nombre de tentatives possibles en cas d'erreur de transfert)
C008-	07		
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 externes
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-	00 00 20 20 20 20 20 20 20		nom en 9 caractères (dernière lettre en C031)
C032-	00 00 00		extension en 3 caractères (dernière lettre en C034)
C035-	00 00	PSDESP	coordonnées du secteur de descripteur principal
C037-	00 00	NSTOTP	nombre de secteurs totaux + PROT/UNPROT
			NB: les 16 octets C029 à C038 = une ligne de catalogue
C039-	D2 D2 D2 D2	TABDRV	table d'activation des lecteurs (4 lecteurs double face, 82 pistes par face)
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 correspondant à la touche
C047-	09		sauve X = nombre de caractères dans buffer entrée
C048-	00		type de code de fonction:
			b6=0 si commande SEDORIC (RAM overlay visée)
			b6=1 si commande BASIC (ROM visée)
			b7=0 si commande re-définissable ou pré-définie
			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 sous-programme D843/D845 RAM overlay
C04B-	00		première lettre d'un mot-clé SEDORIC
			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**
C054-	FF B3	FISALO	adresse de FIn du fichier
			longueur de fiches d'un fichier à accès direct **D**

C056-	00 50	EXSALO	adresse d' EXécution du fichier
C058-	00 00	NSRSAV	nombre de secteurs restant à sauver
			nombre de secteurs supplémentaires requis
C05A-	01 00	NSSAV	nombre de secteurs à sauver
C05C-	0B 0A	PSDESC	coordonnées piste/secteur du premier secteur descripteur
			ou de l'avant-dernier secteur de catalogue
C05E-	01	NSDESC	nombre de secteurs descripteurs utilisés
C05F-	0E	PTDESC	pointeur dans le secteur descripteur (BUF1)
C060-	0F 00 05 01 E2 26		buffer pour la lecture d'un en-tête secteur
C066-	23 DE 80		ces 4 séquences,
C069-	23 DE 80		concernent les 4 routines
C06C-	23 DE 80		définies
C06F-	23 DE 80		par la commande USER
C072-	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)
C073-	01		flag pour affichage de DEFAFF (affichage d'un nombre décimal)
			flag BACKUP "source in drive" (à 1 si en place)
C074-	00		flag BACKUP "format" (à 1 si formatage demandé)
C075-	2E		sauvegarde de "caractère" pour LINPUT
C076/C07F			"Général Field Buffer" (entrée courante du "Field Buffer") dont:
C076/C07A	00 00 00 00 00		nom du champ (5 caractères significatifs)
C07B-	00		index de l'élément de pseudo-tableau
C07C-	00		n° logique pour ce champ
C07D-	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"
C07E-	00		longueur du champ (1 si octet, 2 si entier, 5 si réel, l si alphanumérique)
C07F-	00		type de champ (#00 réel, #01 entier, #40 octet, #80 alphanumérique)
C080-	00		sauvegarde du n° logique de la dernière commande FIELD
C081-	00		compteur de longueur totale des champs du "Field Buffer"
			puis #01, #40 ou #80 (si CLOSE)
C082-	00		flag mis à #80 lors de CLOSE
			flag du point d'entrée du sous-programme F4E6/F4E9/F4EC/F4EF:
			#00 pour localiser un nom de champ
			#01 pour vérifier qu'un nom de champ particulier existe
			#40 pour trouver une place pour un nouveau nom de champ
			#80 pour supprimer tous les noms de champs associés au fichier
C083-	00		HH de l'adresse de Buffer
			longueur d'une fiche (OPEN R) ou #00 (OPEN S ou OPEN D)
C084-	00		pointeur dans le dernier descripteur
C085/08/09			nombre d'octets précédant la fiche dans le fichier (sur 3 octets)
C085-	00		rang de l'octet de début de la fiche dans le secteur
C086-	00		index dans la liste des coordonnées du descripteur courant
C087-	00		n° du descripteur courant
C088-	00		index dans le buffer lu ou à écrire sur la disquette
C089-	00 00		DEBBAS (vise le lien de la première ligne) (SEEK)

C08B-	00	longueur de la chaîne à chercher (SEEK)
C08C-	00	position dans liste des coordonnées pour COPY
C08D-	00 00	nombre de secteurs restant à charger par COPY
C08F-	00	position de pointeur pour gestion des fichiers

C090-	00 00 00 00 00 00 00 00 00 00 00 00 00	nom_de_fichier_ambigu "Source" pour COPY*
C09D-	00 00 00 00 00 00 00 00 00 00 00 00 00	nom_de_fichier_ambigu "Cible" pour COPY*
C0AA-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Zone de 79 caractères
C0B8-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	pour stocker
C0C6-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	la chaîne à chercher
C0D4-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	par la commande
C0E2-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SEEK
C0F0-	00 00 00 00 00 00 00 00 00	(de C0AA à C0F8)

C0F9-	00	drive source pour BACKUP
C0FA-	00	drive cible pour BACKUP
C0FB-	00 00	nombre de secteurs par face restant à BACKUPer
C0FD-	00	HH de la taille du tampon de BACKUP (#92 ou #AF)
C0FE-	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

L'utilisation de BUF1 est universelle. Il peut servir à lire ou à écrire divers secteurs de la disquette. Voici par exemple le Secteur Système (secteur 1 de la piste 20):

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
C100-  D2 D2 D2 D2 40 64 00 0A 00 94 41 6E 64 72 7B 20  RRRR@d....André
C110-  43 68 7B 72 61 6D 79 20 7A 7A 7A 7A 20 90 50 52  Chéramy zzzz .PR
C120-  49 4E 54 22 42 6F 6E 6A 6F 75 72 2C 20 76 6F 69  INT"Bonjour, voi
C130-  63 69 20 6C 65 20 53 7B 64 6F 72 69 63 20 6E 6F  ci le SEDORIC no
C140-  75 76 65 61 75 21 22 3A 50 49 4E 47 3A 50 52 49  uveau!":PING:PRI
C150-  4E 54 22 53 61 6C 75 74 21 22 00 00 00 00 00 00  NT"Salut!"......
C160-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C170-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C180-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C190-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1A0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1B0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1C0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1D0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1E0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C1F0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Ce secteur est commenté en détail en ANNEXE.

Le BUF1 est également utilisé pour lire les secteurs de descripteurs. Voici par exemple le début du descripteur d'un des fichiers système, celui de la BANQUE n°7:

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000-  00 00 FF 40 00 C4 FF C7 00 00 04 00 05 0B 05 0C  ...@.D.G........
0010-  05 0D 05 0E 00 00 00 00 00 00 00 00 00 00 00 00  ................ etc
```

Ce secteur est commenté en détail en ANNEXE. Vous y trouverez de nombreux autres exemples.

BUFFER 2 (BUF2) C200 À C2FF

BUFFER DE LECTURE/ÉCRITURE DES SECTEURS DE BITMAP

Voici un exemple de premier secteur 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
C200-  FF 00 5F 02 00 00 2A 11 01 2A 00 00 00 00 00 00
C210-  00 00 00 00 00 00 00 00 00 00 00 00 F8 FF FF FF
C220-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C230-  FF FF FF FF FF FF FF FF FF FF 0F DB F6 FF FF FF
C240-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C250-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C260-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C270-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C280-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C290-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2A0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2B0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2C0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2D0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2E0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C2F0-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Le deuxième secteur de bitmap correspondant à l'exemple ci-dessus commence ainsi:

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000-  FF 00 CA 02 00 00 2A 11 01 2A 00 00 00 00 00 00
0010-  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF etc...
```

La première ligne des 2 secteurs de bitmap est identique à l'exception des octets n°2 et 3 qui indiquent le nombre de secteurs totaux (ici #02CA = 714 soit 17 x 42) au lieu du nombre de secteurs libres (ici #025F = 607 soit 714 - 607 = 107 secteurs utilisés par SEDORIC sur une disquette Master).

Ces deux secteurs de bitmap sont commentés plus en détail en ANNEXE.

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 50 00 00 00 00 00 00 00 00 00 00 00 00 00
C310-  50 41 54 43 48 20 20 20 20 30 30 32 05 0F 04 40  PATCH    002...@
C320-  50 41 54 43 48 20 20 20 20 30 30 31 06 02 06 40  PATCH    001...@
C330-  50 41 54 43 48 48 45 4C 50 30 30 31 06 08 06 40  PATCHHELP001...@
C340-  50 41 54 43 48 48 45 4C 50 30 30 32 0E 08 06 40  PATCHHELP002...@
C350-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C360-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C370-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C380-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C390-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C3A0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C3B0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C3C0-  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
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  ................
```

La structure de ce genre de secteur est commentée en ANNEXE.

BANQUE n°0 (C400 à C7FF)

INITIALISATION SEDORIC
(CETTE BANQUE SERA ÉCRASÉE PAR LA SUITE)

Entrée réelle?

C400-	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)
C403-	4A	LSR	b0 -> C qui passe à zéro (mais peut-être à 1?)
C404-	A9 00	LDA #00	A = 0000 0000
C406-	6A	ROR	C passe dans le b7
C407-	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
C40A-	10 0F	BPL C41B	continue en C41B si ROM V1.0

ROM V1.1

C40C-	A9 50	LDA #50	A = 80 caractères
C40E-	8D 56 02	STA 0256	longueur ligne imprimante V 1.1
C411-	4A	LSR	A = 40 caractères et Z = 0
C412-	85 31	STA 31	longueur ligne imprimante V 1.0
C414-	85 32	STA 32	position maximale pour tabulation par ","
C416-	8D 57 02	STA 0257	longueur d'une ligne écran V 1.1
C419-	D0 06	BNE C421	suite forcée en C421 car Z = 0 dans tous les cas

ROM V1.0

C41B-	A9 5D	LDA #5D	A = 93 (à cause d'une bogue de la ROM V 1.0)
C41D-	85 31	STA 31	longueur ligne imprimante V 1.0
C41F-	85 32	STA 32	position maximale pour tabulation par ","

Mise en place de la page 4

C421-	EE C1 02	INC 02C1	LL de HIMEM soit #00 -> #01
C424-	EE C2 02	INC 02C2	HH de HIMEM soit #98 -> #99, HIMEM = #9901
C427-	A2 00	LDX #00	mise en place de la page 4: remet à zéro l' index X
C429-	BD 00 C6	LDA C600,X	pointe par défaut sur version ORIC-1
C42C-	2C 24 C0	BIT C024	teste ATMORI: = #80 si V 1.1 N = 1 (négatif)
C42F-	10 03	BPL C434	si N = 0 (version ORIC-1) Ok on copie
C431-	BD 00 C7	LDA C700,X	sinon on remplace par version ATMOS
C434-	9D 00 04	STA 0400,X	recopie en page 4
C437-	E8	INX	compteur passera de #00 à #FF, soit 256 octets

```
C438-  D0 EF      BNE C429      reboucle jusqu'à #FF inclus
```

Modification de la routine CHRGET en page 0

```
C43A-  A9 4C      LDA #4C
C43C-  A0 00      LDY #00
C43E-  A2 04      LDX #04
C440-  85 EF      STA EF        remplace JSR ECB9 par JMP 0400
C442-  84 F0      STY F0
C444-  86 F1      STX F1
```

Modification vecteurs IRQ et NMI

```
C446-  A9 88      LDA #88
C448-  A0 C4      LDY #C4       NB: X a encore la valeur #04
C44A-  2C 24 C0   BIT C024      teste ATMORI: si V 1.1 N = 1 (négatif)
C44D-  10 26      BPL C475      si ORIC-1, continue en C475
C44F-  8D 45 02   STA 0245      remplace JMP #EE22 (IRQ)
C452-  8E 46 02   STX 0246      par JMP #0488 (new IRQ)
C455-  8C 48 02   STY 0248      remplace JMP #F8B2 (NMI)
C458-  8E 49 02   STX 0249      par JMP #04C4 (new NMI)
```

Modification sous-programme "Prendre un caractère au clavier"

```
C45B-  A9 5B      LDA #5B
C45D-  8D 3C 02   STA 023C      remplace JMP #EB78
C460-  8E 3D 02   STX 023D      par JMP #045B
```

Modification délai et vitesse d'autorépétition clavier

```
C463-  A9 09      LDA #09
C465-  A0 01      LDY #01
C467-  8D 4E 02   STA 024E      délai: remplace #10 par #09
C46A-  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" SEDORIC
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 (Handler d' IRQ) placé en #FFFE (sur RAM overlay)
C49C-  A9 67     LDA #67
C49E-  A0 61     LDY #61
C4A0-  8D F5 02  STA 02F5    vecteur "!" re-dirigé sur #0467
C4A3-  8E F6 02  STX 02F6
C4A6-  8C FC 02  STY 02FC    vecteur "&()" re-dirigé 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 n°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 SEDORIC 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 SEDORIC
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 cinquième (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     XCHAR sélectionne le jeu de caractères correct (“normal” ou “accentué”)
                              selon MODCLA
C543-  20 D8 D5  JSR D5D8     XROM exécute à partir de la RAM un sous-programme 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 (sous-programme 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)

MESSAGE: DOS IS ALTERED!

(Ce message se termine par #00. Il n'est décomposé en 5 morceaux que pour en faciliter la compréhension)

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
CRLF**BRK**

Rappel: **BRK** = pour marquer la présence d'un #00 à la fin de certains messages
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

Ceci est un résidu de fausse couche! (Voir plus loin)

C599-	4C 64 D3	JMP D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C59C-	60	RTS	et retourne (mais sous-programme appelé de nulle part!)
C59D-	AD AE C5	LDA C5AE	A = #27 (idem: sous-programme 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 à gauche sous l'adresse)

C5B1- 49 4E 20 44 52 49 56 45 **A0**
01 IN_DRIVE_

C5BA- 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_

C5D5- 20 54 4F **A0**
03 _TO_

C5D9- 0D 0A 4C 4F 41 44 20 53 4F 55 52 43 45 20 44 49 53 43 **A0**
04 CRLFLOAD_SOURCE_DISC_

C5EC- 0D 0A 4C 4F 41 44 20 54 41 52 47 45 54 20 44 49 53 53 2C 28
05 CRLFLOAD_TARGET_DISA-+

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

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 (le "C" de DISC, puis LF et CR). Les valeurs suivantes peuvent être trouvées: #53, #2C, #28, ou #41, #F1, #2B (SEDORIC V1.0), #41, #2D, #2B (SEDORIC V2.x et 3.0), #53, #52, #28 (STRATORIC V1.0). Il peut s'agir d'une zone DATA utilisée pendant le BOOT et écrasée par la suite.

BANQUE n°0, troisième secteur:
Source de la page 4 version ORIC-1

```
C600-  C9 30 90 04 C9 3A 90 35 86 0F AA 30 2E 85 C1 68     bogue "CSAVE" corrigée
C610-  AA 68 48 E0 F7 D0 04 C9 C8 F0 09 E0 58 D0 18 C9
C620-  CA D0 14 24 18 6E FC 04 A0 FF C8 B1 E9 F0 11 C9
C630-  3A F0 0D C9 D4 D0 F3 8A 48 A5 C1 A6 0F 4C 41 EA     bogue "CSAVE" corrigée
C640-  68 20 E9 04 20 67 04 0E FC 04 B0 03 4C AD C8 EA
C650-  EA EA 60 20 77 04 B1 16 4C 77 04 EA EA EA EA EA
C660-  EA A9 8E A0 F8 D0 04 A9 AE A0 D3 8D F0 04 8C F1
C670-  04 20 77 04 20 EF 04 08 48 78 AD FB 04 49 02 8D
C680-  FB 04 8D 14 03 68 28 60 2C 0D 03 50 0F 48 A9 04
C690-  2D 6A 02 F0 03 EE 74 02 68 4C 03 EC 68 68 85 F2
C6A0-  68 AA A9 36 A0 D1 D0 C3 20 F2 04 68 40 8D 14 03
C6B0-  6C FC FF 18 20 77 04 48 A9 04 48 A9 A8 48 08 B0
C6C0-  03 4C 28 02 20 88 F8 A9 17 A0 EC 20 6B 04 4C 75
C6D0-  C4 A9 04 48 A9 F1 48 8A 48 98 48 20 F2 04 4C 70
C6E0-  D2 EA EA EA EA EA EA EA 4C 87 04 4C 71 04 4C
C6F0-  00 00 4C 77 04 4C B3 04 4C B4 04 84 00 00 00 00
```

En C60E et C63A, correction de la bogue "CSAVE". L'ancienne adresse 0E devient **C1** (2 octets différents). C'est la sauvegarde de l'accumulateur A en 0E qui créait une interférence avec les commandes CLOAD et CSAVE de la ROM.

BANQUE n°0, quatrième secteur:
Source de la page 4 version ATMOS

```
C700-  C9 30 90 04 C9 3A 90 35 86 0F AA 30 2E 85 C1 68     bogue "CSAVE" corrigée
C710-  AA 68 48 E0 0E D0 04 C9 C9 F0 09 E0 8A D0 18 C9
C720-  CA D0 14 24 18 6E FC 04 A0 FF C8 B1 E9 F0 11 C9
C730-  3A F0 0D C9 D4 D0 F3 8A 48 A5 C1 A6 0F 4C B9 EC     bogue "CSAVE" corrigée
C740-  68 20 E9 04 20 67 04 0E FC 04 B0 03 4C C1 C8 6E
C750-  52 02 60 20 77 04 B1 16 4C 77 04 A9 45 A0 D8 D0
C760-  0A A9 8E A0 F8 D0 04 A9 AE A0 D3 8D F0 04 8C F1
C770-  04 20 77 04 20 EF 04 08 48 78 AD FB 04 49 02 8D
C780-  FB 04 8D 14 03 68 28 60 2C 0D 03 50 0F 48 A9 04
C790-  2D 6A 02 F0 03 EE 74 02 68 4C 22 EE 68 68 85 F2
C7A0-  68 AA A9 36 A0 D1 D0 C3 20 F2 04 68 40 8D 14 03
C7B0-  6C FC FF 18 20 77 04 48 A9 04 48 A9 A8 48 08 B0
C7C0-  03 4C 44 02 20 B8 F8 A9 17 A0 EC 20 6B 04 4C 71
C7D0-  C4 A9 04 48 A9 F1 48 8A 48 98 48 20 F2 04 4C 06
C7E0-  D3 EA EA EA EA EA EA EA 4C 87 04 4C 71 04 4C
C7F0-  00 00 4C 77 04 4C B3 04 4C B4 04 84 00 00 00 00
```

En C70E et C73A, correction de la bogue "CSAVE". L'ancienne adresse 0E devient **C1** (2 octets différents). C'est la sauvegarde de l'accumulateur A en 0E qui créait une interférence avec les commandes CLOAD et CSAVE de la ROM.

DÉSASSEMBLAGE DE LA PAGE 4 SEDORIC

Complément de l'interpréteur BASIC
(en fait s'intercale entre la fin du sous-programme 00E2 et le début du sous-programme 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 C1 (correction de la bogue "CSAVE") et 0F
040B-	30 2E	BMI 043B	si A est négatif (token BASIC)
040D-	85 C1	STA C1	récupère A et X et continue en ECB9
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 C1	LDA C1	correction de la 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 "!" SEDORIC
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-programme ROM exécuter ligne)

044F-	6E 52 02	ROR 0252	sinon met à 1 le b7 du flag
0452-	60	RTS	"IF" (0252) et fin sous-programme 0400

NB: Lorsque le sous-programme 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 sous-programme "IF", l'adresse CA8A est empilée.

Gestion du vecteur d'exécution

0453-	20 77 04	JSR 0477	entrée appelée de la RAM overlay 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 RAM overlay
045B-	A9 45	LDA #45	sous-programme EB78 modifié
045D-	A0 D8	LDY #D8	initialise pour exécution
045F-	D0 0A	BNE 046B	en D845 (XKEY prend 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	sous-programme à exécuter (EXEVEC+1)
0471-	20 77 04	JSR 0477	entrée sous-programme EXERAM: bascule
0474-	20 EF 04	JSR 04EF	ROM/RAM overlay, vecteur EXEVEC
0477-	08	PHP	entrée sous-programme bascule ROM/RAM overlay
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
047F-	8D FB 04	STA 04FB	STA 0321
0482-	8D 14 03	STA 0314	PLA PLP RTS
0485-	68	PLA	qui est plus courte et se
0486-	28	PLP	termine en 0484, la zone
0487-	60	RTS	0485 à 0487 est donc libre

NB: ce sous-programme permet d'exécuter les routines F590, D3AE, D136, EC17, F88E, D845

Nouvel IRQ (remplace EE22 de la ROM)

0488-	2C 0D 03	BIT 030D	(sous-programme utilisé à partir de la ROM)
048B-	50 0F	BVC 049C	
048D-	48	PHA	
048E-	A9 04	LDA #04	
0490-	2D 6A 02	AND 026A	mode console ORIC-1/ATMOS
0493-	F0 03	BEQ 0498	
0495-	EE 74 02	INC 0274	clignotement curseur
0498-	68	PLA	

```
0499-  4C 22 EE   JMP EE22     on est bien sur la ROM

049C-  68         PLA
049D-  68         PLA
049E-  85 F2      STA F2
04A0-  68         PLA
04A1-  AA         TAX
04A2-  A9 36      LDA #36      initialisation pour exécution
04A4-  A0 D1      LDY #D1      du sous-programme D136 en RAM overlay
04A6-  D0 C3      BNE 046B     avec retour sur la ROM
```

Nouveau COLDSTART (remplace FFFC et donc F88F)

```
04A8-  20 F2 04   JSR 04F2     bascule ROM/RAM overlay
04AB-  68         PLA          dépile
04AC-  40         RTI          retour d'interruption
04AD-  8D 14 03   STA 0314     flag ROM/RAM overlay
04B0-  6C FC FF   JMP (FFFC)   vecteur coldstart ROM
```

Sous-programme IRQRAM et NMIRAM

```
04B3-  18         CLC          entrée IRQRAM (04B4 = entrée NMIRAM)
04B4-  20 77 04   JSR 0477     bascule ROM/RAM overlay
04B7-  48         PHA          empile A
04B8-  A9 04      LDA #04
04BA-  48         PHA
04BB-  A9 A8      LDA #A8
04BD-  48         PHA          empile adresse 04A8 (coldstart)
04BE-  08         PHP          empile indicateurs d'état 6502
04BF-  B0 03      BCS 04C4     si NMI saute en 04C4
04C1-  4C 44 02   JMP 0244     si IRQ continue en 0244 soit en 0488

04C4-  20 B8 F8   JSR F8B8     nouvel NMI: en ROM F8B8 au lieu
04C7-  A9 17      LDA #17      de F8B2 (saute warmstart BASIC)
04C9-  A0 EC      LDY #EC
04CB-  20 6B 04   JSR 046B     exécute sous-programme EC17 (XSTATUS initialise PAPER, INK, mode
                               clavier et status console) sur RAM overlay
04CE-  4C 71 C4   JMP C471     exécute enfin warmstart BASIC
```

Chercher un tableau (lorsqu'on est en RAM overlay)

```
04D1-  A9 04      LDA #04
04D3-  48         PHA          empile 04F1 adresse de retour -1
04D4-  A9 F1      LDA #F1      (le retour se fera en 04F2,
04D6-  48         PHA          c'est à dire bascule ROM/RAM overlay)
04D7-  8A         TXA
04D8-  48         PHA          sauve X sur la pile
04D9-  98         TYA
```

04DA-	48	PHA	sauve Y sur la pile
04DB-	20 F2 04	JSR 04F2	bascule ROM/RAM overlay (ici, accède à la ROM)
04DE-	4C 06 D3	JMP D306	trouver le tableau et retour en RAM overlay

Vecteurs et Drapeaux

04E1-	EA EA EA EA EA EA EA EA NOP NOP NOP NOP NOP NOP NOP NOP		
04E9-	4C 87 04	JMP 0487	DETE2C vecteur utilisateur
04EC-	4C 71 04	JMP 0471	EXERAM exécution de sous-programme indiqué à EXEVEC+1 sur RAM overlay (si ROM active) ou sur ROM (si RAM overlay active)
04EF-	4C 00 00	JMP 0000	EXEVEC vecteur exécution, dont..
04F0-	00 00		adresse exécution
04F2-	4C 77 04	JMP 0477	RAMROM bascule RAM overlay/ROM
04F5-	4C B3 04	JMP 04B3	IRQRAM exécution IRQ sur RAM
04F8-	4C B4 04	JMP 04B4	NMIRAM exécution NMI sur RAM
04FB-	94		flag ROM/RAM overlay
04FC-	00	BRK	flag C90E/CA8A
04FD-	00	BRK	numéro de l'erreur
04FE-	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: RENUM, DELETE et MOVE

Cette BANQUE se trouve à partir du #42 (soixante sixième) secteur de la disquette MASTER.

C400a	00 00	EXTER	Adresse des messages d'erreur externes (néant)
C402a	00 00	EXTMS	Adresse des messages externes (néant)
C404a	4C 32 C4	JMP C432	Entrée commande RENUM
C407a	4C EC C6	JMP C6EC	Entrée commande DELETE
C40Aa	4C 56 C7	JMP C756	Entrée commande MOVE

EXÉCUTION DE LA COMMANDE SEDORIC RENUM

Rappel de la syntaxe

RENUM (NEWNUM)(,NEWPAS)(,PRELGN)(,DERLGN)

Re-numérote 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

00/01	copie de DEBBAS (9A/9B), début programme BASIC, pointe sur le premier lien
02/03	copie de FINBAS (9C/9D), fin du programme BASIC
04/05	copie de HIMEM (A6/A7), limite de la mémoire utilisable
08/09	pointeur source pour relocation finale, initialisé à DEBBAS-1
0A/0B	pointeur cible pour relocation finale, initialisé à DEBBAS-1
C7/C8	copie de HIMEM (A6/A7), adresse haute de la cible lors de la première relocation
C9/CA	copie de FINBAS (9C/9D), adresse haute de la source lors de première relocation
CE/CF	DEBBAS-1 (vise #00 début programme) adresse basse source première relocation
F2	Index pour la table "RENUM" contenant les paramètres à utiliser
F4/F5	pointeur dans le bloc bas (cible)
F8/F9	initialisé avec #00F3, utilisé pour pointage de lien
E9/EA	TXTPTR, pointeur dans le bloc haut (source)
C6E4/C6E5	NEWNUM, égale DEFNUM par défaut
C6E6/C6E7	NEWPAS, égale DEFPAS par défaut
C6E8/C6E9	PRELGN, 0 par défaut
C6EA/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) de SEDORIC: C'est un comble! La documentation n'est pas claire la-dessus: en effet "!restore" fait bel et bien appel au restore de SEDORIC, 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 de SEDORIC ont dû faire preuve d'imagination pour analyser les lignes BASIC afin de re-numéroter correctement les GOTO, GOSUB etc..

voir la routine C548 qui vaut son pesant d'or!

Utilisation en "Langage Machine"

Bien que cela ne présente aucun intérêt, il doit être possible de re-numéroter un programme BASIC à partir d'un programme en langage machine en faisant un JSR F14E après avoir basculé sur la RAM overlay. 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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
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 (sans incrémenter TXTPTR =

			CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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
C45Ca	84 01	STY 01	
C45Ea	A5 9C	LDA 9C	
C460a	A4 9D	LDY 9D	
C462a	85 02	STA 02	copie FINBAS (9C/9D) en 02/03
C464a	84 03	STY 03	
C466a	20 A8 C4	JSR C4A8	entrée réelle de la routine RENUM proprement dite
C469a	4C B4 E0	JMP E0B4	restaure liens de lignes et pointeurs puis termine

Recherche l'adresse d'une ligne BASIC

C46Ca	86 33	STX 33	sauve XA en 33/34
C46Ea	85 34	STA 34	
C470a	A5 00	LDA 00	XA = adresse du lien en 00/01
C472a	A6 01	LDX 01	
C474a	4C DC C6	JMP C6DC	recherche adresse CE/CF de la ligne BASIC XA

Remet octet en place et incrémente pointeurs source et cible

C477a	91 F4	STA (F4),Y	sauve octet lu à TXTPTR selon adresse en F4/F5
C479a	E6 E9	INC E9	
C47Ba	D0 02	BNE C47F	incrémente TXTPTR (source = bloc haut)
C47Da	E6 EA	INC EA	
C47Fa	E6 F4	INC F4	
C481a	D0 0D	BNE C490	incrémente F4/F5 (cible = bloc bas)
C483a	E6 F5	INC F5	
C485a	60	RTS	retourne avec Z = 0 car adresse cible jamais page zéro

Mise à jour du contenu de 08/09

```
C486a  98      TYA
C487a  18      CLC
C488a  65 08   ADC 08
C48Aa  85 08   STA 08      08/09 = 08/09 + Y
C48Ca  90 02   BCC C490
C48Ea  E6 09   INC 09
C490a  60      RTS
```

Mise à jour du contenu de 0A/0B

```
C491a  98      TYA
C492a  18      CLC
C493a  65 0A   ADC 0A
C495a  85 0A   STA 0A      0A/0B = 0A/0B + Y
C497a  90 F7   BCC C490
C499a  E6 0B   INC 0B
C49Ba  60      RTS
```

Mise à jour du pointeur cible F4/F5

```
C49Ca  18      CLC
C49Da  A9 05   LDA #05
C49Fa  65 F4   ADC F4      F4/F5 = F4/F5 + 5
C4A1a  85 F4   STA F4      (en-tête de ligne = 5 octets)
C4A3a  90 EB   BCC C490
C4A5a  E6 F5   INC F5
C4A7a  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 Re-numéroter "TR" qui sont suivis d'un **N**uméro de ligne à Re-numéroter "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 correspondante

- Dernière ligne à re-numéroter = 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° correspondant

Déplace le programme BASIC vers le haut sous HIMEM

```
C4A8a  78        SEI        interdit les interruptions
C4A9a  A4 01     LDY 01
C4ABa  A6 00     LDX 00
C4ADa  D0 01     BNE C4B0
C4AFa  88        DEY        calcule DEBBAS - 1
C4B0a  CA        DEX        (qui vise le #00 de début de la première ligne)
C4B1a  86 CE     STX CE
C4B3a  84 CF     STY CF     et copie cette valeur en CE/CF
C4B5a  86 08     STX 08     (adresse du premier 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 du 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 première 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 première 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 susceptibles 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
C521a	C9 C9	CMP #C9	est-ce le token "THEN"?
C523a	F0 08	BEQ C52D	si oui, continue en C52D
C525a	C9 9A	CMP #9A	est-ce le token "RESTORE"?
C527a	F0 04	BEQ C52D	si oui, continue en C52D
C529a	C9 98	CMP #98	est-ce le token "RUN"?
C52Ba	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 Re-numéroter)?

Le premier 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.

C52Da	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
C530a	B1 E9	LDA (E9),Y	lit octet à TXTPTR + Y (octet suivant le "TR")
C532a	F0 B6	BEQ C4EA	reprend en C4EA si nul (fin ligne atteinte)
C534a	C9 20	CMP #20	est-ce un espace? (à négliger: sans signification)
C536a	F0 F5	BEQ C52D	si oui, OK, reboucle en C52D
C538a	C9 30	CMP #30	est-ce un code < #30? (#30 = début des chiffres)
C53Aa	90 D4	BCC C510	si oui, pas bon, reprend en C510 (ex: parenthèse)
C53Ca	C9 97	CMP #97	est-ce le token "GOTO"?
C53Ea	F0 ED	BEQ C52D	si oui, OK, reboucle en C52D
C540a	C9 9B	CMP #9B	est-ce le token "GOSUB"?
C542a	F0 E9	BEQ C52D	si oui, OK, reboucle en C52D
C544a	C9 3A	CMP #3A	est-ce un code >= #3A? (#39 = fin des chiffres)
C546a	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).

C548a	C8	INY	un chiffre a été trouvé, incrémente index Y. TXTPTR pointe sur le premier chiffre du n° de ligne à mettre à jour
C549a	B1 E9	LDA (E9),Y	lit octet suivant à TXTPTR + Y
C54Ba	F0 1A	BEQ C567	si nul, OK continue en C567 (fin de ligne atteinte)
C54Da	C9 27	CMP #27	est-ce un “ ‘ “? (REM, voir "non documenté" au début)
C54Fa	F0 16	BEQ C567	si oui, OK continue en C567 (c'est valide!)
C551a	C9 30	CMP #30	est-ce un code < #30? (#30 = début des chiffres)
C553a	90 06	BCC C55B	si oui, continue en C55B (BCC C55F aurait été mieux!)
C555a	C9 3A	CMP #3A	est-ce un code < #3A? (#39 = fin des chiffres)
C557a	90 EF	BCC C548	si oui, reboucle en C548 (c'est encore un chiffre)
C559a	F0 0C	BEQ C567	si c'est un ":", OK continue en C567
C55Ba	C9 C8	CMP #C8	est-ce le token "ELSE"?
C55Da	F0 08	BEQ C567	si oui, OK continue en C567
C55Fa	C9 2C	CMP #2C	est-ce une ","?
C561a	F0 04	BEQ C567	si oui, OK continue en C567
C563a	C9 20	CMP #20	est-ce un espace? si oui, OK continue en C567
C565a	D0 83	BNE C4EA	sinon, reboucle en C4EA (autres cas invalides)

Le code suivant le "TR" a été trouvé

C567a	48	PHA	si oui, empile l'octet qui suit le dernier chiffre
C568a	A9 00	LDA #00	et le remplace par un #00 dans le bloc du haut
C56Aa	91 E9	STA (E9),Y	à ce moment, Y = nombre de chiffres du n° de ligne
C56Ca	A6 F4	LDX F4	
C56Ea	8A	TXA	le pointeur cible F4/F5 est copié selon l'adresse
C56Fa	A0 03	LDY #03	en F8/F9 + 3 (et suivante car 2 octets)
C571a	91 F8	STA (F8),Y	
C573a	C8	INY	
C574a	A5 F5	LDA F5	puis en F8/F9 qui garde donc en memmoire le point
C576a	91 F8	STA (F8),Y	où l'on en est resté dans le bloc du bas
C578a	86 F8	STX F8	
C57Aa	85 F9	STA F9	

Voici 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 premier "NR" sera gardé en (F3) + 3 c'est à dire en F6/F7 (voir initialisation en C4E2). L'adresse du deuxième "NR" sera gardée dans les 2 derniers octets du premier 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 deuxième 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 premier 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 = premier 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 premier 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 premier "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 correspondante

```
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
C5D9a  C8        INY
C5DAa  B1 F4     LDA (F4),Y
```

```
C5DCa  20 6C C4  JSR C46C     recherche l'adresse CE/CF de la ligne BASIC n° XA
C5DFa  A0 01     LDY #01
C5E1a  A5 CE     LDA CE
C5E3a  91 F4     STA (F4),Y   place adresse selon F4/F5 + 1 (et suivant car 2 octets)
C5E5a  C8        INY
C5E6a  A5 CF     LDA CF
C5E8a  91 F4     STA (F4),Y
C5EAa  C8        INY
C5EBa  B1 F4     LDA (F4),Y    lit les deux octets suivants
C5EDa  AA        TAX           (lien = adresse du "TR" suivant)
C5EEa  C8        INY
C5EFa  B1 F4     LDA (F4),Y
C5F1a  D0 DB     BNE C5CE      reboucle en C5CE s'il y en a encore à mettre à jour
```

Dernière ligne à re-numéroter = fin provisoire du programme BASIC

```
C5F3a  AE EA C6  LDX C6EA      XA = DERLGN (n° de la dernière ligne à re-numéroter)
C5F6a  AD EB C6  LDA C6EB
C5F9a  20 6C C4  JSR C46C      cherche adresse en CE/CF du lien de ligne BASIC n° XA
C5FCa  A0 01     LDY #01
C5FEa  B1 CE     LDA (CE),Y    empile le HH du lien de cette ligne et le remplace
C600a  48        PHA           par #00, marquant ainsi une fausse fin de programme
C601a  A9 00     LDA #00
C603a  91 CE     STA (CE),Y
C605a  AE E8 C6  LDX C6E8      XA = PRELGN (n° de la première ligne à re-numéroter)
C608a  AD E9 C6  LDA C6E9
C60Ba  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

```
C60Ea  A0 03     LDY #03
C610a  AD E5 C6  LDA C6E5
C613a  91 CE     STA (CE),Y    copie NEWNUM à la place de l'ancien n°
C615a  88        DEY
C616a  AD E4 C6  LDA C6E4
C619a  91 CE     STA (CE),Y
C61Ba  18        CLC
C61Ca  6D E6 C6  ADC C6E6
C61Fa  8D E4 C6  STA C6E4
C622a  AD E5 C6  LDA C6E5      calcule NEWNUM = NEWNUM + NEWPAS
C625a  6D E7 C6  ADC C6E7
C628a  8D E5 C6  STA C6E5
C62Ba  B0 10     BCS C63D      continue en C63D si dépassement de capacité
C62Da  A0 00     LDY #00
C62Fa  B1 CE     LDA (CE),Y
C631a  AA        TAX           XA = adresse de la ligne BASIC suivante
C632a  C8        INY
C633a  B1 CE     LDA (CE),Y
```

```
C635a  F0 18      BEQ C64F      continue en C64F si HH nul
C637a  86 CE      STX CE        (dernière ligne à re-numéroter atteinte)
C639a  85 CF      STA CF        mise à jour de CE/CF avec adresse ligne suivante
C63Ba  D0 D1      BNE C60E      si HH du lien non nul, rebouclage forcé en C60E
```

Dépassement: re-numérotation impossible

```
C63Da  68         PLA           élimine 1 octet de la pile (celui du HH inutile)
C63Ea  20 0D C4   JSR C40D      re-initialise NEWNUM, NEWPAS, PRELGN et DERLGN
C641a  A0 03      LDY #03       index pour écrire 4 octets
C643a  8A         TXA           force A à zéro (sorti du sous-programme C40D avec X = #00)
C644a  99 E4 C6   STA C6E4,Y    force NEWNUM et NEWPAS à zéro
C647a  88         DEY           vise le précédant
C648a  10 FA      BPL C644      reboucle tant qu'il en reste à mettre à zéro
C64Aa  EE E6 C6   INC C6E6      NEWPAS passe à 1
C64Da  D0 A4      BNE C5F3      rebouclage forcé vers C5F3 (RENUM de sauvetage)
```

Dernière ligne à re-numéroter atteinte

```
C64Fa  68         PLA           récupère un octet sur la pile et l'écrit selon
C650a  91 CE      STA (CE),Y    adresse en CE/CF + Y (c'est à dire le remet à sa place)
C652a  A5 F7      LDA F7        teste F7
C654a  F0 42      BEQ C698      si nul (il n'y a pas de "TR"), continue en C698,
C656a  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
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 caractères
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 relocation 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 relocation 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
C6C8a	20 91 C4	JSR C491	mise à jour 0A/0B = 0A/0B + Y
C6CBa	4C 98 C6	JMP C698	reprise forcée en C698

Voilà, c'est fini

C6CEa	88	DEY	dépile 2 octets de la pile
C6CFa	88	DEY	(lien suivant qui n'existe pas)
C6D0a	58	CLI	autorise les interruptions
C6D1a	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

CE/CF	premier octet du bloc à déplacer,
C9/CA	dernier octet du bloc à déplacer,
C7/C8 et AY	adresse cible du bloc (attention: haut du nouveau bloc). Retourne avec C7/C8 pointant sur le nouveau début du bloc - #100.

C6D4a	20 D8 D5	JSR D5D8	XROM Exécute à partir de la RAM une routine ROM
C6D7a	FB C3 F7 C3		adresse ROM 1.0 adresse ROM 1.1
C6DBa	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 premier octet de lien)

C6DCa	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
C6DFa	E8 C6 BD C6		adresse ROM 1.0 adresse ROM 1.1
C6E3a	60	RTS	

Table "RENUM"

C6E4a	00 00	NEWNUM	n° de ligne pour commencer la re-numérotation
C6E6a	00 00	NEWPAS	"pas" pour incrémenter les n° de lignes
C6E8a	00 00	PRELGN	n° de la première ligne à re-numéroter
C6EAa	00 00	DERLGN	n° de la dernière ligne à re-numéroter

EXÉCUTION DE LA COMMANDE SEDORIC DELETE

Rappel de la syntaxe

DELETE (DELPRE)-(DELDER)
DELPRE est le numéro de la première ligne à supprimer (première ligne du programme si ce paramètre est omis). DELDER est le numéro 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

F2/F3	DELPRE	première ligne à supprimer
33/34	DELDER	dernière ligne à supprimer
F4/F5		longueur de la zone à supprimer = nombre d'octets à supprimer
CE/CF		pointeur source dans le bloc haut
F2/F3		pointeur cible dans le bloc bas

Informations non documentées

Contrairement à ce qu'affirme le manuel, lorsque le "-" est omis, DELETE n'équivaut 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 "Langage Machine"

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 RAM overlay. 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

C6ECa	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
C6EFa	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
C6F2a	A5 CE	LDA CE	place l'adresse de cette ligne dans F2/F3 (DELPRE)
C6F4a	85 F2	STA F2	(astuce douteuse: pour économiser un LDA CF,
C6F6a	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 puis D3A1/RAM overlay 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
C708a	E6 33	INC 33	et une "SYNTAX_ERROR" si le caractère trouvé est non numérique
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
C711a  A5 CE     LDA CE             dernière ligne à supprimer
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
```

EXÉCUTION DE LA 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 catastrophes. 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
F4/F5	FINBLOC	adresse de l'octet qui suit le dernier octet du bloc à déplacer
F6/F7	DEBCIBL	nouvelle adresse de début du bloc mémoire
F8		complément à 2 de Y, le nombre d'octets de la page incomplète
F9		nombre de pages entières à transférer

Non documenté dans le manuel

Même si le manuel indique qu'un MOVE malencontreux peut écraser la RAM overlay, il cache l'essentiel, à savoir que l'on peut utiliser MOVE pour travailler sur la RAM overlay. 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 "Langage Machine"

Voilà une routine bien précieuse dans les programmes en langage machine. Cependant, il ne semble pas très simple d'utiliser directement la commande MOVE de SEDORIC, 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

C756a	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)
C759a	84 F2	STY F2	place le résultat YA en F2/F3 (DEBBLOC)
C75Ba	85 F3	STA F3	
C75Da	20 2C D2	JSR D22C	D067/ROM exige une "," et place TXTPTR sur l'octet suivant
C760a	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)
C763a	84 F4	STY F4	place le résultat YA en F4/F5 (FINBLOC)
C765a	85 F5	STA F5	
C767a	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C76Aa	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)
C76Da	84 F6	STY F6	place le résultat YA en F6/F7 (DEBCIBL)
C76Fa	85 F7	STA F7	

Calcule le nombre de pages entières à transférer

C771a	08	PHP	sauvegarde les indicateurs 6502
C772a	78	SEI	interdit les interruptions
C773a	38	SEC	
C774a	A5 F4	LDA F4	calcule:
C776a	E5 F2	SBC F2	YX = F4/F5 - F2/F3 = LNGBLOC longueur du bloc
C778a	A8	TAY	ou X = HH est le nombre de pages entières
C779a	A5 F5	LDA F5	et Y le nombre d'octets de la page incomplète
C77Ba	E5 F3	SBC F3	
C77Da	AA	TAX	
C77Ea	90 48	BCC C7C8	erreur si FINBLOC < DEBBLOC (LNGBLOC négatif)
C780a	86 F9	STX F9	F9 = nombre de pages entières à transférer

Evalue la direction du MOVE à utiliser

C782a	A5 F6	LDA F6	calcule F6/F7 - F2/F3 pour
C784a	C5 F2	CMP F2	évaluer si DEBCIBL < DEBBLOC
C786a	A5 F7	LDA F7	(cas d'un MOVE descendant sans problème)
C788a	E5 F3	SBC F3	
C78Aa	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          FINBLOC
|                        |                |                |
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 SEDORIC 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:

DEBBLOC = FINBLOC - LNGBLOC soit F2/F3 = F4/F5 - YX qui s'écrit aussi F4/F5 - X - Y ou F4/F5 - X + complément 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	décrémente 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

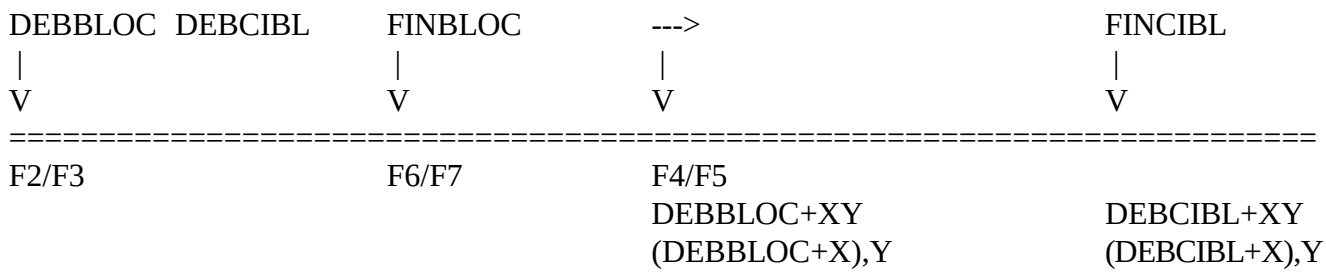
C7ABa	E8	INX	
C7ACa	B1 F4	LDA (F4),Y	lit octet selon source + Y
C7AEa	91 F6	STA (F6),Y	écrit octet selon cible + Y
C7B0a	C8	INY	indexe octet suivant
C7B1a	D0 F9	BNE C7AC	reboucle en C7AC tant que la page n'est pas finie

Transfère 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.



Explication: On ne peut opérer comme précédemment 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 LNGBLOC, 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 + LNGBLOC = 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

C7CBa	8A	TXA	X = nombre de pages entières à transférer
C7CCa	18	CLC	calcule F3 = F3 + X
C7CDa	65 F3	ADC F3	(les HH suffisent)

```
C7CFa  85 F3     STA F3
C7D1a  8A        TXA
C7D2a  18        CLC
C7D3a  65 F7     ADC F7      De même, calcule F7 = F7 + X
C7D5a  85 F7     STA F7
```

Transfère la page incomplète

```
C7D7a  E8        INX         nombre total de pages (partielle + entières)
                             La première page copiée sera partielle puisque Y < 256.
C7D8a  88        DEY         ce qui explique pourquoi l'octet de FINBLOC est exclu du transfert.
                             Il manque un INY avant le début de cette boucle.
C7D9a  B1 F2     LDA (F2),Y  lit octet selon le pointeur source + Y
C7DBa  91 F6     STA (F6),Y  écrit octet selon le pointeur cible + Y
C7DDa  98        TYA         teste Y
C7DEa  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.

```
C7E0a  C6 F3     DEC F3      indexe page précédente en lecture
C7E2a  C6 F7     DEC F7      indexe page précédente en écriture
C7E4a  CA        DEX         décrémente le nombre de pages à déplacer
C7E5a  F0 0F     BEQ C7F6    termine en C7F6 s'il n'en reste plus
C7E7a  88        DEY
C7E8a  B1 F2     LDA (F2),Y  lit octet selon le pointeur source + Y
C7EAa  91 F6     STA (F6),Y  écrit octet selon le pointeur cible + Y
C7ECa  88        DEY
C7EDa  B1 F2     LDA (F2),Y  lit octet selon le pointeur source + Y
C7EFa  91 F6     STA (F6),Y  écrit octet selon le pointeur cible + Y
C7F1a  98        TYA         teste Y
C7F2a  D0 F3     BNE C7E7    reboucle en C7E7 tant que la page n'est pas finie
C7F4a  F0 EA     BEQ C7E0    reboucle en C7E0 pour indexer la page suivante
```

Fini

```
C7F6a  28        PLP         récupère les indicateurs 6502
C7F7a  60        RTS         et retourne (fin de MOVE)

C7F8a  4C B4 04  JMP 04B4    NMIRAM

C7FBa  84 00 00 00 00        5 octets semblent inutilisés = libres
```

BANQUE n°2: BACKUP

Cette BANQUE se trouve à partir du #47 (soixante et onzième) secteur de la disquette MASTER.

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

EXÉCUTION DE LA 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 l'ancien défaut de la commande INIT (version 1.0): lorsque le flag "D" n'est 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. BACKUP marche correctement avec les disquettes ne présentant pas ce défaut du à la version 1.0 de SEDORIC.

Autre bogue (plus commune celle-là): bien que les commandes SEDORIC 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 "Langage Machine"

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 RAM overlay. 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

C404b	20 DE DF	JSR DFDE	vérifie que l'on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"
C407b	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C40Ab	8C F9 C0	STY C0F9	n° du drive source
C40Db	F0 05	BEQ C414	saute les 2 instructions suivantes si fin des paramètres

C40Fb	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")
C411b	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay demande "TO" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C414b	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C417b	8C FA C0	STY C0FA	n° du drive cible

Formatage?

C41Ab	A2 06	LDX #06	indexe le message "CRLFFormat_TARGET_disc_(Y/N)**:**"
C41Cb	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C41Fb	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
C422b	10 FB	BPL C41F	reboucle tant qu'une touche n'a pas été pressée
C424b	20 A1 D3	JSR D3A1	XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
C427b	C9 59	CMP #59	est-ce un "Y"?
C429b	F0 08	BEQ C433	si oui, continue en C433 avec C = 1 (flag "format")
C42Bb	C9 1B	CMP #1B	est-ce "ESC"?
C42Db	D0 01	BNE C430	sinon, saute l'instruction suivante
C42Fb	60	RTS	si oui, simple RTS, c'est fini
C430b	18	CLC	flag "format" OFF
C431b	A9 4E	LDA #4E	lettre "N"
C433b	6E 74 C0	ROR C074	le b7 de C074 porte le flag "format"
C436b	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C439b	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C43Cb	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Monodrive?

C43Fb	AD F9 C0	LDA C0F9	compare n° drive source
C442b	CD FA C0	CMP C0FA	et n° drive cible
C445b	18	CLC	flag "monodrive" OFF par défaut (deux drives)
C446b	D0 01	BNE C449	saute l'instruction suivante s'il y a deux drives
C448b	38	SEC	flag "monodrive" ON (un seul drive)
C449b	6E 72 C0	ROR C072	le b7 de C072 porte le flag "monodrive"
C44Cb	30 18	BMI C466	si monodrive, continue en C466

Demande les deux disquettes si "monodrive" OFF

C44Eb	A2 01	LDX #01	indexe le message "LOAD_DISCS_FOR_BACKUP_FROM_"
C450b	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C453b	AD F9 C0	LDA C0F9	n° du drive source
C456b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
C459b	A2 02	LDX #02	indexe le message "_TO_"
C45Bb	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C45Eb	AD FA C0	LDA C0FA	n° du drive cible
C461b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
C464b	D0 14	BNE C47A	suite forcée en C47A avec C = 0

Demande la disquette source si "monodrive" ON

C466b	A2 03	LDX #03	indexe le message "CRLFLOAD_SOURCE_DISC_"
C468b	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C46Bb	A2 00	LDX #00	indexe le message "IN_DRIVE_"
C46Db	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C470b	AD F9 C0	LDA C0F9	n° du drive source
C473b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
C476b	38	SEC	force C = 1 ("la disquette source est en place")
C477b	6E 73 C0	ROR C073	force à 1 le b7 de C073 (flag "source in drive")

Lit le format de la disquette source

C47Ab	A2 05	LDX #05	indexe le message "CRLFAND_PRESS_RETURN_"
C47Cb	20 64 D3	JSR D364	XAFSC affiche (X+1) ème 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 premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").

Effectue un NEW

C48Eb	A9 00	LDA #00	
C490b	A0 01	LDY #01	force à 0 l'octet de poids fort du premier 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 de la version 1.0 de SEDORIC (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 dixième octet (n°#09) du secteur de bitmap (deuxième secteur, piste 20). Par exemple, pour une disquette formatée en 42 pistes, cet octet indique #2A (**0**010 1010) et il faudra le mettre à #AA (**1**010 1010).

C497b	A9 00	LDA #00	n° de la première piste de la première face
C499b	2C 09 C2	BIT C209	teste b7 du dixième 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 première 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 première 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 première incrémentation Y passera à #00 au lieu de #01
C4BFb	AC 07 C2	LDY C207	huitième 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 tampon 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.

C4E6b	8D FD C0	STA C0FD	sauve HH de la taille du tampon de BACKUP en C0FD
C4E9b	8D B0 C5	STA C5B0	sauve HH de la taille du tampon de BACKUP en C5B0
C4ECb	2C 72 C0	BIT C072	teste si le flag "monodrive" est à zéro
C4EFb	10 1F	BPL C510	si oui (2 drives), continue en C510
C4F1b	2C 73 C0	BIT C073	sinon (monodrive), teste le flag "source in drive"
C4F4b	30 1A	BMI C510	continue en C510 si disquette source est en place. C'est le cas au premier 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

C4F6b	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C4F9b	A2 03	LDX #03	pour quatrième message externe "CRLFLOAD_SOURCE_DISC_"
C4FBb	98	TYA	teste Y (flag "écriture", à zéro si lecture)
C4FCb	48	PHA	empile Y temporairement
C4FDb	F0 01	BEQ C500	saute l'instruction suivante si Y = 0 (lecture)
C4FFb	E8	INX	indexe le message "CRLFLOAD_TARGET_DISC_"
C500b	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C503b	A2 05	LDX #05	indexe le message "CRLFAND_PRESS_RETURN_"
C505b	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C508b	68	PLA	
C509b	A8	TAY	récupère Y
C50Ab	20 69 D6	JSR D669	demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
C50Db	90 01	BCC C510	si "RETURN", saute l'instruction suivante
C50Fb	60	RTS	si "ESC", simple RTS, c'est fini

Formatage?

C510b	78	SEI	interdit les interruptions
C511b	B9 F9 C0	LDA C0F9,Y	n° du drive source si Y = #00 ou cible si Y = #01
C514b	8D 00 C0	STA C000	devient le n° du DRIVE actif
C517b	2C 74 C0	BIT C074	teste si le flag "formatage" est à zéro
C51Ab	10 12	BPL C52E	si oui (pas de formatage), continue en C52E
C51Cb	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.
C51Db	F0 0F	BEQ C52E	si oui (lecture), continue en C52E
C51Fb	4E 74 C0	LSR C074	sinon (écriture), force à zéro par avance le flag
C522b	20 41 C6	JSR C641	"formatage" et effectue le formatage, puis force
C525b	4E 73 C0	LSR C073	à 0 le flag "source in drive" (car cible en place)
C528b	A0 00	LDY #00	pour n° de piste de départ
C52Ab	8C 01 C0	STY C001	piste n° #00 devient PISTE active

```
C52Db  C8         INY          Y = #01 (flag "écriture", à 1 si écriture)
```

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

```
C52Eb  A9 00      LDA #00
C530b  8D 03 C0   STA C003     RWBUF = 0600 adresse du tampon de BACKUP
C533b  A9 06      LDA #06
C535b  8D 04 C0   STA C004
C538b  4E 73 C0   LSR C073     force à zéro le flag "source in drive"
```

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

```
C53Bb  98         TYA          teste si Y est nul
C53Cb  F0 05      BEQ C543     si oui (lecture) continue en C543, sinon...
C53Eb  20 A4 DA   JSR DAA4     XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C541b  F0 03      BEQ C546     et saute l'instruction suivante
C543b  20 73 DA   JSR DA73     XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
```

Indexe le secteur suivant et reboucle si nécessaire

```
C546b  EE 04 C0   INC C004     page suivante (incrémente le HH de RWBUF)
C549b  AE 02 C0   LDX C002     n° du dernier SECTEUR actif
C54Cb  EC 07 C2   CPX C207     teste si la valeur maximale est atteinte
C54Fb  D0 05      BNE C556     sinon, saute les deux instructions suivantes
C551b  EE 01 C0   INC C001     indexe la PISTE suivante
C554b  A2 00      LDX #00      reset le n° de secteur (à 1 en début de boucle)
C556b  E8         INX          indexe le secteur suivant
C557b  8E 02 C0   STX C002     qui devient le SECTEUR actif
C55Ab  CE FD C0   DEC C0FD     décrémente le HH de la taille du tampon de BACKUP
C55Db  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_completeCRLF"
C599b  4C 64 D3    JMP D364      XAFSC affiche (X+1) ème 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)
       (Le numéro d'ordre X est indiqué à gauche sous l'adresse)

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**

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

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)
C647b	8D A6 C6	STA C6A6	C6A6 = #00 si une face et #01 si deux faces
C64Ab	A9 A7	LDA #A7	AY = adresse C6A7 de la chaîne:
C64Cb	A0 C6	LDY #C6	"CRLFLFFormating_Side_0_Track_00"
C64Eb	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
C651b	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
C654b	4E A5 C6	LSR C6A5	rétablit nombre de pistes/face (sans nombre face)
C657b	20 3C C7	JSR C73C	formate la première face (car C vaut toujours 0)
C65Ab	AD A6 C6	LDA C6A6	teste si une seule face
C65Db	F0 0B	BEQ C66A	si oui, continue en C66A
C65Fb	A9 C4	LDA #C4	sinon, AY = C6C4 pour chaîne
C661b	A0 C6	LDY #C6	"<- <- <- <- <- <- <- <- <- <-1_Track_00"
C663b	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
C666b	38	SEC	C = 1 pour deuxième face
C667b	20 3C C7	JSR C73C	formate la deuxième face
C66Ab	A9 D9	LDA #D9	AY = adresse C6D9 pour chaîne
C66Cb	A0 C6	LDY #C6	"LFLFCRCTRL/DDoneBRK"
C66Eb	4C 37 D6	JMP D637	AFSTR affiche chaîne d'adresse AY et retourne

Table de formatage (C671 à C69A)

C671b	28 **4E** 0C **00** 03 **F6** 01 **FC** 28 **4E** FF	(soit #60 = 96 octets au total)
C67Cb	0C **00** 03 **F5**	(soit 15 octets)

C680b 01 **FE** 01 **00** 01 **00** 01 **00** 01 **01** 01 **F7** (FE pp ff ss 01 CRC, soit 6 octets)
C68Cb 16 **4E** 0C **00** 03 **F5** (soit 37 octets)
C692b 01 **FB** 00 **00** 01 **F7** (soit 258 octets)
C698b 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

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

Autres messages (C6A7 à C6EF)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

C6A7b 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**
01 CRLFLFFormating_Side_0_Track_00**BRK**

C6C4b 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**

C6D9b 0A 0A 0D 11 44 6F 6E 65 **00**
03 LFLFCRCTRL/QDone**BRK**

Rappel: **BRK** = pour marquer la présence d'un #00 à la fin de certains messages
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

De C6DD à C6EF, le message "Formating complete" a été raccourci en "Done" par Ray, ce qui permet de dégager 14 octets pour insérer un nouveau sous-programme de débogage appelé à partir de C7B2. Les 18 octets différents sont indiqués en gras.

C6DDb **44** 6F **6E 65 00** "Done"
C6E2b **A9 48** LDA #48 code de "PHA"
C6E4b **8D 15 D0** STA D015 inséré dans la routine XRWTS à la place d'un RTS
C6E7b **20 CD CF** JSR CFCD appel à la routine XRWTS modifiée
C6EAb **A9 60** LDA #60 code de "RTS"

C6ECb	**8D 15 D0**	STA D015	restauration de l'état d'origine de la routine CFCD
C6EFb	**60**	RTS	et retour

Ce sous-programme est appelé en C7B2 de la BANQUE n°2. La routine XRWTS n'est modifiée que pour son utilisation par la commande BACKUP.

J'ai conservé dans la version 3.0 de SEDORIC cette modification que Ray a introduite dans sa V2.1 de SEDORIC, par respect pour son génie, mais elle entraîne le plantage de la commande BACKUP sous STRATORIC. J'ai donc été amené à la neutraliser dans STRATORIC V3.0 (voir en C7B2 de la BANQUE n°2).

En conséquence, pour effectuer un BACKUP de disquette V2.0, 2.1 ou 3.0 sous STRATORIC, lorsque votre TELESTRAT réclame une disquette MASTER, il faut lui fournir impérativement une STRATORIC V3.0 (ou à défaut une SEDORIC V2.0) sous peine de plantage.

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

C6F0b	AD 9F C6	LDA C69F	#10 si 18 ou 19, #70 si 16 ou 17 secteurs/piste
C6F3b	AC A0 C6	LDY C6A0	#98 HH de l'adresse du tampon de formatage
C6F6b	85 0A	STA 0A	0A/0B = #9810 ou #9870 = pointeur dans ce tampon
C6F8b	84 0B	STY 0B	(c'est l'adresse du premier n° de piste)
C6FAb	A2 00	LDX #00	compteur du nombre de secteurs déjà préparés
C6FCb	A0 00	LDY #00	index écriture dans chaque zone de préparation
C6FEb	AD 01 C0	LDA C001	n° de PISTE active
C701b	29 7F	AND #7F	dont on force à zéro le b7 (flag nombre de faces)
C703b	91 0A	STA (0A),Y	écrit le n° de piste #pp au pointeur + Y
C705b	C8	INY	position suivante
C706b	A5 F8	LDA F8	A = n° de face
C708b	91 0A	STA (0A),Y	écrit le n° de face #ff au pointeur + Y
C70Ab	C8	INY	position suivante
C70Bb	A5 F7	LDA F7	A = n° de secteur
C70Db	18	CLC	prépare une addition
C70Eb	69 01	ADC #01	A = n° de secteur + #01
C710b	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	correspondant 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 premier 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 SEDORIC 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] (SEDORIC, 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 SEDORIC, 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 de SEDORIC, #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 SEDORIC).

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 SEDORIC.

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 SEDORIC, 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 SEDORIC, 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 SEDORIC, 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)

C73Cb	08	PHP	sauvegarde les indicateurs 6502 dont C
C73Db	08	PHP	idem une deuxième fois
C73Eb	AD A3 C6	LDA C6A3	
C741b	85 F6	STA F6	F6 = nombre de secteurs par piste
C743b	8D A4 C6	STA C6A4	
C746b	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:
C749b	A0 0C	LDY #0C	Y = #0C (soit 12, valeur pour 19 secteurs/piste)
C74Bb	C9 13	CMP #13	teste si A >= #13 (en fait si A = 19, valeur maximale)
C74Db	B0 08	BCS C757	si oui, continue en C757 (OK pour Y)
C74Fb	A0 1E	LDY #1E	sinon, Y = #1E (30, valeur pour 18 secteurs/piste)

C751b	C9 12	CMP #12	teste si A >= #12 (en fait si A = 18)
C753b	B0 02	BCS C757	si oui, continue en C757 (OK pour Y)
C755b	A0 28	LDY #28	sinon... pour **SEDORIC, toutes versions**: Y = #28 (soit 40, valeur pour A = 16 ou 17)
C755b	A0 2F	LDY #2F	ou pour **STRATORIC, toutes versions**: Y = #2F (soit 47, valeur pour A = 16 ou 17)

Cette variante a été introduite dans STRATORIC V1.0 par Fabrice Broche et maintenue par la suite (légère modification des caractéristiques de formatage).

C757b	8C 98 C6	STY C698	sauve Y en C698 (index de base)
C75Ab	18	CLC	
C75Bb	98	TYA	
C75Cb	69 3C	ADC #3C	C6A1 = Y + #3C (index élargi)
C75Eb	8D A1 C6	STA C6A1	
C761b	AD A5 C6	LDA C6A5	
C764b	85 F5	STA F5	F5 = nombre de pistes par face
C766b	AD 9B C6	LDA C69B	
C769b	AC 9C C6	LDY C69C	AY = #9800 (adresse présente en C69B/C69C)
C76Cb	85 0A	STA 0A	0A/0B = #9800 (adresse pour préparer en RAM
C76Eb	84 0B	STY 0B	une piste complète avec ses "gaps")
C770b	8D 03 C0	STA C003	RWBUF = #9800 (adresse du tampon en RAM
C773b	8C 04 C0	STY C004	qui sera envoyé sur la disquette)
C776b	28	PLP	récupère les indicateurs 6502 dont C
C777b	A9 00	LDA #00	force à 0 le registre A
C779b	AA	TAX	force X à 0 (index de lecture dans la table C671)
C77Ab	A8	TAY	force Y à 0 (index d'écriture dans le tampon 9800)
C77Bb	2A	ROL	force le b0 de A selon C donc A = C
C77Cb	85 F8	STA F8	F8 = #00 si première face ou #01 si deuxième face
C77Eb	28	PLP	récupère les indicateurs 6502 dont C qui passe
C77Fb	6A	ROR	dans b7 de A dont l'ancien b0 est éliminé
C780b	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!)
C783b	86 F7	STX F7	F7 = #00 n° du premier secteur
C785b	AD A3 C6	LDA C6A3	A = nombre de secteurs par piste
C788b	C9 12	CMP #12	pour **SEDORIC, toutes versions**: teste si A >= #12 (c'est à dire, vaut 18 ou 19)
C788b	C9 11	CMP #11	et pour **STRATORIC, toutes versions**: teste si A >= #11 (c'est à dire, vaut 17 ou 18)

Cette variante a été introduite dans STRATORIC V1.0 par Fabrice Broche et maintenue par la suite (légère modification des caractéristiques de formatage).

C78Ab	B0 06	BCS C792	si oui, continue en C792

Elabore un début de piste dans le tampon de formatage

C78Cb	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 première 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

C78Fb	A9 70	LDA #70	valeur pour 16 ou 17 secteurs par piste
C791b	2C A9 10	BIT 10A9	et continue en C794
C792b	A9 10	LDA #10	valeur pour 18 ou 19 secteurs par piste
C794b	8D 9F C6	STA C69F	sauve en C69F la valeur retenue

Elabore le reste de la piste dans le tampon de formatage

C797b	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 deuxième 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

En C7B2, Ray a remplacé le JSR CFCD (XRWTS) par un JSR C6E2 (routine corrective). Cette correction

a été annulée dans le cas de STRATORIC V3.0 pour raison de plantage de la commande BACKUP (voir explications ci-dessus en C6E2). Le JSR CFCD a donc été rétabli en attendant de comprendre la raison de cette incompatibilité, qui n'existe pas avec SEDORIC.

C7B2b	20 **E2 C6**	JSR C6E2	SEDORIC V 3.0: modification transitoire et appel à XRWTS
C7B2b	20 CD CF	JSR CFCD	STRATORIC V3.0: 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 sous-programme)
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 récupérée dès que possible pour le tampon de BACKUP.

2) L'ancienne bogue de INIT (SEDORIC V1.0) a des effets désastreux sur la commande BACKUP qui ne peut deviner si la disquette à copier est simple ou double face. Attention donc avec les anciennes disquettes.

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

C7F6b E6 0B INC 0B
C7F8b C6 0C DEC 0C
C7FAb D0 F5 BNE C7F1
C7FCb F0 E5 BEQ C7E3
C7FEb 60 RTS
C7FFb 00 BRK

BANQUE n°3: SEEK, CHANGE et MERGE

Cette BANQUE se trouve à partir du #4C (soixante seizième) secteur de la disquette MASTER.

C400c	DC C7	EXTER	adresse des messages d'erreur externes
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_ERROR"
C410c	A2 1A	LDX #1A	"INVALID_STRING_ERROR"
C412c	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EXÉCUTION DE LA 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'occurrences 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'occurrences 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. SEDORIC 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 chaîne 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" reconnaît 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 SEDORIC 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 "Langage Machine"

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 RAM overlay. 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

C415c	08	PHP	sauvegarde les indicateurs 6502
C416c	A9 00	LDA #00	
C418c	85 F5	STA F5	force à zéro F5, F6 et DEFAFF
C41Ac	85 F6	STA F6	(code ASCII devant les nombres décimaux)
C41Cc	8D 4C C0	STA C04C	
C41Fc	46 F7	LSR F7	force à zéro le b7 de F7 (flag ",S")
C421c	46 F8	LSR F8	force à zéro le b7 de F8 (flag ",M")
C423c	38	SEC	
C424c	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.

C426c	38	SEC	
C427c	6E F2 02	ROR 02F2	force à 1 b7 de 02F2 (LIST retourne à l'appelant)
C42Ac	28	PLP	récupère les indicateurs 6502 dont Z
C42Bc	D0 11	BNE C43E	continue en C43E s'il y a des paramètres

Il n'y a pas de paramètre

C42Dc	46 F9	LSR F9	pas de paramètre, force à 0 le b7 de F9
C42Fc	AD 8B C0	LDA C08B	LNG de chaîne est mise a jour avec SEEK précédent
C432c	85 F4	STA F4	F4 = C08B = longueur de la chaîne à chercher
C434c	78	SEI	interdit les interruptions
C435c	AC 89 C0	LDY C089	
C438c	AE 8A C0	LDX C08A	YX = C089/C08A = DEBBAS
C43Bc	4C 86 C4	JMP C486	suite forcée en C486

Il y a des paramètres

C43Ec	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
C441c	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
C444c	85 F4	STA F4	F4 = LNG, longueur de la chaîne à chercher
C446c	8D 8B C0	STA C08B	C08B = LNG, longueur de la chaîne à chercher
C449c	A8	TAY	teste LNG
C44Ac	F0 58	BEQ C4A4	continue en C4A4 (en fait en C500) si nulle
C44Cc	C0 50	CPY #50	teste si LNG >= 80 caractères
C44Ec	B0 BD	BCS C40D	si oui, continue en C40D ("STRING_TOO_LONG_ERROR")
C450c	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_ERROR”)
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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
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 puis D3A1/RAM overlay 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 premier lien de première 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 début 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 première 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)
C4D5c	E6 F3	INC F3	
C4D7c	D0 E2	BNE C4BB	et reboucle en C4BB
C4D9c	C8	INY	si identiques, incrémente Y = nombre caractères identiques
C4DAc	C4 F4	CPY F4	la fin de la chaîne à chercher est-elle atteinte?
C4DCc	D0 DF	BNE C4BD	sinon, reboucle en C4BD sans remettre Y à zéro
C4DEc	E6 F5	INC F5	
C4E0c	D0 02	BNE C4E4	si oui, incrémente F5/F6
C4E2c	E6 F6	INC F6	(nombre d'occurrences de la chaîne dans le programme)
C4E4c	A5 F4	LDA F4	chaîne complète de longueur A trouvée
C4E6c	24 F7	BIT F7	teste si b7 de F7 est à 1 (paramètre ",S")
C4E8c	30 C8	BMI C4B2	si oui (ne pas afficher lignes), reboucle en C4B2
C4EAc	20 28 D6	JSR D628	sinon, affiche un espace, puis
C4EDc	20 B4 D1	JSR D1B4	C76C/ROM exécute la commande "LIST" simplifiée
C4F0c	A4 CE	LDY CE	
C4F2c	A6 CF	LDX CF	YX = adresse du lien suivant
C4F4c	24 F9	BIT F9	teste si b7 de F9 est à 1 ("arguments présents")
C4F6c	30 8E	BMI C486	si oui, reboucle en C486 (reprend début ligne suivante)
C4F8c	8C 89 C0	STY C089	sinon ("pas d'argument"),
C4FBc	8E 8A C0	STX C08A	sauve adresse du lien suivant en C089/C08A (pour futur SEEK)
C4FEc	58	CLI	autorise les interruptions
C4FFc	60	RTS	et sort de la commande SEEK

Fin du programme BASIC atteinte

C500c	24 F7	BIT F7	teste si b7 de F7 est à 0 (pas de paramètre ",S")
C502c	10 FA	BPL C4FE	si oui (afficher les lignes), CLI et RTS en C4FE
C504c	A5 F5	LDA F5	sinon,
C506c	A4 F6	LDY F6	AY = nombre d'occurrences trouvées
C508c	20 F5 D7	JSR D7F5	mise à jour de la variable SK (HH=Y et LL=A)
C50Bc	24 F8	BIT F8	teste si b7 de F8 est à 1 (paramètre ",M")
C50Dc	30 EF	BMI C4FE	si oui (ne rien afficher), CLI et RTS en C4FE
C50Fc	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C512c	A5 F5	LDA F5	
C514c	A4 F6	LDY F6	AY = nombre d'occurrences trouvées
C516c	20 53 D7	JSR D753	affichage en décimal sur 5 digits d'un nombre AY
C519c	58	CLI	autorise les interruptions
C51Ac	A2 00	LDX #00	indexe le message "_FoundsLFCR" (bogue: avec une faute!)
C51Cc	4C 64 D3	JMP D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C51Fc	4C 10 C4	JMP C410	"INVALID_STRING_ERROR"
C522c	4C 77 E9	JMP E977	"STRING_TOO_LONG_ERROR"

EXÉCUTION DE LA 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 occurrences 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 présent que dans la chaîne n°1, il sera remplacé par le caractère indiqué à la même place dans la chaîne n°2.

Variables utilisées

00		flag début de ligne BASIC
9C/9D	FINBAS	fin du programme BASIC
C7/C8	FINCIBL	dernier octet de l'emplacement qui recevra le bloc
C9/CA	FINBLOC	dernier octet du bloc qui sera déplacé vers le haut
CE/CF	DEBBLOC	premier octet du bloc qui sera déplacé vers le haut
F2/F3	SOURCE	nouveau début du bloc haut
F4/F5	CIBLE	point de redescente du bloc
F6		
F7	LNG2	longueur de la chaîne de remplacement
F8	LNG1	longueur de la chaîne cherchée
F9		différence de longueur LNG2 - LNG1 des 2 chaînes
C100/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 deuxième chaîne peut être vide, mais pas la première: CHANGE CHR$(157) TO "" élimine tous les REM. Ainsi la ligne 30 REM devenue vide sera éliminée. 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 déclenchée. 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 ignorez le token de la commande.

Attention TKEN marche seulement en mode programme. Dans les 3 cas, il n'est pas nécessaire de coder la deuxième partie (WAIT100): la chaîne de remplacement n'a pas besoin d'être “tokenisée”.

Utilisation en "Langage Machine"

Voilà une perspective des plus farfelues: modifier un programme BASIC à l'aide de la commande CHANGE à partir d'un programme en "Langage Machine"! Bon, mais j’ai trop de sympathie pour les bidouilleurs pour les laisser sur leur faim. Si la BANQUE n°3 est déjà en place, il suffira de basculer sur la RAM overlay, 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 puis D3A1/RAM overlay 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
C54Ac	A8	TAY	
C54Bc	84 F7	STY F7	F7 = longueur LNG2 de la nouvelle chaîne (n°2)
C54Dc	F0 14	BEQ C563	si nulle, continue en C563 (suite analyse syntaxe)
C54Fc	88	DEY	indexe chaîne n°2 (ira de #00 à longueur - 1)
C550c	C0 4E	CPY #4E	LNG2 est-elle >= 78? (re-bogue: #4F aurait été mieux!)
C552c	B0 CE	BCS C522	si oui, "STRING_TOO_LONG_ERROR"
C554c	B1 91	LDA (91),Y	sinon, lit octet de la chaîne
C556c	F0 C7	BEQ C51F	si octet est nul, termine en C51F "INVALID_STRING_ERROR"
C558c	99 00 C2	STA C200,Y	si octet OK, le copie dans BUF2 à la position Y
C55Bc	88	DEY	vise l'octet précédent (opère par la fin)
C55Cc	10 F6	BPL C554	reboucle en C554 tant qu'il en reste à copier
C55Ec	A9 00	LDA #00	force à zéro DEFAFF
C560c	8D 4C C0	STA C04C	(code ASCII affiché devant les nombres décimaux)
C563c	A5 F8	LDA F8	teste LNG2, longueur de la chaîne cherchée (n°1)
C565c	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

C567c	38	SEC	
C568c	A5 F7	LDA F7	F9 = F7 - F8 (LNG2 - LNG1)
C56Ac	E5 F8	SBC F8	
C56Cc	85 F9	STA F9	
C56Ec	A5 9C	LDA 9C	
C570c	A4 9D	LDY 9D	C9/CA (FINBLOC) = FINBAS
C572c	85 C9	STA C9	(dernier octet du bloc à déplacer vers le haut)
C574c	84 CA	STY CA	

```
C576c   A4 9B       LDY 9B
C578c   A6 9A       LDX 9A
C57Ac   D0 01       BNE C57D
C57Cc   88          DEY
C57Dc   CA          DEX         F4/F5 (CIBLE) = DEBBAS - 1
C57Ec   86 F4       STX F4      (point de retour ultérieur pour le bloc
C580c   84 F5       STY F5      vise le #00 placé devant la première ligne)
C582c   86 CE       STX CE      CE/CF (DEBBLOC) = DEBBAS - 1
C584c   84 CF       STY CF      (premier octet du bloc à transférer vers le haut)
C586c   A4 A3       LDY A3
C588c   A6 A2       LDX A2
C58Ac   D0 01       BNE C58D
C58Cc   88          DEY
C58Dc   CA          DEX         AY = C7/C8 (FINCIBL) = A2/A3 - 1
C58Ec   8A          TXA         (sous les chaînes BASIC)
C58Fc   85 C7       STA C7
C591c   84 C8       STY C8
C593c   20 5C D1    JSR D15C    C3F4/ROM décale un bloc mémoire vers le haut. En CE/CF adresse du
                                premier 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).
C596c   78          SEI         interdit les interruptions pendant le MOVE
C597c   A4 C8       LDY C8
C599c   A6 C7       LDX C7
C59Bc   C8          INY         F2/F3 (SOURCE) = C7/C8 + #100 (pointeur dans le
C59Cc   86 F2       STX F2      bloc haut, ajusté sur le premier #00)
C59Ec   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 bâtarde, 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 correspondant de la chaîne cherchée, met flag "ligne vide" à 0, décrémente le garde- fou F6, que l'on teste (éventuel “STRING_TOO_LONG_ERROR”), si OK reprend la recherche en C5D0

- en C5EF octet lu est identique à l'octet correspondant 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 chaque 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 éliminer 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)
C5DBc E0 5F CPX #5F est-ce un "£"? (joker pour CHANGE)
C5DDc F0 10 BEQ C5EF si oui, continue directement en C5EF
C5DFc 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)?
C5E2c F0 0B BEQ C5EF si oui, continue en C5EF (trouvé!)

L'octet descendu est différent de l'octet correspondant du modèle

C5E4c 46 00 LSR 00 sinon, force à zéro le b7 de 00 (la ligne de programme BASIC en cours contient au moins 1 octet)
C5E6c C6 F6 DEC F6 et décrémente F6
(nombre d'octets pouvant encore être ajoutés à la nouvelle ligne, avant que

```
                                  ça fasse trop long)
C5E8c  F0 4E      BEQ C638        continue en C638 lorsque F6 atteint zéro
C5EAc  20 5A C6   JSR C65A        sinon, incrémente F2/F3 (SOURCE) et F4/F5 (CIBLE)
C5EDc  D0 E1      BNE C5D0        reprise forcée en C5D0
```

L'octet descendu est identique à l'octet correspondant du modèle
Teste s'il y a assez de place pour écrire la nouvelle chaîne

```
C5EFc  C8         INY             trouvé! incrémente le nombre d'octets identiques
C5F0c  C4 F8      CPY F8          Y a t-il atteint la longueur de la chaîne 1
C5F2c  D0 DE      BNE C5D2        sinon, reboucle en C5D2
C5F4c  A5 F4      LDA F4
C5F6c  18         CLC
C5F7c  65 F9      ADC F9          si oui, calcule AX = F4/F5 + F9
C5F9c  A6 F5      LDX F5          (F9 = différence LNG2 - LNG1)
C5FBc  90 01      BCC C5FE
C5FDc  E8         INX
```

Teste si place en RAM pour écrire nouvelle chaîne à la place de l'ancienne

```
C5FEc  E4 F3      CPX F3          teste si X (HH bloc bas) < F3 (HH bloc haut)
C600c  90 06      BCC C608        si oui (OK), continue en C608
C602c  D0 50      BNE C654        si X > F3, continue en C654 (en fait C667)
C604c  C5 F2      CMP F2          si X = F3, teste si A >= F2 (LL blocs bas et haut)
C606c  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

```
C608c  A4 F7      LDY F7          OK, Y = LNG2, longueur de la nouvelle chaîne
C60Ac  F0 19      BEQ C625        continue en C625 si cette longueur est nulle
C60Cc  46 00      LSR 00          sinon, force à zéro le b7 de 00 (au moins 1 octet)
C60Ec  B9 00 C2   LDA C200,Y      lit un octet de la nouvelle chaîne dans BUF2, à la position Y
C611c  91 F4      STA (F4),Y      écrit cet octet dans le bloc du bas
C613c  C6 F6      DEC F6          et décrémente F6 (nombre octets encore ajoutables)
C615c  F0 21      BEQ C638        continue en C638 lorsque F6 atteint 0 (trop longue)
C617c  88         DEY             vise l'octet précédent (opère par la fin)
C618c  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

```
C61Ac  A5 F7      LDA F7          fini,
C61Cc  18         CLC
C61Dc  65 F4      ADC F4          mise à jour de CIBLE = F4/F5 + F7
C61Fc  85 F4      STA F4          (F7 = LNG2, longueur de la chaîne 2)
C621c  90 02      BCC C625
C623c  E6 F5      INC F5
C625c  A5 F8      LDA F8
C627c  18         CLC
```

```
C628c  65 F2      ADC F2      mise à jour de SOURCE = F2/F3 + F8
C62Ac  85 F2      STA F2      (F8 = LNG1, longueur de la chaîne 1)
C62Cc  90 A2      BCC C5D0
C62Ec  E6 F3      INC F3
C630c  D0 9E      BNE C5D0    reprise forcée en C5D0 car HH de SOURCE jamais nul
```

Fin du programme BASIC atteinte

```
C632c  91 F4      STA (F4),Y  écrit A = #00 dans HH du lien de fin de programme
C634c  58         CLI         autorise les interruptions
C635c  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

```
C638c  A0 00      LDY #00     force Y à zéro
C63Ac  20 60 C6   JSR C660    incrémente SOURCE
C63Dc  B1 F2      LDA (F2),Y  lit octet de ligne BASIC dans le bloc du haut
C63Fc  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 "LFCRLINE_:"
C646c  20 64 D3   JSR D364    XAFSC affiche (X+1) ème 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	

EXÉCUTION DE LA 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 insérée avant 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 proprement dites à copier (longueur de la ligne)
F8/F9	PTRHAUT	pointeur dans le bloc du haut (programme complémentaire)
026A		indicateurs du status console
C016		flag "BANQUE changée" (b7 à 1 si changée)
C027	POSNMX	position du nom de fichier dans le secteur de catalogue
C04E	VSALO1	code pour SAve/LOad (b6 = 1 si ",A" et b7 = 1 si ",J")
C04C	DEFAFF	code ASCII à mettre devant les nombres décimaux
C074		flag "listing" (b7 à 1 si OFF)
C052/C053	DESALO	adresse de début du fichier (adresse de chargement)
C100/C1FF	BUF1	
C300/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 "Langage Machine"

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)

C68Cc	4C DD E0	JMP E0DD	"FILE_NOT_FOUND_ERROR"
C68Fc	4C E0 E0	JMP E0E0	"FILE_TYPE_MISMATCH_ERROR"

Fini

C692c	68	PLA	élimine 2 octets de la pile et retourne
C693c	68	PLA	
C694c	60	RTS	

Analyse de la syntaxe et saisie des paramètres

C695c	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
C698c	08	PHP	sauvegarde les indicateurs 6502 dont Z
C699c	48	PHA	sauve A qui contient le caractère suivant
C69Ac	2C 16 C0	BIT C016	teste si b7 du flag "BANQUE changée" est à 0
C69Dc	10 0A	BPL C6A9	si oui (BANQUE pas changée), continue en C6A9
C69Fc	A2 03	LDX #03	indexe le message "LOAD"
C6A1c	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C6A4c	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)

C6A7c	B0 E9	BCS C692	si "ESC", dépile 2 octets et termine en C692
C6A9c	68	PLA	récupère A
C6AAc	28	PLP	récupère les indicateurs 6502 dont Z
C6ABc	18	CLC	pour flag C074 "listing" ON par défaut
C6ACc	F0 09	BEQ C6B7	continue en C6B7 s'il n'y a plus de paramètres
C6AEc	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C6B1c	A9 4C	LDA #4C	caractère "L" (pour inhiber le listing)
C6B3c	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay demande "L" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C6B6c	38	SEC	pour flag C074 "listing" OFF (présence de ",L")
C6B7c	6E 74 C0	ROR C074	force le b7 de C074 selon C flag "listing"

Cherche FICHCOMPL dans le catalogue

C6BAc	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
C6BDc	F0 CD	BEQ C68C	continue en C68C si pas trouvé (FILE NOT FOUND)

Charge dand BUF1 le descripteur principal de FICHCOMPL

C6BFc	BD 0C C3	LDA C30C,X	lit dans BUF3 (secteur de catalogue) les
C6C2c	BC 0D C3	LDY C30D,X	coordonnées AY du premier descripteur du fichier
C6C5c	20 5D DA	JSR DA5D	XPBUF1 Charge dans BUF1 le secteur Y de la piste A
C6C8c	2C 03 C1	BIT C103	teste si le le b7 du quatrième octet de BUF1 est nul (flag "type de fichier" indique qu'il s'agit d'un fichier non BASIC)
C6CBc	10 C2	BPL C68F	si oui, continue en C68F (“FILE_TYPE_MISMATCH_ERROR”)

Vérifie qu'il y a assez de place pour MERGEr

C6CDc	AD 06 C1	LDA C106	si fichier BASIC, calcule:
C6D0c	ED 04 C1	SBC C104	AY = adresse fin - adresse début = longueur de FICHCOMPL
C6D3c	48	PHA	(A reste sur la pile)
C6D4c	AD 07 C1	LDA C107	
C6D7c	ED 05 C1	SBC C105	
C6DAc	A8	TAY	
C6DBc	18	CLC	pour addition
C6DCc	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_ERROR" 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	XCUROFF cache le curseur (= 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 "LFCRMerging_line:"
C72Ac	20 64 D3	JSR D364	XAFSC affiche (X+1) ème 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
C759c  A2 05      LDX #05       pour se replacer au début des 5 digits
C75Bc  20 69 EE   JSR EE69      XAFXGAU affiche X fois "flèche gauche"
```

Calcule la longueur de la ligne à MERGEr

```
C75Ec  A0 03      LDY #03
C760c  C8         INY           recherche le début de ligne suivante du programme haut
C761c  B1 F8      LDA (F8),Y    (calcule le nombre d'octets qu'il faudra copier)
C763c  D0 FB      BNE C760
C765c  84 F6      STY F6        sauve Y dans F6 (nombre d'octets instructions proprement dites)
C767c  A0 01      LDY #01       pour indexer le HH du lien
C769c  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 (premier octet de lien)
C76Cc  90 04      BCC C772      si pas trouvé, saute les 2 instructions suivantes
C76Ec  66 F5      ROR F5        le b7 de F5 est mis à 1 (flag "trouvé")
C770c  30 39      BMI C7AB      suite forcée C7AB (ajuste PTRHAUT p sauter ligne)
```

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

```
C772c  A5 A0      LDA A0        ligne non trouvée, CE/CF contient adresse de la
C774c  A4 A1      LDY A1        ligne suivante, c'est à dire l'adresse du premier octet
C776c  85 C9      STA C9        du bas du bloc qu'il faut remonter pour faire place
C778c  84 CA      STY CA        à la ligne qui sera insérée
C77Ac  38         SEC           C9/CA = AY = A0/A1 (fin des tableaux)
C77Bc  65 F6      ADC F6        adresse du dernier octet du haut du bloc
C77Dc  85 C7      STA C7        qu'il faut remonter
```

```
C77Fc  48        PHA       C7/C8 = AY = AY + F6 + #01
C780c  98        TYA       F6 = nombre d'octets à copier = longueur de ligne
C781c  69 00     ADC #00   C7/C8 = adresse cible vers le haut
C783c  85 C8     STA C8
C785c  A8        TAY
C786c  68        PLA
C787c  20 5C D1  JSR D15C  C3F4/ROM décale un bloc mémoire vers le haut. En CE/CF adresse du
                           premier 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)

```
C78Ac  A2 02     LDX #02
C78Cc  38        SEC
C78Dc  B5 9C     LDA 9C,X  l'adresse présente en 9E/9F (début des tableaux)
C78Fc  65 F6     ADC F6    est incrémentée du contenu de F6 + 1
C791c  95 9C     STA 9C,X
C793c  90 02     BCC C797
C795c  F6 9D     INC 9D,X
C797c  CA        DEX       l'adresse présente en 9C/9D (début des variables)
C798c  CA        DEX       est incrémentée du contenu de F6 + 1
C799c  10 F1     BPL C78C  (reboucle une fois en C78C)
C79Bc  A4 F6     LDY F6    nombre d'octets à copier
C79Dc  B1 F8     LDA (F8),Y lit octet à PTRHAUT
C79Fc  91 CE     STA (CE),Y écrit octet à PTRBAS
C7A1c  88        DEY       octet précédent (opère par la fin)
C7A2c  10 F9     BPL C79D  reboucle tant qu'il en reste
C7A4c  A5 CE     LDA CE
C7A6c  A4 CF     LDY CF    AY = adresse du nouveau bloc
C7A8c  20 8C D1  JSR D18C  C563/ROM restaure les liens à partir de l'adresse AY
```

Mise à jour de PTRHAUT

```
C7ABc  38        SEC
C7ACc  A5 F6     LDA F6
C7AEc  65 F8     ADC F8    F8/F9 = F8/F9 + F6 + #01
C7B0c  85 F8     STA F8
C7B2c  90 02     BCC C7B6
C7B4c  E6 F9     INC F9
C7B6c  4C 2D C7  JMP C72D  reprise forcée en C72D
```

Messages externes de la BANQUE n°3 (C7B9 à C7DC)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

```
C7B9c  20 46 6F 75 6E 64 73 0A 8D
01     _FoundsLF CR
```

(NB: Avec une belle faute car le participe passé ne prend jamais la forme plurielle en anglais!)

C7C2c 0A 0D 4D 65 72 67 69 6E 67 20 6C 69 6E 65 **BA**
02 LFCRMerging_line**:**

C7D1c 0A 0D 4C 49 4E 45 20 **BA**
03 LFCRLINE **_:**

C7D9c 4C 4F 41 **C4**
04 LOA**D**

Rappel: **_** = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

Messages d'erreur externes de la BANQUE n°3 (C7DD à C7FE)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

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: COPY

Cette BANQUE se trouve à partir du #51 (quatre vingt unième) secteur de la disquette MASTER.

C400d	00 00	EXTER	adresse des messages d'erreur externes (néant)
C402d	8F C7	EXTMS	adresse des messages externes

EXÉCUTION DE LA COMMANDE SEDORIC COPY

Rappel de la syntaxe:

a) **COPY (nom_de_fichier_ambigu_source)(TOnom_de_fichier_ambigu_cible)(,C)(,N)** il doit y avoir correspondance entre les jokers du nom_de_fichier_ambigu_source et ceux du nom_de_fichier_ambigu_cible ou alors le nom_de_fichier_ambigu_cible doit être *.* ou omis (ce qui revient au même).

a) **COPYO (nom_de_fichier_ambigu_source)(TOnom_de_fichier_ambigu_cible)(,C)(,N)** il doit y avoir correspondance entre les jokers du nom_de_fichier_ambigu_source et ceux du nom_de_fichier_ambigu_cible ou alors le nom_de_fichier_ambigu_cible doit être *.* ou omis (ce qui revient au même).

b) **COPYM (nom_de_fichier_ambigu_source) TOnom_de_fichier_non_ambigu_cible (,C)(,N)** les jokers ne sont pas autorisés, dans le fichier cible.

Dans tous les cas, l'option ",C" permet que soit demandée confirmation pour chacun des fichiers correspondant à un nom_de_fichier_ambigu_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

00/01	n° de PISTE et n° de SECTEUR actifs en lecture
02/03	n° de PISTE et n° de SECTEUR actifs en écriture
04	initialisé à #00 (b7 à zéro si première passe de lecture)
05	n° du message d'erreur
06	initialisé à #80
07	b6 à 1 si option ",C" (confirmation demandée avant copie)
	b7 à 1 si option ",N" (inhibition de demande changement de disquette)
0A/0B	nombre de secteurs restant à sauver
16	b6 à 1 si COPYM (et à 0 si COPY ou COPYO)
	b7 à 1 si COPY (et à 0 si COPYM ou COPYO)
F4	b7 à 1 s'il y a au moins un "?" dans le nom de fichier source
F5/F6	RWBUF
F7/F8	nombre de secteurs à charger

F9	POSNMX puis flag "lecture/écriture" (b7 à 1 si écriture)
C025	POSNMP
C026	POSNMS
C027	POSNMX
C08C	position dans la liste des coordonnées des secteurs à charger
C08D/C08E	nombre de secteurs restant à charger
C090/C09C	nom_de_fichier_ambigu source
C09D/C09A	nom_de_fichier_ambigu 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.

Le nom_de_fichier_ambigu_source peut être omis (dans ce cas, tous les fichiers du drive courant seront copiés). Le "TO nom_de_fichier_cible" peut aussi être omis, sauf avec COPYM. Finalement, le nom_de_fichier_ambigu_cible peut 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” marchent parfaitement, à condition de ne pas inhiber l’affichage de changement de disquette avec l’option ,N. Par contre, COPYMTO”TOTO”,N copiera sans problème tous les fichiers dans TOTO.COM sans rien demander et ceci sur la même disquette. Toutefois, TOTO.COM contiendra tous les fichiers d’origine, plus TOTO.COM lui-même, c’est à dire tous les fichiers d’origine une deuxième fois! Et ça ne plante pas!

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 "Langage Machine"

Si la BANQUE n°4 est déjà en place, il suffira de basculer sur la RAM overlay, 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 nom_de_fichier_ambigu_source, zone C09D/C09E avec nom_de_fichier_ambigu_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

C404d	C9 4D	CMP #4D	est-ce un "M" ? (Merge)
C406d	F0 07	BEQ C40F	si oui, continue en C40F
C408d	C9 4F	CMP #4F	est-ce un "O" ? (Over)
C40Ad	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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
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 que l'on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"
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 (nom_de_fichier_ambigu_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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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 puis D3A1/RAM overlay 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 (nom_de_fichier 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_ERROR"
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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C475d	D0 E2	BNE C459	reprend en C459 s'il y a encore un paramètre

Initialise drive source = drive actif et BUFNOM avec nom_de_fichier_ambigu_source

C477d	20 58 FE	JSR FE58	copie nom_de_fichier_ambigu_source dans BUFNOM et vérifie les jockers (revient avec b7 de F4 à 1 s'il y a au moins un "?" dans le nom_de_fichier_ambigu_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 AND PRESS 'RETURN' si b7 de 07 à 1 (multidrive ou monodrive avec demande inhibée)
ou	“LOAD_SOURCE_DISC_IN_DRIVE_X AND_PRESS_'RETURN' si b7 de 07 à zéro (uniquement monodrive avec demande non inhibée)

C48Fd	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C492d	24 07	BIT 07	teste si le b7 de 07 est à 1
C494d	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 inhibée), demande d'abord la disquette source.
C496d	A2 00	LDX #00	indexe le message "LOAD_SOURCE"
C498d	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C49Bd	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)
C49Ed	90 2B	BCC C4CB	si "RETURN", continue en C4CB
C4A0d	58	CLI	si "ESC", autorise les interruptions
C4A1d	60	RTS	et retourne

Demande initiale uniquement

C4A2d	A2 03	LDX #03	index le message "LOAD_DISCS_FOR_COPY_FROM_"
C4A4d	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C4A7d	AD 90 C0	LDA C090	n° du lecteur source
C4AAd	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
C4ADd	A2 04	LDX #04	indexe le message "_TO_"
C4AFd	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C4B2d	AD 9D C0	LDA C09D	n° du lecteur cible
C4B5d	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
C4B8d	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C4BBd	A2 0D	LDX #0D	indexe "AND_PRESS_'RETURN'"
C4BDd	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
C4C0d	20 69 D6	JSR D669	demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
C4C3d	B0 DB	BCS C4A0	si "ESC", continue en C4A0 (simple CLI et RTS)
C4C5d	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C4C8d	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Recherche le fichier source

C4CBd	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
C4CEd	D0 03	BNE C4D3	si trouvé, saute l'instruction suivante
C4D0d	4C DD E0	JMP E0DD	si pas trouvé, "FILE_NOT_FOUND_ERROR"
C4D3d	86 F9	STX F9	sauve POSNMX dans F9
C4D5d	24 07	BIT 07	teste si le b6 de 07 est à zéro
C4D7d	50 2B	BVC C504	si oui (sans confirmation), continue en C504

Demande s'il faut copier le fichier source

C4D9d	20 B4 DA	JSR DAB4	affiche nom de fichier présent à POSNMX dans BUF3
C4DCd	A2 0A	LDX #0A	indexe " (Y)es_or_(N)o:CR**LF**"
C4DEd	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
C4E1d	58	CLI	autorise les interruptions
C4E2d	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII

```
                                    correspondant, sinon N = 0
C4E5d  20 A1 D3   JSR D3A1        XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
C4E8d  C9 1B      CMP #1B         est-ce un "ESC"?
C4EAd  F0 B4      BEQ C4A0        si oui, CLI et RTS en C4A0
C4ECd  C9 4E      CMP #4E         est-ce un "N"?
C4EEd  D0 03      BNE C4F3        sinon, saute l'instruction suivante
C4F0d  4C 77 C5   JMP C577        si oui (on ne copie pas), continue en C577

C4F3d  C9 59      CMP #59         est-ce un "Y"?
C4F5d  D0 EB      BNE C4E2        sinon, reprend en C4E2 (nouvelle saisie de touche)
C4F7d  20 2A D6   JSR D62A        XAFCAR affiche le caractère ASCII contenu dans A
C4FAd  20 06 D2   JSR D206        CBF0/ROM va à la ligne
C4FDd  24 07      BIT 07          teste si le b7 de 07 est à 1
C4FFd  30 03      BMI C504        si oui (inhibe demande disc cible) saute l'instruction suivante
C501d  20 06 D2   JSR D206        CBF0/ROM va à la ligne

       Copie le fichier source

C504d  AD 25 C0   LDA C025        empile POSNMP
C507d  48         PHA
C508d  AD 26 C0   LDA C026        empile POSNMS
C50Bd  48         PHA
C50Cd  AD 27 C0   LDA C027        empile POSNMX
C50Fd  48         PHA             (coordonnées pour fichier source)
C510d  20 8D C5   JSR C58D        transfère les secteurs d'un fichier
C513d  68         PLA
C514d  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+1) ème message externe terminé par "caractère + 128"
C555d	30 03	BMI C55A	et continue en C55A
C557d	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
C55Ad	20 58 FE	JSR FE58	copie nom_de_fichier_ambigu_source dans BUFNOM et vérifie les jockers (revient avec b7 de F4 à 1 s'il y a au moins un "?" dans le nom_de_fichier_ambigu_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+1) ème 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_completeLF**CR**"
C58Ad	4C 64 D3	JMP D364	XAFSC affiche (X+1) ème 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 première passe lecture,
C597d	85 04	STA 04	et b6 à 0 si première 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 (première passe de lecture)
C5A9d	10 13	BPL C5BE	si oui, continue en C5BE, sinon passe ultérieure...

Demande éventuellement la disquette source

C5ABd	24 07	BIT 07	teste si le b7 de 07 est à 1
C5ADd	30 1A	BMI C5C9	si oui (inhibe demande disquette), continue en C5C9
C5AFd	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C5B2d	A2 00	LDX #00	indexe le message "LOAD_SOURCE"
C5B4d	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C5B7d	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)
C5BAd	90 0D	BCC C5C9	si "RETURN", continue en C5C9
C5BCd	58	CLI	si "ESC", autorise les interruptions
C5BDd	60	RTS	et retourne

Première passe en lecture: prend les coordonnées du premier descripteur

C5BEd	AE 27 C0	LDX C027	X = POSNMX
C5C1d	BD 0C C3	LDA C30C,X	lit PISTE dans BUF3 à la position POSNMX + #0C
C5C4d	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)
C5C7d	D0 04	BNE C5CD	secteur jamais nul = suite forcée en C5CD

Passe ultérieure en lecture: reprend les coordonnées du descripteur en cours

C5C9d	A5 00	LDA 00	récupère le n° de PISTE active en lecture
C5CBd	A4 01	LDY 01	récupère le n° de SECTEUR actif en lecture
C5CDd	20 5D DA	JSR DA5D	XPBUF1 Charge dans BUF1 le secteur Y de la piste A
C5D0d	A0 B4	LDY #B4	(c'est à dire le descripteur du fichier source)
C5D2d	8C 04 C0	STY C004	HH de RWBUF = #B4 pour former l'adresse B400
C5D5d	84 F6	STY F6	force le b7 de F6 à 1 pour indiquer que l'on va charger le premier descripteur
C5D7d	AE 04 C0	LDX C004	X = HH du RWBUF courant
C5DAd	E0 05	CPX #05	teste si la limite inférieure (#05) est atteinte
C5DCd	F0 1D	BEQ C5FB	si oui, continue en C5FB, sinon...
C5DEd	20 73 DA	JSR DA73	XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C5E1d	06 06	ASL 06	récupère le b7 de 06 dans C qui sera donc toujours nul, sauf lors du premier tour, pour indiquer qu'on vient de charger en RAM le premier descripteur du fichier
C5E3d	20 0C C7	JSR C70C	lecture des secteurs du fichier dans la limite de la place disponible en RAM
C5E6d	AC 04 C0	LDY C004	HH de RWBUF
C5E9d	8C 8F C0	STY C08F	position actuelle du pointeur sauvegardée en C08F
C5ECd	B0 0D	BCS C5FB	continue en C5FB (il reste des secteurs à charger)
C5EEd	A8	TAY	Y = pointeur dans la liste des decripteurs
C5EFd	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

```
C5F2d  78        SEI          interdit les interruptions
C5F3d  B0 05     BCS C5FA     continue en C5FA si C = 1 (il n'y a plus de descripteur)
C5F5d  38        SEC
C5F6d  66 06     ROR 06       force à 1 le b7 de 06
C5F8d  30 DD     BMI C5D7     reprise forcée en C5D7 si b7 = 1
```

Il n'y a plus de descripteur

```
C5FAd  18        CLC          force C = zéro
```

Initialise le drive cible

```
C5FBd  66 04     ROR 04       C -> b7 (0 il n'y a plus de descripteur, 1 il en reste)
C5FDd  AD 9D C0  LDA C09D     n° de drive cible
C600d  8D 00 C0  STA C000     devient DRIVE actif
C603d  24 07     BIT 07       teste si le b7 de 07 est à 1
C605d  30 0A     BMI C611     si oui (inhibe demande disquette), continue en C611
C607d  A2 01     LDX #01      sinon, indexe le message "LOAD_TARGET"
C609d  20 64 D3  JSR D364     XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
       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)
C60Fd  B0 AB     BCS C5BC     si "ESC", continue en C5BC (simple CLI et RTS)
C611d  38        SEC
C612d  66 F9     ROR F9       force à 1 le b7 de F9 (flag "écriture")
C614d  24 04     BIT 04       teste si le b6 de 04 à 0 (première passe en écriture)
C616d  50 0A     BVC C622     si oui, continue en C622, sinon, reprend les
C618d  A5 02     LDA 02       coordonnées du descripteur en cours:
C61Ad  A4 03     LDY 03       A = piste selon 02 et Y = secteur selon 03
C61Cd  20 5D DA  JSR DA5D     XPBUF1 Charge dans BUF1 le secteur Y de la piste A
C61Fd  38        SEC          force C = 1
C620d  F0 4B     BEQ C66D     suite forcée en C66D
C622d  AE 27 C0  LDX C027     X = POSNMX
C625d  A0 00     LDY #00      index d'exploration
C627d  B9 9E C0  LDA C09E,Y   lit un octet du nom de fichier cible
C62Ad  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 SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
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_EXISTSLF**CR**"
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 la 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 premier secteur descripteur dans PSDESC (C05C/C05D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place
C6AEd	68	PLA	récupère LL de NSRSAV
C6AFd	18	CLC	prépare une addition pour calculer dans 0A/0B le nombre total de secteurs
C6B0d	65 0A	ADC 0A	
C6B2d	85 0A	STA 0A	0A = 0A + LL de NSRSAV
C6B4d	48	PHA	
C6B5d	A5 0B	LDA 0B	
C6B7d	6D 5B C0	ADC C05B	
C6BAd	85 0B	STA 0B	0B = 0B + HH de NSRSAV
C6BCd	68	PLA	
C6BDd	6D 5E C0	ADC C05E	
C6C0d	85 0A	STA 0A	
C6C2d	90 02	BCC C6C6	
C6C4d	E6 0B	INC 0B	
C6C6d	28	PLP	récupère les indicateurs 6502
C6C7d	B0 0A	BCS C6D3	continue en si C = 1
C6C9d	AD 5C C0	LDA C05C	
C6CCd	AC 5D C0	LDY C05D	
C6CFd	85 08	STA 08	08/09 = C05C/C05D (PSDESC)
C6D1d	84 09	STY 09	
C6D3d	68	PLA	
C6D4d	8D 04 C0	STA C004	récupère HH de RWBUF
C6D7d	06 06	ASL 06	
C6D9d	20 0C C7	JSR C70C	
C6DCd	AC 04 C0	LDY C004	
C6DFd	84 F6	STY F6	F6 = HH de RWBUF
C6E1d	88	DEY	
C6E2d	CC 8F C0	CPY C08F	teste si Y >= C08F
C6E5d	B0 97	BCS C67E	si oui, reprend en C67E
C6E7d	24 04	BIT 04	sinon, teste si le b7 de 04 est à zéro
C6E9d	10 03	BPL C6EE	si oui, saute l'instruction suivante
C6EBd	4C 9F C5	JMP C59F	sinon, reprend en C59F

```
C6EEd A5 08      LDA 08
C6F0d A4 09      LDY 09      C05C/C05D (PSDESC) = 08/09
C6F2d 8D 5C C0   STA C05C
C6F5d 8C 5D C0   STY C05D
C6F8d A5 0A      LDA 0A
C6FAd A4 0B      LDY 0B
C6FCd A2 00      LDX #00
C6FEd 8E 5E C0   STX C05E    force C05E à zéro
C701d 8D 5A C0   STA C05A
C704d 8C 5B C0   STY C05B    C05A/C05B = 0A/0B
C707d 20 81 DF   JSR DF81    mise à jour du nombre de secteurs totaux du fichier
C70Ad 18         CLC
C70Bd 60         RTS
```

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

```
C70Cd 90 0A      BCC C718    continue en C718 si C = zéro
C70Ed A9 0A      LDA #0A     sinon (C = 1) on est au début de la première passe...
C710d AE 0A C1   LDX C10A    A = 10 position initiale dans liste des coordonnées
C713d AC 0B C1   LDY C10B    XY = nombre de secteurs à charger
C716d B0 09      BCS C721    suite forcée en C721
C718d AE 8D C0   LDX C08D    on est au début d'une passe ultérieure...
C71Bd AC 8E C0   LDY C08E    XY = nombre de secteurs à charger
C71Ed AD 8C C0   LDA C08C    A = position dans la liste des coordonnées
C721d E8         INX
C722d D0 01      BNE C725    incrémente le nombre de secteurs à transférer
C724d C8         INY         pour avoir valeur correcte en début de boucle C72D
C725d 86 F7      STX F7      et sauve le résultat en F7/F8
C727d 84 F8      STY F8
C729d A8         TAY         Y = position dans la liste des coordonnées
C72Ad 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")

```
C72Dd A5 F7      LDA F7
C72Fd D0 02      BNE C733    décrémente F7/F8, le nombre de secteurs à transférer
C731d C6 F8      DEC F8
C733d 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)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

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**

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

C7F8d	C6 0C		le reste semble n'avoir aucun sens...
C7FAd	D0 F5	BNE C7F1	ce n'est pas une adresse de branchement
C7FCd	F0 E5	BEQ C7E3	idem
C7FEd	60	RTS	
C7FFd	00	BRK	

BANQUE n°5: SYS DNAME DTRACK TRACK INIST DNUM DSYS DKEY VUSER

Cette BANQUE se trouve à partir du #56 (quatre vingt sixième) secteur de la disquette MASTER.

C400e	00 00	EXTER	adresse des messages d'erreur externes (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

EXÉCUTION DE LA 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 sous-programme 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 "Langage Machine"

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 RAM overlay) 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	XAFXGAU 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

EXÉCUTION DE LA 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

F2	index pour écrire dans BUF1 (selon le n° du drive)	
F7	b7 à 0 pour ";S" et à 1 pour ";D"	
F9	LL totalisateur pour calcul nombre de secteurs/face	
C016	flag "BANQUE changée"	
C072	flag TRACK/DTRACK (b7 à 0 si TRACK à 1 si DTRACK)	
C100/C1FF	BUF1	pour charger le secteur système
C200/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 "Langage Machine" 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 était plus que farfelue et a été corrigée.

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. Je pense 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 "Langage Machine"

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"

C43Ee	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C441e	20 DB C6	JSR C6DB	demande la disquette cible

```
C444e  38      SEC      C = 1 = flag DTRACK
C445e  24 18   BIT 18   et continue en C447
```

EXÉCUTION DE LA 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

```
F2              index pour écrire dans BUF1 (selon le n° du drive)
F7              b7 à 0 pour ";S" et à 1 pour ";D"
F9              LL totalisateur pour calcul nombre de secteurs/face
C039/C03C       TABDRV      table de configuration des lecteurs
C072            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 "Langage Machine" 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 était plus que farfelue et a été corrigée.

Voir les "Informations non documentées" de la commande DTRACK concernant l'utilité des commandes TRACK et DTRACK.

Utilisation en "Langage Machine"

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 premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").
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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C46Ae	A9 44	LDA #44	caractère "D"
C46Ce	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay 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	STA C100,Y	écrit le nombre de pistes/face avec l'indice du nombre de faces (b7) dans BUF1 à la position Y
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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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

Ce sous-programme était largement périmé, puisque la double bitmap de Ray autorise au moins 3838 secteurs, chiffre qui peut tout juste être atteint dans les limites maximales actuelles (101 pistes de 19 secteurs, double face pour l'extension "**BIGDISK**" à utiliser seulement avec EUPHORIC).

Je l'ai donc simplifié, ce qui dégage de la place pour d'éventuelles fonctions supplémentaires. Au total les 52 octets d'origine sont donc différents.

C4A1e	**A8**	TAY	teste si A est nul ("unconnected")
C4A2e	**F0 08**	BEQ C4AC	si oui, retourne
C4A4e	**C9 15**	CMP #15	teste si A < 21 pistes
C4A6e	**90 2D**	BCC C4D5	si oui, “ILLEGAL_QUANTITY_ERROR”
C4A8e	**C9 66**	CMP #66	teste si A >= 102 pistes (soit un nouveau maximum de 101 pistes)
C4AAe	**B0 29**	BCS C4D5	si oui, “ILLEGAL_QUANTITY_ERROR”
C4ACe	**60**	RTS	

De C4AD à C4D4, nettoyage du code inutile, remplacé par **40 NOPs**.

C4D5e 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

Rétablissement du JMPDE20 (bogue des versions 2.x), comme dans la version 1.006

EXÉCUTION DE LA 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 "Langage Machine" 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 "Langage Machine"

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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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

EXÉCUTION DE LA 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 sous-programme 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 SEDORIC, 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 "Langage Machine"

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 RAM overlay) suffit. Il est possible de mettre la BANQUE n°5 en place par un JSR F139 sous RAM overlay. 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

C509e	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)
C50Ce	A2 3C	LDX #3C	X = 60
C50Ee	20 69 EE	JSR EE69	XAFXGAU affiche X fois "flèche gauche" (retourne au début)
C511e	A9 3C	LDA #3C	A = 60 (nombre de caractères à saisir)
C513e	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
C515e	A2 1E	LDX #1E	X = 30 (position de INIST dans BUF1)
C517e	20 D6 C5	JSR C5D6	saisit une chaîne et la copie dans BUF1
C51Ae	A4 F4	LDY F4	teste si le mode de sortie est 1
C51Ce	88	DEY	c'est à dire sortie par ESC
C51De	F0 **8D**	BEQ C4AC	branche vers un RTS

Ici j'ai adapté le BEQ C4D4 d'origine en BEQ C4AC. Cette adaptation est nécessitée par les modifications intervenues dans la zone C4A1 à C4AC

C51Fe	4C A4 DA	JMP DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

EXÉCUTION DE LA 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

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/F4	adresse de lecture dans BUF1 (secteur système)
F4	mode de sortie de XLINPU
F7	n° du lecteur
F8	on se demande bien à quoi il sert (utilisé par sous-programme C68E)
F8/F9	adresse de la table des lecteurs TABDRV
F9	position de la chaîne dans BUF1
C000/C004	DRIVE, PISTE, SECTEUR, RWBUF
C016	flag "BANQUE changée"
C04C	DEFAFF, code ASCII devant les nombres décimaux
C100/C1FF	BUF1 pour charger le secteur système

Informations non documentées

Néant

Utilisation en "Langage Machine"

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 RAM overlay) suffit. Il est possible de mettre la BANQUE n°5 en place par un JSR F139 sous RAM overlay. 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

C522e	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)
C525e	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Affichage des instructions de démarage INIST

C528e	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+1) ème message externe terminé par "caractère + 128"
C53Ae	A2 0C	LDX #0C	indexe le message: "ACCENT_"
C53Ce	20 64 D3	JSR D364	XAFSC affiche (X+1) ème 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+1) ème 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+1) ème message externe terminé par "caractère + 128"
C555e	A2 11	LDX #11	indexe le message: "ERTYLF**CR**"
C557e	4C 64 D3	JMP D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"

EXÉCUTION DE LA COMMANDE SEDORIC SYS

Rappel de la syntaxe

SYS tout court

Affiche une partie de la configuration de SEDORIC présent en RAM overlay: 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 "Langage Machine"

Simple: on passe sous RAM overlay et on fait un JSR F15A (voir ANNEXE).

Affiche la configuration courante des lecteurs

C55Ae	A9 39	LDA #39	F8/F9 = C039 (TABDRV en RAM overlay 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+1) ème 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+1) ème 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)
C58Be	29 7F	AND #7F	masque 0111 1111, force b7 à zéro
C58De	20 4E D7	JSR D74E	affichage en décimal sur 2 digits d'un nombre A
C590e	A2 07	LDX #07	indexe le message "_tracks_"
C592e	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C595e	68	PLA	récupère A
C596e	30 03	BMI C59B	si b7=1 (Double face), continue en C59B
C598e	A2 08	LDX #08	indexe le message "single_"
C59Ae	2C A2 09	BIT 09A2	continue en C59D
C59Be	A2 09	LDX #09	indexe le message "double_"
C59De	20 64 D3	JSR D364	XAFSC affiche (X+1) message externe terminé par "caractère + 128"
C5A0e	A2 0A	LDX #0A	indexe le message "sided_"
C5A2e	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C5A5e	A4 F7	LDY F7	numéro de lecteur
C5A7e	C8	INY	lecteur suivant (numéro 3 = maximum)
C5A8e	C0 04	CPY #04	teste si numéro de lecteur < 4
C5AAe	90 BD	BCC C569	si oui, reboucle en C569

Affiche les valeurs courantes de DEFNUM et DEFPAS

C5ACe	A9 20	LDA #20	sinon, A = espace
C5AEe	8D 4C C0	STA C04C	DEFAFF, code ASCII devant les nombres décimaux
C5B1e	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C5B4e	A2 03	LDX #03	indexe le message "LFCRNum_origin:"
C5B6e	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C5B9e	A0 05	LDY #05	index de lecture
C5BBe	20 CB C5	JSR C5CB	lit et affiche sur 5 digits la valeur de DEFNUM
C5BEe	A2 02	LDX #02	indexe le message "LFCRNum_step__:"
C5C0e	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C5C3e	A0 07	LDY #07	index de lecture
C5C5e	20 CB C5	JSR C5CB	lit et affiche sur 5 digits la valeur de DEFPAS
C5C8e	4C 06 D2	JMP D206	CBF0/ROM va à la ligne

Lit et affiche en décimal sur 5 digits

C5CBe	B1 F8	LDA (F8),Y	lit octet à l'adresse indiquée en F8/F9 + Y
C5CDe	48	PHA	et l'empile
C5CEe	C8	INY	octet suivant
C5CFe	B1 F8	LDA (F8),Y	lit octet à l'adresse indiquée en F8/F9 + Y
C5D1e	A8	TAY	et le passe dans Y
C5D2e	68	PLA	récupère A
C5D3e	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 X ème position dans BUF1

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
F3	options E, S, C, J, K pour XLINPU
F4	mode de sortie de XLINPU
F9	position de la chaîne dans BUF1
C075	caractère à utiliser pour matérialiser la fenêtre

Sous-programme commun à plusieurs commandes

C5D6e	85 F2	STA F2	longueur de la chaîne à saisir
C5D8e	86 F9	STX F9	position de la chaîne dans BUF1
C5DAe	84 F3	STY F3	options E, S, C, J, K pour LINPUT
C5DCe	A9 20	LDA #20	sauvegarde du caractère "espace" pour XLINPU
C5DEe	8D 75 C0	STA C075	caractère à utiliser pour matérialiser la fenêtre
C5E1e	20 36 ED	JSR ED36	XLINPU routine principale de LINPUT (routine de saisie de chaîne), au retour F4 contient le mode de sortie et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM.
C5E4e	20 3E D7	JSR D73E	XCURON rend le curseur visible (= vidéo inverse)
C5E7e	A6 F4	LDX F4	mode de sortie de XLINPU
C5E9e	CA	DEX	teste si mode de sortie est différent de 1 ("ESC")
C5EAe	D0 01	BNE C5ED	si oui, saute l'instruction suivante
C5ECe	60	RTS	simple RTS si "ESC"
C5EDe	A0 00	LDY #00	Y = index de lecture dans la chaîne saisie
C5EFe	A6 F9	LDX F9	X = index d'écriture dans BUF1
C5F1e	B1 D1	LDA (D1),Y	lecture caractère dans la chaîne saisie
C5F3e	9D 00 C1	STA C100,X	écriture dans BUF1
C5F6e	E8	INX	suivant en écriture
C5F7e	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	

EXÉCUTION DE LA 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 "Langage Machine"

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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
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 puis D3A1/RAM overlay 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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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,

EXÉCUTION DE LA COMMANDE SEDORIC VUSER

Rappel de la syntaxe

VUSER tout court

Affiche les chaînes de définitions des 16 commandes re-definissables utilisateurs stockées en RAM overlay 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 re-definissables
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 re-definissables

Informations non documentées

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

Utilisation en "Langage Machine"

Bon, ce coup-là, c'est vraiment simple: on bascule sur la RAM overlay et on fait un JSR F121.

Entrée de la commande

C62Ae	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C62De	A2 04	LDX #04	indexe le message "LFCRUser_fonctions:LF**CR**"
C62Fe	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C632e	A9 30	LDA #30	A = "0"
C634e	8D 4C C0	STA C04C	DEFAFF, code ASCII devant les nombres décimaux
C637e	A2 00	LDX #00	F9=0 compteur du nombre de commandes affichées
C639e	86 F9	STX F9	c'est à dire le n° de code de fonction de #00 à #0F
C63Be	86 F8	STX F8	F8=0 index lecture table commandes re-definissables
C63De	A9 80	LDA #80	au début de chaque ligne de commnde,
C63Fe	85 F7	STA F7	force à 1 le b7 de F7, (reste à 1 tant qu'un caractère significatif n'a pas été rencontré)
C641e	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C644e	A5 F9	LDA F9	A = n° de code de fonction de #00 à #0F
C646e	20 4E D7	JSR D74E	affichage en décimal sur 2 digits d'un nombre A
C649e	A9 3A	LDA #3A	A = ":"

C64Be	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C64Ee	20 28 D6	JSR D628	affiche un espace pour séparer l'en-tête de la ligne de commande
C651e	A6 F8	LDX F8	index lecture table commandes re-definissables
C653e	BD 80 C8	LDA C880,X	lit octet dans table commandes re-definissables
C656e	08	PHP	sauvegarde les indicateurs 6502 dont N (à1 si dernier caractère d'une ligne)
C657e	29 7F	AND #7F	masque 0111 1111 pour forcer b7 à 0
C659e	C9 20	CMP #20	est-ce un "espace"? (positionne C à 1 si A >= #20)
C65Be	D0 04	BNE C661	sinon, continue en C661
C65De	24 F7	BIT F7	si oui, teste si b7 de F7 est à 1 (mis à 1 au début de chaque ligne de commande)
C65Fe	30 1B	BMI C67C	si oui, continue en C67C (saute l'affichage des premiers espaces)
C661e	85 F7	STA F7	F7 reçoit le caractère dont le b7 à été mis à 0
C663e	B0 14	BCS C679	si caractère affichable, continue en C679
C665e	20 28 D6	JSR D628	si code de CTRL, affiche un espace (sera devant le code de CRTL)
C668e	A5 F7	LDA F7	récupère le caractère
C66Ae	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 £)
C66Ce	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C66Fe	AC 69 02	LDY 0269	n° de colonne TEXT = abscisse X du curseur
C672e	88	DEY	case précédente
C673e	09 80	ORA #80	masque 1000 0000 pour forcer b7 à 1 (vidéo inverse)
C675e	91 12	STA (12),Y	affiche caractère en vidéo inverse
C677e	D0 03	BNE C67C	saut forcé de l'instruction suivante
C679e	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C67Ce	28	PLP	récupère les indicateurs 6502 dont N
C67De	30 03	BMI C682	continue en C682 si b7=1 (c'était le dernier caractère de la ligne)
C67Fe	E8	INX	index de lecture table commandes re-definissables
C680e	D0 D1	BNE C653	reprise forcée en C653
C682e	E6 F9	INC F9	nombre de lignes de commande affichées
C684e	E8	INX	index de lecture table commandes re-definissables
C685e	D0 B4	BNE C63B	reboucle en C63B (commande suivante) tant que toute la table n'a pas été affichée, soit 256 octets
C687e	4C 06 D2	JMP D206	CBF0/ROM va à la ligne

Affiche INIST

Variables utilisées par le sous-programme C68A

F3/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

C68Ae	18	CLC	C = 0 flag INITIAL STATEMENTS, "instructions au
C68Be	08	PHP	démarrage" sauvegarde les indicateurs 6502 dont C
C68Ce	90 11	BCC C69F	suite forcée en C69F

Affiche DNAME ou INIST

Variables utilisées par les sous-programmes C68E et C690

F3/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

Ray a corrigé ici une bogue de la BANQUE n°5, qui affectait les commandes **DKEY**, **DNAME**, **DNUM**, **DSYS**, **DTRACK**, **INIST** & **TRACK** . La routine C6DB "Demande la disquette cible" était boguée (mauvaise gestion de "ESC") et a été remplacée (2 octets différents) par une nouvelle routine en C7A0 (voir plus loin). L'ancien JSR C6DB a été remplacé pour pouvoir utiliser la routine déboguée.

C69Ae	20 **A0 C7**	JSR C7A0	nouvelle routine "Demande la 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 premier 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 deuxième 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+1) ème 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+1) ème message externe terminé par "caractère + 128"
C6E5e  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)
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
C6ECe  68        PLA         le JMP D206 exécture la routine CBF0 "va à la ligne" en ROM
C6EDe  4C 06 D2  JMP D206    et retourne à l'avant-dernier appelant

C6F0e  60        RTS
```

Messages externes de la BANQUE n°5 (C6F1 à C792)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

C6F1e 0A 0D 44 69 73 63 20 6E 61 6D 65 **BA**
01 LFCRDisc_name**:**

C6FDe 0A 0D 49 6E 69 74 20 73 74 61 74 65 6D 65 6E 74 **BA**
02 LFCRInit_statement**:**

C70Ee 0A 0D 4E 75 6D 20 73 74 65 70 20 20 **BA**
03 LFCRNum_step__**:**

C71Be 0A 0D 4E 75 6D 20 6F 72 69 67 69 6E **BA**
04 LFCRNum_origin**:**

C728e 0A 0D 55 73 65 72 20 66 6F 6E 63 74 69 6F 6E 73 3A 0A **8D**
05 LFCRUser_fonctions:LF**CR**

C73Be 0A 0D 4B 65 79 62 6F 61 72 64 **BA**
06 LFCRKeyboard**:**

C746e 0A 0D 44 72 69 76 65 **A0**
07 LFCRDrive_

C74Ee 20 74 72 61 63 6B 73 **A0**
08 _tracks_

C756e 73 69 6E 67 6C 65 **A0**
09 single_

C75De 64 6F 75 62 6C 65 **A0**
0A double_

C764e 73 69 64 65 64 **A0**
0B sided_

C76Ae 75 6E 63 6F 6E 6E 65 63 74 65 64 **A0**
0C unconnected_

C776e 41 43 43 45 4E 54 **A0**
0D ACCENT_

C77De 53 45 54 **AC**
0E SET**,**

C781e 4F 46 46 **AC**
0F OFF**,**

C785e 41 **DA**
10 A**Z**

C787e 51 **D7**
11 Q**W**

C789e 45 52 54 59 0A **8D**
12 ERTYLF**CR**

C78Fe 4C 4F 41 **C4**
13 LOA**D**

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

Le reste (C793 à C7FF) n'avait aucun sens. Il s'agissait en fait de la fin de la BANQUE n°2 qui est absolument identique. C'était donc le résidu d'une compilation antérieure. Les #6D (109) octets correspondants etaient donc récupèrables. C'est ce qu'a fait RAY en plaçant ici sa nouvelle routine déboguée "Demande la disquette cible".

De C793e à C79Fe, nettoyage code inutile, remplacé par **13 NOPs.**

Nouvelle routine de Ray: "Demande la disquette cible" (25 octets différents)

C7A0e	**2C 16 C0**	BIT C016	test si le b7 du flag "BANQUE changée" est à zéro
C7A3e	**10 14**	BPL C7B9	si oui (BANQUE pas changée), simple RTS en C7B9
C7A5e	**A2 12**	LDX #12	sinon (BANQUE changée), indexe le message "LOAD"
C7A7e	**20 64 D3**	JSR D364	et l'affiche
C7AAe	**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)
C7ADe	**58**	CLI	autorise les interruptions
C7AEe	**90 09**	BCC C7B9	simple RTS en C7B9 si "RETURN" a été tapé
C7B0e	**68 68**	PLA PLA	si "ESC", dépile 5 octets au lieu de 2 (pour retourner à l'interpréteur,
C7B2e	**68 68**	PLA PLA	il faut retirer de la pile les adresses de retour correspondant à 2
C7B4e	**68**	PLA	niveaux de JSR soit 4 octets, plus un octet mis sur la pile par un PLP)
C7B5e	20 **06 D2**	JSR D206	effectue un retour à la ligne (avec un JSR au lieu d'un JMP afin de
C7B8e	**38**	SEC	pouvoir exécuter le SEC qui suit. En effet la routine la routine D206
C7B9e	**60**	RTS	met C à 0 or pour témoigner d'une sortie par "ESC" il faut avoir C = 1)

De C7BAe à C7FFe, nettoyage code inutile, remplacé par **70 NOPs.** On peut regretter que Ray ait placé sa routine au milieu de la zone libre. Mais cela peut facilement s'arranger si nécessaire.

BANQUE n°6: INIT

Cette BANQUE se trouve à partir du #5B (quatre vingt onzième) secteur de la disquette MASTER.

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

EXÉCUTION DE LA 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 101, 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)
C6ADf	12	nombre de secteurs par piste + #01
C6AEf	00	nombre de pistes/face (repris dans F5) [et nombre de faces]
C6AFf	00	nombre de faces: #00 si une face et #01 si deux faces
C6B0/C6F8		autres messages

Informations non documentées

Le nombre maximum de secteurs par disquette est maintenant de 3838 (101 pistes de 19 secteurs, double face). Ce chiffre ne peut être atteint qu'avec EUPHORIC qui ne connaît pas la limite physique de 83 pistes des lecteurs de disquettes. 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 de la version 1.0 était sévèrement boguée. Attention donc aux vielles disquettes double face, dont l'indicateur “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 corriger ces vielles disquettes, il faut forcer "manuellement" le b7 du dixième octet du deuxième 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.

Utilisation en "Langage Machine"

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

C404f 4C 82 C4 JMP C482 qui est le début proprement dit

Messages externes de la BANQUE n°6
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

C407f 0D 0A 46 6F 72 6D 61 74 20 28 59 2F 4E 29 **BA**
01 CRLFFormat_(Y/N)**:**

C416f 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**
02 CRLFLFName:_____XX/XX/X**X**

C433f 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**
03 CRLFInit_another_disc_(Y/N)**:**

C44Df 0D 0A 0A 4D 61 73 74 65 72 20 64 69 73 63 20 28 59 2F 4E 29 **BA**
04 CRLFLFMaster_disc_(Y/N)**:**

C462f 0D 0A 0A 49 6E 69 74 20 73 74 61 74 65 6D 65 6E 74 **BA**
05 CRLFLFInit_statement**:**

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

Table des secteurs réservés

C474f 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

C47Cf 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

C47Ff 4C 23 DE JMP DE23 "SYNTAX_ERROR"

Début proprement dit: analyse de syntaxe

Le sous-programme 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 correspondant au lecteur qui sera utilisé (si besoin, cette valeur sera prise par défaut). Le sous-programme 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 sous-programme lit dans la table des drives actifs TABDRV (C039/C03C), le nombre de faces correspondant 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 correspondant 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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
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 **66**	CPX #66	teste si le nombre de pistes par face est >= #66 (102)

Ceci constitue mon extension "BIGDISK" (maximum 101 pistes par face au lieu de 99, utilisable avec EUPHORIC, les lecteurs de disquettes 3"1/2 restants quand à eux limités à 83 pistes par face et les lecteurs 3" à 44 pistes par face). Un octet différent: #64 (100) devient #66 (102).

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 correspondant à ce lecteur dans la table des drives actifs TABDRV
C4BFf	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à

			#39), sinon C = 1, Y et X inchangés
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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
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 sous-programme C4DE/C4F1, remplit le buffer BUF2 de #00, puis de #FF à partir de la position #10 (six septième octet), initialise à #01 le nombre de secteurs de directory (à la position #08) et à #FF le premier 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 (dix septième 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
C4ECf	8E 08 C2	STX C208	nombre de secteurs de directory = 1
C4EFf	8D 00 C2	STA C200	le premier octet de bitmap doit toujours contenir #FF

Calcule le nombre total de secteurs

Le sous-programme 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).

C4F2f	AD AE C6	LDA C6AE	nombre de pistes par face et nombre de faces
C4F5f	29 7F	AND #7F	élimine b7 (nombre de faces)
C4F7f	AA	TAX	X = nombre de pistes par face
C4F8f	A9 00	LDA #00	mise à zéro LL du totalisateur
C4FAf	A8	TAY	mise à zéro HH du totalisateur

C4FBf	18	CLC	calcule AY =
C4FCf	6D AC C6	ADC C6AC	nombre de secteurs par piste (C6AC)
C4FFf	90 01	BCC C502	que multiplie
C501f	C8	INY	nombre de pistes par face (X)
C502f	CA	DEX	= nombre de secteurs par face
C503f	D0 F6	BNE C4FB	
C505f	2C AE C6	BIT C6AE	teste s'il y a deux faces
C508f	10 06	BPL C510	continue en C510 si "Simple face"
C50Af	0A	ASL	
C50Bf	48	PHA	
C50Cf	98	TYA	si "double face", multiplie le résultat par deux
C50Df	2A	ROL	
C50Ef	A8	TAY	
C50Ff	68	PLA	

Teste la validité de AY = nombre total de secteurs

Le début de ce sous-programme (contrôle du nombre total de secteurs) était largement périmé, puisque la double bitmap de Ray autorise au moins 3838 secteurs, chiffre qui peut tout juste être atteint dans les limites maximales actuelles (101 pistes de 19 secteurs, double face pour l'extension "**BIGDISK**" à utiliser seulement avec EUPHORIC).

J'ai donc supprimé ce contrôle, ce qui dégage 9 octets que j'ai utilisés pour déplacer une routine que Ray avait ajoutée de F638 à F63F, dans le NOYAU (STX 3115 & JMP E635):

C510f	**4C 19 C5**	JMP C519	by-passe les 6 octets suivants
C513f	**8E 15 31**	STX 3115	pour remplacer le STX écrasé en C5AE/C5B0 pour introduire un
C516f	**4C 35 E6**	JMP E635	détour vers un sous-programme complémentaire de Ray en E635
C519f	...		reprise du cours normal de la commande INIT

Le sous-programme C519/C51E écrit le nombre total de secteurs AY dans BUF2 aux positions 02/03 (nombre de secteurs libres).

C519f	8D 02 C2	STA C202	écrit le nombre total de secteurs dans BUF2
C51Cf	8C 03 C2	STY C203	aux positions #02/03 (nombre de secteurs libres)

Formate la disquette?

Le sous-programme C51F/C52B affiche "CRLFFormat_(Y/N)**:**", saisit une touche, si "Y", formate la ou les faces (en C64A), sinon passe directement à la suite.

C51Ff	A2 00	LDX #00	indexe le message n° #01 "CRLFFormat_(Y/N)**:**"
C521f	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C524f	20 0F C6	JSR C60F	saisie de touche, retourne avec C = 1 si "Y"
C527f	90 03	BCC C52C	saute l'instruction suivante si ce n'est pas "Y"

C529f 20 4A C6 JSR C64A si "Y", formate la ou les faces

Elabore un secteur système et l'écrit sur la disquette

Le sous-programme C52C/C566:

- rempli le buffer BUF1 de #00,

- affiche le message: "CRLFLFName:_____XX/XX/X**X**",

- appelle le sous-programme XLINPU pour saisir DNAME (21 caractères au maximum),

- affiche le message "CRLFLFInit_statement**:**",

- appelle le sous-programme XLINPU pour saisir INIST (60 caractères au maximum),

- 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+1) ème 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	XAFXGAU 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 caractères 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+1) ème message externe terminé par "caractère + 128"
C557f	A9 3C	LDA #3C	A = 60 caractères 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 premier secteur de directory et l'écrit sur la disquette

Le sous-programme C567/C575 rempli BUF1 de #00, copie #10 (qui vise le dix septième octet du secteur de directory) en C102 (n° de l'octet de la première 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 dix septième octet du secteur de directory
C56Cf	8D 02 C1	STA C102	n° de l'octet de la première 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 deuxième secteur en attente en RAM

Le sous-programme C576/C5B3 copie le nombre de pistes/face + nombre de faces (C6AE) dans BUF2 à la position #09. Puis le sous-programme 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 #63 (99 secteurs à copier), sinon, affiche "S" et initialise F9 à #08 (8 secteurs à copier). Le sous-programme place #01 si "Slave" ou #00 si "Master" à la position #16 du deuxième 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 deuxième secteur en RAM (c'est à dire en 3115).

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+1) ème 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"
C593f	90 04	BCC C599	si C = zéro, saute les deux instructions suivantes
C595f	A0 4D	LDY #4D	"M" pour "Master" (C = 1)
C597f	A2 **63**	LDX #63	99 = nombre de secteurs de la disquette Master à copier en RAM

En C598, adaptation du nombre de secteurs à copier pour formater une disquette Master, pour tenir compte de la nouvelle BANQUE n°7 (1 octet différent).

C599f	86 F9	STX F9	F9 = nombre de secteurs à copier
C59Bf	A9 00	LDA #00	calcule A = #01 si "Slave" ou #00 si "Master"
C59Df	2A	ROL	sauve ce résultat à la position #16 (vingt troisième octet)
C59Ef	49 01	EOR #01	du deuxième secteur en attente en RAM (secteur de boot),
C5A0f	8D 16 31	STA 3116	ainsi que dans BUF2 à la position #0A
C5A3f	8D 0A C2	STA C20A	(flag type de disquette)
C5A6f	98	TYA	"S" ou "M"
C5A7f	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C5AAf	AE AC C6	LDX C6AC	nombre de secteurs par piste
C5ADf	E8	INX	plus un

En C5AE, modification par Ray du sous-programme pour ajouter une sauvegarde de la deuxième bitmap (3 octets différents). Le STX 3115 qui était situé en C5AE a été remplacé par:

C5AEf	**20 13 C5**	JSR C513	qui effectue un petit détour en C513 où on retrouve le STX 3115 manquant et où l'on poursuit vers une routine complémentaire de Ray, permettant de gérer la deuxième bitmap.
C5B1f	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Copie F9 secteurs de la RAM sur la disquette

Le sous-programme 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.

C5B4f	A9 00	LDA #00	
C5B6f	A0 30	LDY #30	RWBUF = #3000 = début secteurs en attente en RAM
C5B8f	8D 03 C0	STA C003	
C5BBf	8C 04 C0	STY C004	
C5BEf	A0 00	LDY #00	
C5C0f	8C 01 C0	STY C001	n° de PISTE active = #00
C5C3f	C8	INY	
C5C4f	78	SEI	interdit les interruptions
C5C5f	8C 02 C0	STY C002	n° de SECTEUR actif = #01
C5C8f	AD 01 C0	LDA C001	n° de PISTE active
C5CBf	20 2D DD	JSR DD2D	XCREAY marque dans la bitmap en BUF2 que le secteur AY est occupé
C5CEf	20 A4 DA	JSR DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C5D1f	EE 04 C0	INC C004	incrémente HH de RWBUF (page suivante)
C5D4f	AC 02 C0	LDY C002	n° de SECTEUR actif
C5D7f	CC AC C6	CPY C6AC	teste si n° SECTEUR actif < nombre secteurs/piste
C5DAf	90 05	BCC C5E1	si oui, saute les deux instructions suivantes
C5DCf	EE 01 C0	INC C001	incrémente le n° de PISTE active et reset le n° de
C5DFf	A0 00	LDY #00	SECTEUR actif (pour avoir #01 en début de boucle)
C5E1f	C8	INY	secteur suivant

C5E2f	C6 F9	DEC F9	décrémente le nombre de secteurs à copier
C5E4f	D0 DF	BNE C5C5	et reboucle en tant qu'il en reste à copier
C5E6f	58	CLI	fini, autorise les interruptions

Termine l'élaboration du secteur de bitmap et le sauve

Le sous-programme 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.

C5E7f	A2 07	LDX #07	index pour lire les 8 octets de la table C474
C5E9f	86 F9	STX F9	sauve X dans F9
C5EBf	A6 F9	LDX F9	récupère X en début de boucle
C5EDf	A9 14	LDA #14	
C5EFf	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
C5F2f	20 2D DD	JSR DD2D	XCREAY marque dans la bitmap en BUF2 que le secteur AY est occupé
C5F5f	C6 F9	DEC F9	octet précédent
C5F7f	10 F2	BPL C5EB	reboucle tant qu'il en reste

En C5F9, autre modification de Ray pour gérer la double bitmap (2 octets différents): le JSR DA8A (XSMAP, sauve BUF2 dans le secteur de bitmap sur la disquette) a été remplacé par un JSR DC89 (sauve BUF2 dans le premier secteur de bitmap sur la disquette, plus spécifique dans ce cas).

C5F9f	20 **89 DC**	JSR DC89	écrit BUF2 dans le premier secteur de bitmap sur la disquette

Une autre? (même format)

Le sous-programme C5FC/C60E affiche "CRLFInit_another_disc_(Y/N)**:**", saisit une touche. Si c'est "Y", reprend en C51F pour formatage identique, sinon retourne.

C5FCf	A2 02	LDX #02	indexe le message "CRLFInit_another_disc_(Y/N)**:**"
C5FEf	20 64 D3	JSR D364	XAFSC affiche (X+1) ème message externe terminé par "caractère + 128"
C601f	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

En C609, dernière modification de Ray pour gérer la deuxième bitmap (2 octets différents): le JMP C51F (reprend en C51F pour formatage identique) a été remplacé par un JMP C4DE.

C609f	4C **DE C4**	JMP C4DE	le rebouclage pour effectuer un autre formatage identique reprend maintenant au début de l'élaboration des bitmaps.
C60Cf	4C 06 D2	JMP D206	CBF0/ROM va à la ligne et termine

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

Le sous-programme C60F/C623 attend qu'une touche soit pressée et revient avec le code ASCII

correspondant (lettre convertie en MAJUSCULE). Si c'est un "ESC", dépile une adresse de retour, sort de la commande INIT et retourne au sous-programme 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 correspondant, sinon N = 0
C612f	10 FB	BPL C60F	reprend la saisie tant que pas de touche pressée
C614f	20 A1 D3	JSR D3A1	XminMAJ convertit (si nécessaire) 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 sous-programme C624/C649 appelle les sous-programmes ED36 XLINPU (routine de saisie de chaîne) et D73E (XCURON rend le 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 sous-programme 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 principale de LINPUT (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 et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM.
C62Ff	20 3E D7	JSR D73E	XCURON rend le 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 sous-programme 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 sous-programme D740 XCUROFF cache le curseur (= vidéo normale)

- rétablit en C6AE le nombre de pistes/face (hélas sans tenir compte du nombre de faces!)

- appelle le sous-programme C745 qui formate la première face, si une seule face,

- affiche "LFLFCRFormating_complete" et retourne.

- Sinon, affiche "<- <- <- <- <- <- <- <- <- <- 1_Track_00",

- appelle le sous-programme C745 qui formate la deuxième face,

- affiche "LFLFCRFormating_complete"

- et retourne.

Première partie de la correction par Ray de la fameuse bogue de **INIT** (9 octets différents):

```
**C64A**f  **AD** AE C6  LDA C6AE    reprend C6AE sans en modifier le contenu
C64Df   **29 7F**   AND #7F     calcule dans A le nombre de pistes par face
C64Ff   **EA**      NOP         en forçant simplement le b7 à zéro: c'en est fini de la bogue!

        Au lieu de  ASL C6AE    b7 -> C (à zéro si "Simple", à 1 si "Double" face)
                    LDA #00     force A à zéro
                    ROL         C -> b0 de A et b7 de A (nul) -> C (important)

C650f   8D AF C6    STA C6AF    C6AF = #00 si une face et #01 si deux faces
C653f   A9 B0       LDA #B0     AY = adresse C6B0 de la chaîne:
C655f   A0 C6       LDY #C6     "CRLFLFFormating_Side_0_Track_00"
C657f   20 37 D6    JSR D637    AFSTR affiche chaîne terminée par 0 d'adresse AY
C65Af   20 40 D7    JSR D740    XCUROFF cache le curseur( = vidéo normale)

C65Df   **EA EA EA**  NOP NOP NOP
C660f   20 45 C7    JSR C745    formate la première face (car C vaut toujours 0)
C663f   AD **AE** C6  LDA C6AE  teste si une seule face (b7 à zéro)
```

```
C666f   10 0B       BPL C673        si oui, continue en C673

        Au lieu de  LSR C6AE        rétablit nombre de pistes/face (sans nombre face)
                    JSR C745        formate la première face (car C vaut toujours 0)
                    LDA C6AF        teste si une seule face
                    BEQ C673        si oui, continue en C673

C668f   A9 CD       LDA #CD         sinon, AY = adresse C6CD pour chaîne:
C66Af   A0 C6       LDY #C6         "<- <- <- <- <- <- <- <- <- <-1_Track_00"
C66Cf   20 37 D6    JSR D637        AFSTR affiche chaîne terminée par 0 d'adresse AY
C66Ff   38          SEC             pour la deuxième face
C670f   20 45 C7    JSR C745        formate la deuxième face

C673f   A9 E2       LDA #E2         AY = adresse C6E2 pour chaîne:
C675f   A0 C6       LDY #C6         "LFLFCR.Formating_complete BRK"
C677f   4C 37 D6    JMP D637        AFSTR affiche chaîne terminée par 0 d'adresse AY et retourne.
```

Table de formatage (C67A à C6A3)

```
C67Af   28 4E 0C 00 03 F6 01 FC 28 4E FF                (soit #60 = 96 octets au total)
C685f   0C 00 03 F5                                     (soit 15 octets)
C689f   01 FE 01 00 01 00 01 00 01 01 01 F7             (FE pp ff ss 01 CRC, soit 6 octets)
C695f   16 4E 0C 00 03 F5                               (soit 37 octets)
C69Bf   01 FB 00 00 01 F7                               (soit 258 octets)
C6A1f   28 4E FF                (soit 40, 30 ou 12 octets selon le nombre de secteurs/piste)
```

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

La deuxième (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 je l'indique plus loin.

Variables internes à la BANQUE n°6

```
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)
C6ADf 12 nombre de secteurs par piste + #01
C6AEf 00 nombre de pistes/face (repris dans F5) [et nombre de faces]
C6AFf 00 nombre de faces: #00 si une face et #01 si deux faces

Autres messages (C6B0 à C6F8) terminés par #00
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

C6B0f 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**
01 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**

Rappel: **BRK** = pour marquer la présence d'un #00 à la fin de certains messages
CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

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

Le sous-programme 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 sous-programme C73A. 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 sous-programme 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 premier 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	correspondant 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 premier n° de secteur de la piste suivante

Le sous-programme 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.
C742f	85 F7	STA F7	et replace le résultat dans F7
C744f	60	RTS	

Rappels sur la structure d'une piste

Une piste SEDORIC 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 (SEDORIC, 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 SEDORIC, 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 de SEDORIC, #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 SEDORIC, 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 SEDORIC.

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 SEDORIC, 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 SEDORIC, 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 SEDORIC, 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 sous-programme 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.

C745f	08	PHP	sauvegarde les indicateurs 6502 dont C
C746f	08	PHP	idem une deuxième fois (génial, voir plus loin)
C747f	AD AC C6	LDA C6AC	
C74Af	85 F6	STA F6	F6 = nombre de secteurs par piste
C74Cf	8D AD C6	STA C6AD	
C74Ff	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:

C752f	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 maximale)
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, pour **SEDORIC, toutes versions**: Y = #28 (soit 40, valeur pour A = 16 ou 17)
C75Ef	A0 2F	LDA #2F	et pour **STRATORIC, toutes versions**: Y = #2F (soit 47, valeur pour A = 16 ou 17)
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	

Deuxième partie de la correction par Ray de la fameuse bogue de INIT (un octet différent): En C76A, le LDA C6AE (nombre de piste par face **et** nombre de faces) a été modifié en:

C76Af	AD **AF** C6	LDA C6AF	nombre de faces
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 première face ou #01 si deuxième 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	pour **SEDORIC, toutes versions**: teste si A >= #12 (c'est à dire, si vaut 18 ou 19)
C791f	C9 11	CMP #11	pour **STRATORIC, toutes versions**: teste si A >= #11 (c'est à dire, si vaut 17 ou 18)
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 sous-programme C795/C797 appelle le sous-programme C7E3 qui élabore un en-tête de piste de 96 octets, au début du tampon (de 9800 à 985F), selon les valeurs de la première 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 première 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 sous-programme 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 sous-programme C7A0/C7A8 appelle le sous-programme C7E3 qui élabore une "zone secteur" selon les valeurs de la deuxième 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 deuxième 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 sous-programme 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.

C7A9f	A9 4E	LDA #4E	A = #4E
C7ABf	91 0A	STA (0A),Y	écrit #4E selon l'adresse en 0A/0B + Y
C7ADf	C8	INY	indexe la position suivante
C7AEf	D0 FB	BNE C7AB	et reboucle en C7AB tant que Y n'est pas nul
C7B0f	E6 0B	INC 0B	page suivante
C7B2f	A6 0B	LDX 0B	pour test
C7B4f	EC A7 C6	CPX C6A7	teste si HH atteint la valeur en C6A7 (#B1)
C7B7f	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 sous-programme C7B9/C7E2 envoie la commande #08 (positionnement sur piste #00) au sous-programme XRWTS (CFCD, routine de gestion des lecteurs). A l'aide d'une boucle, appelle le sous-programme C6F9 qui met à jour les n° de piste, de face et de secteur, envoie la commande #F0 (commande "formater une piste") au sous-programme 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.

C7B9f	A2 08	LDX #08	probablement pour positionnement ou initialisation
C7BBf	20 CD CF	JSR CFCD	XRWTS routine de gestion des lecteurs, X = commande
C7BEf	20 F9 C6	JSR C6F9	met à jour les n° de piste, de face et de secteur
C7C1f	A2 F0	LDX #F0	probablement commande "formater une piste"
C7C3f	20 75 DA	JSR DA75	XRWTS X = commande et traite une éventuelle erreur
C7C6f	A9 08	LDA #08	A = "flèche gauche" pour reculer de 2 positions
C7C8f	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C7CBf	20 2A D6	JSR D62A	idem
C7CEf	A2 30	LDX #30	X = "0"
C7D0f	8E 4C C0	STX C04C	DEFAFF, code ASCII devant les nombres décimaux
C7D3f	AD 01 C0	LDA C001	n° de PISTE formatée à afficher
C7D6f	29 7F	AND #7F	élimine le b7 indiquant la face
C7D8f	20 4E D7	JSR D74E	affichage en décimal sur 2 digits d'un nombre A
C7DBf	EE 01 C0	INC C001	n° de PISTE active suivante à écrire
C7DEf	C6 F5	DEC F5	nombre de pistes par face
C7E0f	D0 DC	BNE C7BE	et reboucle en C7BE tant qu'il en reste
C7E2f	60	RTS	

Copie une suite d'octets dans le tampon de formatage

A l'aide d'une boucle, le sous-programme 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 premier 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.

C7E3f	BD 7A C6	LDA C67A,X	lecture d'un octet dans la table, à la position X.
C7E6f	E8	INX	indexe la position suivante dans la table
C7E7f	C9 FF	CMP #FF	l'octet lu est-il #FF?
C7E9f	F0 13	BEQ C7FE	si oui, simple RTS (seule sortie de ce sous-programme)
C7EBf	85 0C	STA 0C	sauve l'octet lu en 0C (nombre d'octets à copier)
C7EDf	BD 7A C6	LDA C67A,X	lecture de l'octet suivant dans la table
C7F0f	E8	INX	indexe la position suivante dans la table
C7F1f	91 0A	STA (0A),Y	copie l'octet lu dans le buffer, à la position Y
C7F3f	C8	INY	indexe la position suivante dans le buffer
C7F4f	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)
C7F6f	E6 0B	INC 0B	indexe la page suivante du buffer (incrémente HH)
C7F8f	C6 0C	DEC 0C	décrémente le nombre d'octets à copier
C7FAf	D0 F5	BNE C7F1	reboucle en C7F1 en tant qu'il en reste à copier
C7FCf	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).

C7FEf 60 RTS

C7FFf 00 BRK

NOUVELLE BANQUE n°7: CHKSUM EXT PROT STATUS SYSTEM UNPROT VISUHIRES

Cette BANQUE se trouve à partir du #60 (quatre vingt seizième) secteur de la disquette MASTER.

De C400 à C403, Dans touts les BANQUES ces 4 octets sont réservés pour les vecteurs des messages.

C400g	00 00	EXTER	adresse des messages d'erreur externes (néant)
C402g	00 00	EXTMS	adresse des messages externes (néant)

EXÉCUTION DE LA COMMANDE SEDORIC EXT

Cette commande, qui était située à l'origine dans le NOYAU de E9ED à EA3A, se trouve maintenant dans la BANQUE n°7 de C404 à C451.

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. 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

Dans les versions précédentes de SEDORIC, une bogue faisait que la validité du troisième caractère de l'extension n'est pas vérifiée. Il était donc possible de mettre n'importe quoi comme troisième caractère, ce qui n'était pas sans risque: une extension à "CO?" est acceptée, mais illégale! Cette bogue a été corrigée.

Enfin, petite curiosité, EXT PRINT est accepté, mais pas EXT print. Je rappelle que l'utilisation des minuscules dans les commandes SEDORIC n'est plus supportée par la version 3.0, même si cela marche toujours dans certains cas. C'était le seul moyen de résoudre plusieurs dizaines de bogues et de trouver de la place pour de nouvelles commandes.

Analyse de la syntaxe et des paramètres

C404g	D0 0C	BNE C412	continue en C412 s'il y a des paramètres

Re-initialise l'extension courante avec "COM"

C406g	A2 02	LDX #02	si pas de paramètre, remet COM par défaut

C408g	BD FD CC	LDA CCFD,X	lit 3 caractères à partir de CCFD (EXT par défaut, c'est à dire COM)
C40Bg	9D F7 CC	STA CCF7,X	et les copie à partir de CCF7 (EXT courante)
C40Eg	CA	DEX	caractère précédent
C40Fg	10 F7	BPL C408	reboucle en C408 tant qu'il en reste à copier
C411g	60	RTS	et retourne

Il y a un paramètre

C412g	C9 BA	CMP #BA	le paramètre est-il un "?" (le token PRINT)
C414g	D0 11	BNE C427	sinon, poursuit l'analyse de syntaxe en C427

Si c'est un "?", affiche l'extension courante

C416g	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!
C418g	B9 FA CB	LDA CBFA,Y	lus à partir de CCF7
C41Bg	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C41Eg	C8	INY	caractère suivant
C41Fg	D0 F7	BNE C418	jusqu'à ce que Y soit nul
C421g	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C424g	4C 3A D3	JMP D33A	JSR 00E2/ROM incrémente TXTPTR, lit un caractère (CHARGET), 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

C427g	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
C42Ag	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
C42Dg	C9 03	CMP #03	cette chaîne a t-elle 3 caractères?
C42Fg	D0 1E	BNE C44F	sinon, "INVALID_FILE_NAME_ERROR"
C431g	A0 **03**	LDY #03	prépare pour index #02 en début de boucle (ici, la **bogue** a été corrigée, la valeur d'origine était #**02**)
C433g	88	DEY	on aura donc Y = 2, 1, et 0
C434g	30 0B	BMI C441	sortie de boucle en C441 lorsqu'Y devient négatif
C436g	B1 91	LDA (91),Y	lit un caractère de la chaîne (par la fin)
C438g	20 A1 D3	JSR D3A1	XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
C43Bg	20 C7 D5	JSR D5C7	vérifie que caractère est valide (lettre ou chiffre)
C43Eg	4C 33 C4	JMP C433	reboucle obligatoirement en C433

Copie cette extension en RAM overlay

```
C441g  A0 02       LDY #02       indexe pour 3 caractères (Y = 2, 1, et 0)
C443g  B1 91       LDA (91),Y    lit un caractère de la chaîne (par la fin)
C445g  20 A1 D3    JSR D3A1      XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
C448g  99 F7 CC    STA CCF7,Y    et l'écrit dans la zone CCF7 à CCF9 (EXT courante)
C44Bg  88          DEY           caractère précédent
C44Cg  10 F5       BPL C443      reboucle tant que Y n'est pas négatif
C44Eg  60          RTS           et termine
C44Fg  4C AC D5    JMP D5AC      "INVALID_FILE_NAME_ERROR"
```

Vectorisation de la commande **VISUHIRES**

```
C452g  4C DF C6    JMP C6DF      entrée de VISUHIRES
```

Vectorisation de la commande **STATUS**

```
C455g  4C FF C5    JMP C5FF      entrée de STATUS
```

Vectorisation de la commande **PROT**

```
C458g  4C A1 C6    JMP C6A1      entrée de PROT
```

Vectorisation de la commande **UNPROT**

```
C45Bg  4C A4 C6    JMP C6A4      entrée de UNPROT
```

Vectorisation de la commande **SYSTEM**

```
C45Eg  4C D3 C6    JMP C6D3      entrée de SYSTEM
```

Copyright " © 1996 André Chéramy "

```
C461g  20 60 20 31 39 39 36 20 41 6E 64 72 7B 20 43 68 7B 72 61 6D 79 20
01     _©_1996_André_Chéramy_
```

EXÉCUTION DE LA COMMANDE SEDORIC CHKSUM

Il s'agit d'une nouvelle commande située dans la BANQUE n°7 de C477 à C5F7, avec entrée en C47A.

Syntaxe

CHKSUM nom_de_fichier_ambigu(,AUTO)

Cette commande vérifie que le ou les fichiers indiqués peuvent être lus sur la disquette et calcule la checksum de chacun.

Exemples: CHKSUM"TOTO.SOS"
CHKSUM"BANQUE?"

CHKSUM"*.TXT",AUTO

Le paramètre ",AUTO" doit être tapé en MAJUSCULES. S'il est omis, il faudra appuyer sur une touche pour passer au fichier suivant. S'il est présent, l'affichage sera automatique. Toutefois, il peut être interrompu par pression sur la touche <ESPACE> et repris par une pression sur n'importe quelle touche sauf <ESPACE> et <ESC>. La touche <ESC> permet d'abandonner dans tous les cas.

Cette checksum, ou somme de tous les octets, est caractéristique de chaque fichier. Elle permet donc de savoir si un fichier a été modifié. Par exemple si vous faites passer un fichier SEDORIC dans le monde PC, par une ligne RS232 ou à l'aide d'un des nombreux utilitaires qui ont été développés autour d'EUPHORIC, vous pourrez vous assurer que le fichier arrivé à destination a toujours la même checksum. Il existe plusieurs programmes PC permettant de calculer la checksum d'un fichier, mais je vous recommande le petit utilitaire CHKSUM.EXE, écrit par mon fils Robert Chéramy.

C477g	4C 23 DE	JMP DE23	SYNTAX_ERROR

Entrée réelle de la commande: analyse de syntaxe

C47Ag	78	SEI	Interdit les interruptions
C47Bg	20 51 D4	JSR D451	lit un nom_de_fichier_ambigu à TXTPTR
C47Eg	A0 00	LDY #00	"non AUTO" par défaut
C480g	84 00	STY 00	flag AUTO/non AUTO
C482g	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
C485g	F0 0E	BEQ C495	continue en C495 si pas de paramètre
C487g	20 2C D2	JSR D22C	demande une "," à TXTPTR et lit le caractère suivant
C48Ag	C9 C7	CMP #C7	est-ce le token "AUTO" ?
C48Cg	D0 E9	BNE C477	SYNTAX_ERROR si ce n'est pas le cas
C48Eg	85 00	STA 00	flag AUTO = token "AUTO"
C490g	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C493g	D0 E2	BNE C477	SYNTAX_ERROR si pas fin d'instruction

Recherche le ou les fichiers indiqués

C495g	20 30 DB	JSR DB30	charge catalogue et met à jour POSNMX
C498g	F0 22	BEQ C4BC	rien trouvé = terminé
C49Ag	BD 0C C3	LDA C30C,X	coordonnées du
C49Dg	BC 0D C3	LDY C30D,X	descripteur approprié
C4A0g	20 5D DA	JSR DA5D	charge ce descripteur
C4A3g	A2 02	LDX #02	début
C4A5g	BD 00 C1	LDA C100,X	boucle
C4A8g	C9 FF	CMP #FF	de
C4AAg	F0 12	BEQ C4BE	recherche

```
C4ACg  E8        INX        du
C4ADg  D0 F6     BNE C4A5   flag #FF
C4AFg  AD 00 C1  LDA C100   dans
C4B2g  AC 01 C1  LDY C101   le
C4B5g  D0 E9     BNE C4A0   descripteur
C4B7g  20 41 DB  JSR DB41   cherche entrée suivante
C4BAg  D0 DE     BNE C49A   trouvée = reboucle
C4BCg  58        CLI        rien trouvé = terminé
C4BDg  60        RTS        sortie de la commande CHKSUM
```

Un fichier a été trouvé: affiche son nom et son status

```
C4BEg  8A        TXA        suite avec X = POSNMX
C4BFg  48        PHA
C4C0g  20 B4 DA  JSR DAB4   affiche le nom du fichier
C4C3g  68        PLA
C4C4g  AA        TAX
C4C5g  20 28 D6  JSR D628   affiche un espace
C4C8g  BD 03 C1  LDA C103,X
C4CBg  20 13 D6  JSR D613   affiche l'adresse de début (HH puis LL)
C4CEg  BD 02 C1  LDA C102,X
C4D1g  20 13 D6  JSR D613
C4D4g  20 28 D6  JSR D628   affiche un espace
C4D7g  38        SEC
C4D8g  BD 04 C1  LDA C104,X
C4DBg  48        PHA
C4DCg  FD 02 C1  SBC C102,X calcule longueur (C04F/50) = adresse de fin - adresse de début
C4DFg  8D 4F C0  STA C04F
C4E2g  BD 05 C1  LDA C105,X
C4E5g  A8        TAY
C4E6g  FD 03 C1  SBC C103,X
C4E9g  8D 50 C0  STA C050
C4ECg  98        TYA
C4EDg  20 13 D6  JSR D613   affiche adresse de fin (HH puis LL)
C4F0g  68        PLA
C4F1g  20 13 D6  JSR D613
C4F4g  20 28 D6  JSR D628   affiche un espace
C4F7g  BD 01 C1  LDA C101,X
C4FAg  85 F9     STA F9     met FTYPE dans F9
C4FCg  20 13 D6  JSR D613   affiche FTYPE
C4FFg  20 28 D6  JSR D628   affiche un espace
C502g  BD 07 C1  LDA C107,X
C505g  20 13 D6  JSR D613   affiche adresse exécution (HH puis LL)
C508g  BD 06 C1  LDA C106,X
C50Bg  20 13 D6  JSR D613
C50Eg  20 28 D6  JSR D628   affiche un espace
```

Initialise le calcul de la checksum

```
C511g   18          CLC          mise en place des pointeurs pour checksum
C512g   A9 00       LDA #00      LL adresse début de stockage
C514g   85 6A       STA 6A
C516g   8D 03 C0    STA C003     calcule 64/65 = adresse de fin de stockage = adresse de début + longueur
C519g   6D 4F C0    ADC C04F
C51Cg   85 64       STA 64
C51Eg   A9 06       LDA #06      HH adresse début de stockage
C520g   85 6B       STA 6B       6A/6B = 0600 = adresse de début
C522g   A8          TAY          64/65 = adresse de fin de stockage
C523g   88          DEY          C003/C004 = RWBUF =
C524g   8C 04 C0    STY C004     début -#100 pour compenser
C527g   6D 50 C0    ADC C050     la mise à jour en entrée de boucle
C52Ag   85 65       STA 65
C52Cg   A5 F9       LDA F9       teste FTYPE à cause d'une bogue concernant les programmes BASIC
C52Eg   30 06       BMI C536     by-passe si BASIC pour qui l'adresse de fin est déjà incrémentée
C530g   E6 64       INC 64       sinon incrémente
C532g   D0 02       BNE C536
C534g   E6 65       INC 65       adresse de fin de stockage
C536g   BD 08 C1    LDA C108,X
C539g   85 F7       STA F7       nombre de secteurs à charger
C53Bg   BD 09 C1    LDA C109,X
C53Eg   85 F8       STA F8
C540g   8A          TXA
C541g   18          CLC
C542g   69 06       ADC #06      met à jour l'index Y
C544g   A8          TAY
C545g   20 28 E2    JSR E228
```

Chargement du fichier dans la zone de stockage

```
C548g   A5 F7       LDA F7       charge les secteurs complets dans la zone de stockage
C54Ag   D0 02       BNE C54E     décrémente le nombre
C54Cg   C6 F8       DEC F8       de secteurs à charger
C54Eg   C6 F7       DEC F7       c'est à dire le nombre de secteurs complets
C550g   EE 04 C0    INC C004     met à jour RWBUF
C553g   A5 F7       LDA F7       reste-t'il des secteurs
C555g   05 F8       ORA F8       complets à charger ?
C557g   F0 09       BEQ C562     non, by-passe
C559g   20 28 E2    JSR E228     oui, met à jour l'index Y
C55Cg   20 50 E2    JSR E250     lit les coordonnées du secteur et le charge
C55Fg   4C 48 C5    JMP C548     rebouclage forcé

C562g   AD 03 C0    LDA C003     charge le dernier secteur qui est probablement incomplet dans BUF2
C565g   AE 04 C0    LDX C004
C568g   85 F5       STA F5       l'adresse de stockage dans RWBUF est sauvée dans F5/F6
C56Ag   86 F6       STX F6
C56Cg   20 28 E2    JSR E228     met à jour l'index Y
C56Fg   98          TYA          et l'empile
```

```
C570g   48          PHA
C571g   A9 00       LDA #00
C573g   A2 C2       LDX #C2     place dans RWBUF l'adresse de BUF2 (C200)
C575g   8D 03 C0    STA C003    pour chargement intermédiaire
C578g   8E 04 C0    STX C004
C57Bg   20 50 E2    JSR E250    lit les coordonnées du dernier secteur et le charge dans BUF2
C57Eg   A0 FF       LDY #FF     met à jour pour entrée de boucle
C580g   C8          INY         recopie un à un les octets
C581g   B9 00 C2    LDA C200,Y  significatifs (LL de la longueur)
C584g   91 F5       STA (F5),Y  de BUF2 vers la fin de stockage
C586g   CC 4F C0    CPY C04F
C589g   D0 F5       BNE C580
```

Et maintenant on calcule la checksum

```
C58Bg   A0 00       LDY #00
C58Dg   84 68       STY 68
C58Fg   84 69       STY 69      mise à zéro du totalisateur
C591g   A5 6A       LDA 6A
C593g   C5 64       CMP 64      teste si la fin du stockage est atteinte
C595g   A5 6B       LDA 6B      rappel:     6A/6B = début de stockage, puis pointeur de stockage
C597g   E5 65       SBC 65                  64/65 = fin de stockage
C599g   B0 14       BCS C5AF    si c'est fini, on continue en C5AF
C59Bg   A5 68       LDA 68
C59Dg   71 6A       ADC (6A),Y  calcule la checksum
C59Fg   85 68       STA 68
C5A1g   A5 69       LDA 69      et met à jour totalisateur 68/69
C5A3g   69 00       ADC #00
C5A5g   85 69       STA 69
C5A7g   E6 6A       INC 6A      incrémente le pointeur
C5A9g   D0 E6       BNE C591    qui vise l'octet suivant
C5ABg   E6 6B       INC 6B
C5ADg   D0 E2       BNE C591    rebouclage forcé en C591
```

Puis on l'affiche

```
C5AFg   20 28 D6    JSR D628    affiche un espace
C5B2g   A5 69       LDA 69
C5B4g   20 13 D6    JSR D613    affiche la checksum
C5B7g   A5 68       LDA 68
C5B9g   20 13 D6    JSR D613
C5BCg   20 06 D2    JSR D206    affiche un CRLF
```

Affichage automatique ou pas?

```
C5BFg   58          CLI         pour test de touche ultérieur
C5C0g   A4 00       LDY 00      teste le flag "AUTO/non AUTO"
C5C2g   F0 29       BEQ C5ED    continue en C5ED si "non AUTO"
```

Affichage automatique

C5C4g	A0 FF	LDY #FF	pour temporisateur
C5C6g	88	DEY	il y aura 255 essais de touche
C5C7g	F0 14	BEQ C5DD	timer out: passe au fichier suivant
C5C9g	20 02 D3	JSR D302	touche ?
C5CCg	C9 1B	CMP #1B	est-ce un ESC ?
C5CEg	F0 26	BEQ C5F6	si oui, abandonne
C5D0g	C9 20	CMP #20	sinon, est-ce un espace ?
C5D2g	D0 F2	BNE C5C6	non, reboucle pour un autre essai
C5D4g	20 02 D3	JSR D302	oui, se met en stand-by
C5D7g	10 FB	BPL C5D4	jusqu'à nouvelle touche
C5D9g	C9 20	CMP #20	nouvelle touche reçue, est-ce un espace
C5DBg	F0 F7	BEQ C5D4	oui, retour au stand-by (dispositif de sécurité)

Fichier suivant

C5DDg	78	SEI	non, fin de la pause, passe au suivant (re-interdit les interruptions)
C5DEg	68	PLA	
C5DFg	A8	TAY	
C5E0g	20 28 E2	JSR E228	met à jour l'index Y pour coordonnées suivantes
C5E3g	B0 05	BCS C5EA	les dernières coordonnées du dernier descripteur ont été lues
C5E5g	98	TYA	il y a encore un descripteur
C5E6g	AA	TAX	en cherche le début
C5E7g	4C A5 C4	JMP C4A5	rebouclage forcé pour chercher le flag #FF du fichier “mergé” suivant
C5EAg	4C B7 C4	JMP C4B7	rebouclage forcé pour chercher le fichier suivant

Affichage non automatique

C5EDg	20 02 D3	JSR D302	touche ?
C5F0g	10 FB	BPL C5ED	sinon, stand-by en attente de touche
C5F2g	C9 1B	CMP #1B	touche détectée, est-ce un ESC ?
C5F4g	D0 E7	BNE C5DD	non, fin de la pause, passe au suivant (re-interdit les interruptions)

Abandon

C5F6g	EA	NOP	oui, abandonne
C5F7g	60	RTS	
C5F8g	00	BRK	

EXÉCUTION DE LA COMMANDE SEDORIC STATUS

Cette commande, qui était située à l'origine dans le NOYAU de E62E à E6CF, se trouve maintenant dans la BANQUE n°7 de C5F9 à C6A0, avec entrée en C5FF.

Rappel de la syntaxe

STATUS nom_de_fichier_non_ambigu (,A adresse_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 SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, le ",AUTO" doit être tapé en MAJUSCULES.

C5F9g	4C DD E0	JMP E0DD	FILE_NOT_FOUND_ERROR
C5FCg	4C D2 E2	JMP E2D2	nom_de_fichier IS PROTECTED

Analyse de la syntaxe

C5FFg	20 4F D4	JSR D44F	lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
C602g	20 9E D7	JSR D79E	XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé
C605g	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si fichier n'a pas été trouvé
C608g	F0 EF	BEQ C5F9	si Z = 1, branche vers "FILE_NOT_FOUND_ERROR"
C60Ag	BD 0F C3	LDA C30F,X	teste le dernier octet de “l’entrée” correspondante
C60Dg	30 ED	BMI C5FC	si b7=1, erreur (nom_de _fichier + "IS PROTECTED")
C60Fg	BD 0C C3	LDA C30C,X	
C612g	BC 0D C3	LDY C30D,X	AY = piste/secteur du descripteur principal
C615g	20 5D DA	JSR DA5D	XPBUF1 charge descripteur principal dans BUF1
C618g	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

C61Bg	F0 73	BEQ C690	si fin d'instruction continue en C690 (non AUTO)
C61Dg	20 2C D2	JSR D22C	exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C620g	C9 41	CMP #41	est-ce "A"?
C622g	D0 39	BNE C65D	sinon, poursuit l'analyse en C65D; si oui..
C624g	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C627g	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)
C62Ag	98	TYA	les octets du nombre YA sont inversés
C62Bg	A4 34	LDY 34	YA -> AY (nouvelle adresse de chargement)
C62Dg	48	PHA	empile A = LL de cette adresse (HH gardé dans Y)
C62Eg	85 F6	STA F6	
C630g	84 F7	STY F7	sauve l'adresse AY dans F6/F7
C632g	38	SEC	prépare soustraction adresse_de_fin - adresse_de_début
C633g	AD 06 C1	LDA C106	LL de l'ancienne adresse de fin
C636g	ED 04 C1	SBC C104	LL de l'ancienne adresse de début
C639g	48	PHA	empile LL de la longueur du fichier
C63Ag	AD 07 C1	LDA C107	HH de l'ancienne adresse de fin
C63Dg	ED 05 C1	SBC C105	HH de l'ancienne adresse de début
C640g	AA	TAX	sauve dans X, HH de la longueur du fichier
C641g	68	PLA	récupère dans A, LL de la longueur du fichier
C642g	18	CLC	prépare l'addition nouvelle_adresse_de_début + longueur
C643g	65 F6	ADC F6	A + F6 = LL de la nouvelle adresse de fin
C645g	8D 06 C1	STA C106	sauve LL de la nouvelle adresse de fin
C648g	8A	TXA	récupère dans A, HH de la longueur du fichier
C649g	65 F7	ADC F7	A + F7 = HH de la nouvelle adresse de fin
C64Bg	8D 07 C1	STA C107	sauve HH de la nouvelle adresse de fin
C64Eg	68	PLA	A = LL de la nouvelle adresse de chargement
C64Fg	8D 04 C1	STA C104	sauve AY nouvelle adresse de chargement
C652g	8C 05 C1	STY C105	c'est à dire nouvelle adresse de début
C655g	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
C658g	F0 44	BEQ C69E	si fin d'instruction, fini, sauve le secteur descripteur
C65Ag	20 2C D2	JSR D22C	exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C65Dg	C9 54	CMP #54	est-ce "T"?
C65Fg	D0 12	BNE C673	sinon, poursuit l'analyse en C673; si oui...
C661g	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C664g	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)
C667g	A6 33	LDX 33	XY = nouvelle adresse d'exécution
C669g	A4 34	LDY 34	(sera sauvée dans tous les cas, même BASIC!)
C66Bg	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
C66Eg	F0 17	BEQ C687	si fin d'instruction, OK, continue en C687
C670g	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")
C673g	C9 C7	CMP #C7	est-ce le token "AUTO"?
C675g	D0 F9	BNE C670	sinon, "SYNTAX_ERROR" (tous les paramètres possibles ont été examinés; s'il reste quelque chose, c'est une erreur); si oui...
C677g	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C67Ag	D0 F4	BNE C670	si pas fin d'instruction, "SYNTAX_ERROR"
C67Cg	2C 03 C1	BIT C103	teste b7 de flag "type de fichier" (1 = BASIC)
C67Fg	30 06	BMI C687	si fichier BASIC, continue en C687 (on ferme les yeux sur la valeur courante de XY, mais on l'écrira quand même!)
C681g	AE 04 C1	LDX C104	si c'est un fichier autre que BASIC,
C684g	AC 05 C1	LDY C105	XY = adresse de début
C687g	8E 08 C1	STX C108	sauve nouvelle adresse d'exécution
C68Ag	8C 09 C1	STY C109	(une bogue si c'est un programme BASIC!)
C68Dg	A9 01	LDA #01	A = #01 (pour mettre en "AUTO")
C68Fg	2C A9 00	BIT 00A9	et continue en C692
C690g	A9 00	LDA #00	A = #00 (flag pas de paramètre = "non AUTO")
C692g	85 F5	STA F5	sauve temporairement le flag "AUTO/non AUTO"
C694g	AD 03 C1	LDA C103	indication du type de fichier
C697g	29 FE	AND #FE	1111 1110 mise à zéro du b0 (AUTO/non AUTO)
C699g	05 F5	ORA F5	transfert du nouveau flag "AUTO/non AUTO"
C69Bg	8D 03 C1	STA C103	et sauvegarde (bogue si fichier non exécutable)
C69Eg	4C A4 DA	JMP DAA4	XSVSEC re-écrit le secteur descripteur modifié selon DRIVE, PISTE, SECTEUR et RWBUF

EXÉCUTION DE LA COMMANDE SEDORIC PROT

Cette commande, qui était située à l'origine dans le NOYAU de E6D0 à E701, se trouve maintenant dans la BANQUE n°7 de C6A1 à C6D2.

Rappel de syntaxe

PROT nom_de_fichier_ambigu

Protège le ou les fichiers spécifiés contre DEL, DESTROY, SAVEO, STATUS etc...

C6A1g	A9 80	LDA #80	prépare un b7 à 1 pour PROT
C6A3g	2C A9 00	BIT 00A9	et continue en C6A6

EXÉCUTION DE LA COMMANDE SEDORIC UNPROT

Cette commande, qui était située à l'origine dans le NOYAU de E6D3 à E701, se trouve maintenant dans la BANQUE n°7 de C6A4 à C6D2.

Rappel de syntaxe

UNPROT nom_de_fichier_ambigu

Annule la protection obtenue avec la commande PROT ci-dessus.

C6A4g	A9 00	LDA #00	prépare un b7 à 0 pour UNPROT
C6A6g	85 F9	STA F9	sauve le flag PROT/UNPROT
C6A8g	20 51 D4	JSR D451	lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
C6ABg	D0 2F	BNE C6DC	"SYNTAX_ERROR" s'il y a des paramètres (fin d'instruction requise)
C6ADg	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si fichier n'a pas été trouvé
C6B0g	F0 1E	BEQ C6D0	"FILE_NOT_FOUND_ERROR"
C6B2g	20 A0 D7	JSR D7A0	cherche "?" dans BUFNOM (C = 1 si pas trouvé)
C6B5g	90 10	BCC C6C7	si "?" trouvé, continue en C6C7
C6B7g	AE 27 C0	LDX C027	POSNMX premier octet de “l’entrée” dans le secteur de catalogue
C6BAg	BD 0F C3	LDA C30F,X	lit dernier octet de “l’entrée”
C6BDg	29 7F	AND #7F	0111 1111 mise à zéro du b7 (flag PROT/UNPROT)
C6BFg	05 F9	ORA F9	force b7 avec nouvelle valeur flag PROT/UNPROT
C6C1g	9D 0F C3	STA C30F,X	et remet en place
C6C4g	4C 82 DA	JMP DA82	XSCAT sauve le secteur de catalogue selon POSNMP et POSNMS
C6C7g	20 B7 C6	JSR C6B7	mise à jour flag PROT/UNPROT "entrée" courante
C6CAg	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)
C6CDg	D0 F8	BNE C6C7	reboucle tant qu'il en reste à mettre à jour
C6CFg	60	RTS	
C6D0g	4C DD E0	JMP E0DD	"FILE_NOT_FOUND_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC SYSTEM

Cette commande, qui était située à l'origine dans le NOYAU de E702 à E70A, se trouve maintenant dans la BANQUE n°7 de C6D3 à C6DE.

Rappel de la syntaxe

SYSTEM lecteur

Définit le lecteur où SEDORIC devra lire les blocs externes (BANQUES interchangeables) pour exécuter certaines commandes (INIT, COPY etc...).

```
C6D3g  20 0D E6   JSR E60D     valide drive si indiqué, sinon valide DRVDEF
C6D6g  D0 04      BNE C6DC     "SYNTAX_ERROR" s'il y a des paramètres (fin d'instruction requise)
C6D8g  8C 0A C0   STY C00A     sauve drive indiqué ou DRVDEF dans DRVSYS
C6DBg  60         RTS

C6DCg  4C 23 DE   JMP DE23     SYNTAX_ERROR
```

EXÉCUTION DE LA COMMANDE SEDORIC VISUHIRES

Il s'agit d'une nouvelle commande située dans la BANQUE n°7 de C6DF à C7FF.

Syntaxe

VISUHIRES nom_de_fichier_ambigu(,AUTO)

Visionneuse HIRES: cette commande affiche le ou les fichiers HIRES indiqués.

Exemples: VISUHIRES"TOTO.HRS"
VISUHIRES"IMAGE?"
VISUHIRES"*.HRS",AUTO

Le paramètre ",AUTO" doit être tapé en MAJUSCULES. S'il est omis, il faudra appuyer sur une touche pour passer au fichier suivant. S'il est présent, l'affichage sera automatique. Toutefois, il peut être interrompu par pression sur la touche <ESPACE> et repris par une pression sur n'importe quelle touche sauf <ESPACE> et <ESC>. La touche <ESC> permet d'abandonner dans tous les cas.

Attention, si vous tentez d'afficher avec cette commande, des fichiers autres que des écrans HIRES, vous risquez le pire. La commande est sécurisée, mais de manière rudimentaire: pour être affiché un fichier doit être du type "Bloc de mémoire", avoir une adresse de début supérieure à A000 et une adresse de fin inférieure à C000. Attention notamment aux fichiers AUTO comportant non seulement une écran HIRES, mais aussi un lanceur, système qui fut utilisé pour charger le fichier suivant. Pour se débarrasser de ce lanceur, passez en HIRES, chargez le fichier avec l'option ",N" (pour bloquer le lancement AUTO) et resauvez le avec la commande ESAVE.

Passe en mode HIRES

```
C6DFg  20 D8 D5   JSR D5D8     exécute une routine de la ROM: passage en mode HIRES
```

C6E2g	BB E9		située en E9BB dans la ROM1.0
C6E4g	33 EC		située en EC33 dans la ROM1.1

Analyse de syntaxe

C6E6g	78	SEI	interdit les interruptions
C6E7g	20 51 D4	JSR D451	lit nom_de_fichier_ambigu à TXTPTR
C6EAg	A0 00	LDY #00	"non AUTO" par défaut
C6ECg	84 00	STY 00	flag "AUTO/non AUTO"
C6EEg	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
C6F1g	F0 0E	BEQ C701	by-passe si aucun paramètre
C6F3g	20 2C D2	JSR D22C	demande une "," et lit le caractère suivant à TXTPTR
C6F6g	C9 C7	CMP #C7	est-ce le token "AUTO" ?
C6F8g	D0 E2	BNE C6DC	non, SYNTAX_ERROR (rien d'autre autorisé)
C6FAg	85 00	STA 00	oui, flag "AUTO/non AUTO" = token "AUTO" = #C7
C6FCg	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
C6FFg	D0 DB	BNE C6DC	SYNTAX_ERROR si pas fin d'instruction

Cherche le ou les fichiers

C701g	20 30 DB	JSR DB30	charge le catalogue et met à jour POSNMX
C704g	F0 22	BEQ C728	rien trouvé = terminé
C706g	BD 0C C3	LDA C30C,X	coordonnées du
C709g	BC 0D C3	LDY C30D,X	descripteur approprié
C70Cg	20 5D DA	JSR DA5D	charge ce descripteur
C70Fg	A2 02	LDX #02	boucle
C711g	BD 00 C1	LDA C100,X	de
C714g	C9 FF	CMP #FF	recherche
C716g	F0 19	BEQ C731	du
C718g	E8	INX	flag
C719g	D0 F6	BNE C711	#FF
C71Bg	AD 00 C1	LDA C100	dans
C71Eg	AC 01 C1	LDY C101	le
C721g	D0 E9	BNE C70C	descripteur
C723g	20 41 DB	JSR DB41	cherche l'entrée suivante
C726g	D0 DE	BNE C706	trouvée, reboucle

On retourne au mode TEXT et on termine

C728g	58	CLI	rien trouvé, quitte
C729g	20 D8 D5	JSR D5D8	exécute une routine de la ROM, ici la commande TEXT
C72Cg	A9 E9		située en E9A9 (ROM1.0)

```
C72Eg  21 EC                ou en EC21 (ROM1.0)
C730g  60        RTS        sortie de la commande VISUHIRES
```

Examine si le fichier trouvé est correct

```
C731g  BD 01 C1  LDA C101,X FTYPE
C734g  29 40     AND #40    rejette les fichiers
C736g  F0 EB     BEQ C723   non "bloc mémoire", recherche le fichier suivant
C738g  BD 03 C1  LDA C103,X HH de l'adresse de début
C73Bg  85 F9     STA F9     gardé dans F9
C73Dg  C9 A0     CMP #A0    rejette les fichiers
C73Fg  90 E2     BCC C723   commençant avant A000, recherche le fichier suivant
C741g  BD 05 C1  LDA C105,X HH de l'adresse de fin
C744g  48        PHA        gardé sur la pile
C745g  C9 C0     CMP #C0    rejette les fichiers
C747g  B0 DA     BCS C723   finissant après BFFF, recherche le fichier suivant
```

Affichage du fichier HIRES

```
C749g  8A        TXA
C74Ag  48        PHA
C74Bg  20 B4 DA  JSR DAB4   affiche le nom du fichier retenu
C74Eg  68        PLA
C74Fg  AA        TAX
C750g  20 06 D2  JSR D206   affiche un CRLF
C753g  38        SEC
C754g  BD 04 C1  LDA C104,X
C757g  FD 02 C1  SBC C102,X calcule la longueur du fichier, soit:
C75Ag  8D 4F C0  STA C04F   adresse de fin - adresse de début
C75Dg  68        PLA
C75Eg  E5 F9     SBC F9
C760g  8D 50 C0  STA C050   place le résultat en C04F/50
C763g  BD 02 C1  LDA C102,X
C766g  8D 03 C0  STA C003   met à jour RWBUF (adresse de chargement)
C769g  A4 F9     LDY F9
C76Bg  88        DEY        avec adresse de début -#100
C76Cg  8C 04 C0  STY C004   pour compenser mise à jour de début de boucle
C76Fg  BD 08 C1  LDA C108,X
C772g  85 F7     STA F7     nombre de secteurs à charger
C774g  BD 09 C1  LDA C109,X
C777g  85 F8     STA F8
C779g  8A        TXA
C77Ag  18        CLC
C77Bg  69 06     ADC #06
C77Dg  A8        TAY
C77Eg  20 28 E2  JSR E228   met à jour l'index Y
C781g  A5 F7     LDA F7     charge les secteurs complets
C783g  D0 02     BNE C787
```

```
C785g  C6 F8       DEC F8      décrémente le nombre de secteurs à charger
C787g  C6 F7       DEC F7      = nombre de secteurs complets
C789g  EE 04 C0    INC C004    met à jour RWBUF
C78Cg  A5 F7       LDA F7
C78Eg  05 F8       ORA F8      reste-il des secteurs complets à charger ?
C790g  F0 08       BEQ C79A    non, by-passe
C792g  20 28 E2    JSR E228    oui, met à jour l'index Y
C795g  20 50 E2    JSR E250    lit les coordonnées du secteur et le charge
C798g  F0 E7       BEQ C781    rebouclage forcé
C79Ag  58          CLI         autorise les interruptions pour test de touche
C79Bg  AD 03 C0    LDA C003
C79Eg  AE 04 C0    LDX C004    sauve l'adresse de chargement de l'écran HIRES dans F5/F6
C7A1g  85 F5       STA F5
C7A3g  86 F6       STX F6
C7A5g  20 28 E2    JSR E228    met à jour l'index Y
C7A8g  98          TYA         et l'empile
C7A9g  48          PHA
C7AAg  A9 00       LDA #00
C7ACg  A2 C2       LDX #C2
C7AEg  8D 03 C0    STA C003    ajuste RWBUF pour effectuer un chargement intermédiaire
C7B1g  8E 04 C0    STX C004    dans BUF2 (C200)
C7B4g  20 50 E2    JSR E250    lit les coordonnées du secteur et le charge dans BUF2
C7B7g  A0 FF       LDY #FF     met Y à jour pour entrée de boucle
C7B9g  C8          INY
C7BAg  B9 00 C2    LDA C200,Y  recopie les octets significatifs (LL de la longueur)
C7BDg  91 F5       STA (F5),Y  de BUF2 vers l'écran HIRES
C7BFg  CC 4F C0    CPY C04F
C7C2g  D0 F5       BNE C7B9
```

Affichage automatique demandé?

```
C7C4g  A4 00       LDY 00      teste le flag "AUTO/non AUTO"
C7C6g  F0 2D       BEQ C7F5    continue en C7F5 si "non AUTO"
```

Affichage automatique

```
C7C8g  A0 FF       LDY #FF     pour temporisateur
C7CAg  88          DEY         255 tests de touche seront effectués
C7CBg  F0 14       BEQ C7E1    timer out: fichier suivant
C7CDg  20 02 D3    JSR D302    touche ?
C7D0g  C9 1B       CMP #1B     est-ce un ESC ?
C7D2g  F0 1D       BEQ C7F1    oui, abandonne
C7D4g  C9 20       CMP #20     est-ce un espace ?
C7D6g  D0 F2       BNE C7CA    non, reboucle pour un nouveau test de touche
C7D8g  20 02 D3    JSR D302    oui, re-test de touche
C7DBg  10 FB       BPL C7D8    stand-by tant que pas de touche
C7DDg  C9 20       CMP #20     touche détectée, est-ce un espace ?
C7DFg  F0 F7       BEQ C7D8    oui, retourne en stand-by (dispositif de sécurité)
```

Fichier suivant

C7E1g	78	SEI	interdit les interruptions et passe au fichier suivant
C7E2g	68	PLA	
C7E3g	A8	TAY	
C7E4g	20 28 E2	JSR E228	met à jour l'index Y
C7E7g	B0 05	BCS C7EE	pas de descripteur suivant, cherche entrée suivante
C7E9g	98	TYA	il y a encore un descripteur
C7EAg	AA	TAX	en cherche le début
C7EBg	4C 11 C7	JMP C711	rebouclage forcé pour chercher le #FF du fichier “mergé” suivant
C7EEg	4C 23 C7	JMP C723	rebouclage forcé pour chercher l'écran HIRES suivant

Abandon

C7F1g	68	PLA	abandon
C7F2g	4C 29 C7	JMP C729	rebouclage forcé vers la sortie

Affichage non automatique

C7F5g	20 02 D3	JSR D302	touche?
C7F8g	10 FB	BPL C7F5	stand-by en attente de touche
C7FAg	C9 1B	CMP #1B	touche détectée, est-ce un ESC ?
C7FCg	F0 F3	BEQ C7F1	oui, abandon
C7FEg	D0 E1	BNE C7E1	non, passe au fichier suivant

DÉBUT DU NOYAU PERMANENT
C'est la zone de RAM overlay située de C800 à FFFF

Le début du NOYAU permanent de SEDORIC est occupé par les tables KEYDEF, REDEF et PREDEF. Cette zone, qui se trouve de C800 à C9DD a été complètement remaniée (362 octets différents, indiqués en gras dans les dumps qui suivent) dans la version 3.0 de SEDORIC.

TABLE “KEYDEF”

Elle est responsable de l'affectation de codes de fonctions (#00 à #FF) aux codes de touches (#80 à #BF). Les codes de fonctions sont décrits en ANNEXE. Les codes de touches sont représentés, sur l'image d'un clavier, dans le manuel SEDORIC, page 104. L'ordre de ces codes de touche est des plus mystérieux, puisque le premier (#80) est attribué à la touche "7", le second (#81) à la touche "J", le troisième (#82) à la touche "M", etc jusqu'au dernier (#BF) à la touche "=".

Voici un tableau qui résume la correspondance entre les touches (dump du centre), les codes de touches correspondants (à gauche, valeurs de #80 à #BF) et les index de lecture dans la table **KEYDEF située en C800** pour un appui sur une touche + FUNCT ou sur une touche + FUNCT + SHIFT. Ces index de lecture sont la valeur à ajouter (de #00 à #7F) à l'adresse du début de la table #C800 pour trouver le code de function associé à cette combinaison de touches. Autrement dit, à chaque combinaison FUNCT+touche ou FUNCT+SHIFT+touche correspond un code de fonction choisit parmi 256. Cette correspondance est donnée par la table "KEYDEF", qui liste les #80 codes correspondant aux #40 combinaisons FUNCT+touche et aux #40 combinaisons FUNCT+SHIFT+touche.

TABLE DES CODES DE TOUCHES

Codes pour touche seule	0 1 2 3 4 5 6 7 8 9 A B C D E F	Index de lecture dans la table KEYDEF pour: touche + FUNCT	touche + FUNCT+SHIFT
#80 à #8F	7 J M K _ U Y 8 N T 6 9 , I H L	#00 à #0F	#40 à #4F
#90 à #9F	5 R B ; . O G 0 V F 4 - ↑ P E /	#10 à #1F	#50 à #5F
#A0 à #AF	m m c m g f m s 1 e Z m ← d A r	#20 à #2F	#60 à #6F
#B0 à #BF	X Q 2 \ ↓] S m 3 D C ' → [W =	#30 à #3F	#70 à #7F

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT gauche, m touche manquante, r RETURN, s SHIFT droit, et _ SPACE. Exemple: le code de touche #80 correspond à la touche "7" et le code de touche #BF à la touche "=". A chaque code de touche correspond un index de lecture dans la table KEYDEF. Mais certains codes (exemple #A0) ne correspondent à aucune touche (“m” pour touche manquante).

La table "KEYDEF" ci-après est une suite de 128 (#80) codes de fonctions, choisis parmi les 256 possibles et rangés dans le désordre de C800 à C87F. En fait ils sont rangés dans l'ordre des codes de touches:

d'abord "7", puis "J", puis "M", ... jusqu'à "=". L'index d'entrée dans cette table va de #00 à #3F pour les combinaisons FUNCT+touche et de #40 à #7F pour les combinaisons FUNCT+SHIFT+touche. Il faut ajouter respectivement #80 ou #40 à cet index pour obtenir le code de fonction correspondant.

La table KEYDEF a été complètement revue pour intégrer des commandes SEDORIC dont l'utilisation était impossibles dans les versions précédentes. Ceci a été rendu possible grâce à la correction de la bogue de la routine "Prendre un caractère au clavier" en D907.

TABLE KEYDEF PROPREMENT DITE

FUNCT+touche (commandes SEDORIC)

	Codes de fonctions (voir en ANNEXE)								Touches (de #80 à #BF)							
C800-	**07**	**45**	**57**	**4B**		**00**	**18**	**07**	**08**	7	J	M	K	_	U	Y	8
C808-	**59**	**7B**	**06**	**09**		00	**42**	**41**	**52**	N	T	6	9	,	I	H	L
C810-	05	**66**	**25**	00		00	**5B**	**27**	**00**	5	R	B	;	.	O	G	0
C818-	**1B**	**3F**	**04**	**0A**		**00**	**5E**	**3D**	**0D**	V	F	4	-	↑	P	E	/
C820-	00	00	00	00		00	00	00	00	m	m	c	m	g	f	m	s
C828-	**01**	**00**	**08**	00		**00**	**00**	**22**	FF	1	e	Z	m	←	d	A	r
C830-	**6D**	**62**	**02**	**0C**		**00**	**0F**	**72**	00	X	Q	2	\	↓	]	S	m
C838-	**03**	**31**	**29**	**00**		**00**	**0E**	**1E**	**0B**	3	D	C	'	→	[	W	=

FUNCT+SHIFT+touche (commandes BASIC)

	Codes de fonctions (voir en ANNEXE)								Touches (de #80 à #BF)							
C840-	**17**	**B2**	A8	**F1**		00	**8C**	**A6**	**18**	7	J	M	K	_	U	Y	8
C848-	**90**	**C9**	**16**	**19**		00	92	**A2**	**BC**	N	T	6	9	,	I	H	L
C850-	**15**	9C	**CA**	00		00	D2	9B	**10**	5	R	B	;	.	O	G	0
C858-	**EB**	**8D**	**14**	**1A**		**00**	**87**	**C8**	**1D**	V	F	4	-	↑	P	E	/
C860-	00	00	00	00		00	00	00	00	m	m	c	m	g	f	m	s
C868-	**11**	00	**A5**	00		00	00	**D1**	FF	1	e	Z	m	←	d	A	r
C870-	**A4**	**9A**	**12**	**1C**		00	**1F**	**CB**	00	X	Q	2	\	↓	]	S	m
C878-	**13**	**91**	ED	00		**00**	**1E**	**B5**	**1B**	3	D	C	'	→	[	W	=

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT gauche, m touche manquante (certains numéros de code ne correspondent à aucune touche), r RETURN, s SHIFT droit, et _ SPACE.

Ce nouveau tableau permet d'accéder aux fonctions **BASIC** avec **FUNCT+SHIFT+touche** et aux commandes **SEDORIC** avec **FUNCT+touche**. Et ceci en respectant autant que possible les initiales. Les commandes SEDORIC sans n° (UNPROT, USING, VISUHIRES, VUSER, WIDTH, WINDOW et !RESTORE) sont maintenant accessibles.

UNE TABLE KEYDEF UN PEU PLUS PRATIQUE:

J'ai reclassé ce tableau de correspondance dans l'autre sens: A chaque touche (dans l'ordre alphabétique, de A à Z) correspond une commande SEDORIC (avec appui simultané sur FUNCT) ou une commande BASIC (avec appui simultané sur FUNCT+SHIFT). Les codes de fonction sont aussi indiqués.

Touche	FUNCT (SEDORIC)	Code n°	FUNCT+SHIFT (BASIC)	Token n°
A/Q	AZERTY	#22	AND	#D1
B	BACKUP	#25	NOT	#CA
C	COPY	#29	CHR$	#ED
D	**DIR**	**#31**	DATA	#91
E	ESAVE	#3D	ELSE	#C8
F	FIELD	#3F	FOR	#8D
G	CHANGE	#27	GOSUB	#9B
H	HCUR	#41	HIRES	#A2
I	INIT	#42	INPUT	#92
J	JUMP	#45	INK	#B2
K	KEYSAVE	#4B	KEY$	#F1
L	LINPUT	#52	**LIST**	**#BC**
M/?	MOVE	#57	MUSIC	#A8
N	NUM	#59	NEXT	#90
O	OLD	#5B	OR	#D2
P	PROT	#5E	PLOT	#87
Q/A	QWERTY	#62	RESTORE	#9A
R	RENUM	#66	RETURN	#9C
S	SAVEU	#72	STEP	#CB
T	TYPE	#7B	THEN	#C9
U	UNPROT	#18	UNTIL	#8C
V	VISUHIRES	#1B	VAL	#EB
W/Z	WINDOW	#1E	WAIT	#B5
X	SEEK	#6D	EXPLODE	#A4
Y	PAPER0:INK7	#07	PING	#A6
Z/W	CALL#F8D0+CR	#08	ZAP	#A5

Exemples: FUNCT+"D" affiche "DIR" (Code n°49 = #31, voir en ANNEXE), tandis que FUNCT+SHIFT+"L" affiche "LIST" (Token n° 188 = #BC, voir en ANNEXE). Les touches A/Q, M/?, Q/A, W/Z et Z/W ont une double étiquette. Ceci correspond aux claviers AZERTY/QWERTY. La touche ;/M n'est pas utilisée, il en est de même pour les touches ' , . et /. qui toutes ont reçu le code #00. Il est possible de re-definir ces touches à l'aide de la commande KEYDEF.

TABLE "REDEF"

16 commandes re-definissables avec KEYUSE (codes de fonction de 0 à 15)

De C880 à C97F, la table REDEF des fonctions re-definissables a été complètement modifiée (les octets modifiés sont indiqués en gras dans le dump hexadécimal). Chaque chaîne se termine par un caractère +#80 (indiqué en gras à la fin de chaque chaîne alphanumérique):

```
C880-  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 A0     espace
C890-  20 20 20 20 20 20 44 4F 4B 45 23 32 46 35 2C A3     DOKE#2F5,#
C8A0-  20 20 44 4F 4B 45 23 32 46 35 2C 23 34 36 37 8D     DOKE#2F5,#467+CR
C8B0-  20 20 20 20 20 20 44 4F 4B 45 23 32 46 39 2C A3     DOKE#2F9,#
C8C0-  20 44 4F 4B 45 23 32 46 39 2C 23 44 30 37 30 8D     DOKE#2F9,#D070+CR
C8D0-  20 20 20 20 20 20 44 4F 4B 45 23 32 46 43 2C A3     DOKE#2FC,#
C8E0-  20 20 44 4F 4B 45 23 32 46 43 2C 23 34 36 31 8D     DOKE#2FC,#461+CR
C8F0-  20 20 20 20 50 41 50 45 52 30 3A 49 4E 4B 37 8D     PAPER0:INK7+CR
C900-  20 20 20 20 20 20 43 41 4C 4C 23 46 38 44 30 8D     CALL#F8D0+CR
C910-  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 FE     ê
C920-  20 20 20 20 3F 48 45 58 24 28 50 45 45 4B 28 A3     ?HEX$(PEEK(#
C930-  20 20 20 20 3F 48 45 58 24 28 44 45 45 4B 28 A3     ?HEX$(DEEK(#
C940-  20 20 20 20 20 20 20 20 20 20 50 45 45 4B 28 A3     PEEK(#
C950-  20 20 20 20 20 20 20 20 20 20 44 45 45 4B 28 A3     DEEK(#
C960-  20 20 20 20 20 20 20 20 20 20 20 50 4F 4B 45 A3     POKE#
C970   20 20 20 20 20 20 20 20 20 20 20 44 4F 4B 45 A3     DOKE#
```

TABLE "PREDEF"

16 commandes pré-definies (codes de fonction de 16 à 31)

De C980 à C9DD, la table PREDEF des fonctions pré-definies a également été complètement modifiée (chaque chaîne se termine par un caractère +#80, indiqué en gras):

```
C980   48 45 58 24 A8                                     HEX$(
C985   43 41 4C 4C A3                                     CALL#
C98A   54 45 58 54 8D                                     TEXT+CR
C98F   46 4F 52 49 3D 31 54 CF                            FORI=1TO
C997   4C 45 46 54 24 A8                                  LEFT$(
C99D   4D 49 44 24 A8                                     MID$(
C9A2   52 49 47 48 54 24 A8                               RIGHT$(
C9A9   53 54 52 24 A8                                     STR$(
C9AE   55 4E 50 52 4F 54 8D                               UNPROT+CR
C9B5   E0                                                 ©
C9B6   55 53 49 4E C7                                     USING
C9BB   56 49 53 55 48 49 52 45 53 A2                      VISUHIRES"
C9C5   56 55 53 45 52 8D                                  VUSER+CR
C9CB   57 49 44 54 C8                                     WIDTH
C9D0   57 49 4E 44 4F D7                                  WINDOW
C9D6   21 52 45 53 54 4F 52 C5                            !RESTORE
```

DES TABLES “REDEF” ET “PREDEF” UN PEU PLUS PRATIQUES

Les nouvelles fonctions peuvent être obtenues avec les combinaisons de touches suivantes:

Touche	FUNCT (Cdes re-definissables)	Cde n°	Touche	FUNCT+SHIFT (Cdes pré-definies)	Cde n°
0	espace (=rien)	#00	0	HEX$(	#10
1	DOKE#2F5,#	#01	1	CALL#	#11
2	DOKE#2F5,#467+CR	#02	2	TEXT	#12
3	DOKE#2F9,#	#03	3	FORI=1TO	#13
4	DOKE#2F9,#D070+CR	#04	4	LEFT$(	#14
5	DOKE#2FC,#	#05	5	MID$(	#15
6	DOKE#2FC,#461+CR	#06	6	RIGHT$(	#16
7	PAPER0:INK7+CR	#07	7	STR$(	#17
8	**CALL#F8D0+CR**	**#08**	8	UNPROT	#18
9	ê (ASCII n°126 = #7E)	#09	9	© (ASCII 96 = #60)	#19
- £	?HEX$(PEEK(#	#0A	- £	USING	#1A
= +	?HEX$(DEEK(#	#0B	= +	**VISUHIRES"**	**#1B**
\ |	PEEK(#	#0C	\ |	VUSER	#1C
/ ?	DEEK(#	#0D	/ ?	WIDTH	#1D
[{	POKE#	#0E	[{	WINDOW	#1E
]}	DOKE#	#0F	]}	!RESTORE	#1F

Exemple: FUNCT+"8" déclenche une régénération des caractères (c'est une commande utilisateur re-definissable avec KUSE, visualisable avec VUSER, manuel SEDORIC page 55 & 102), tandis que FUNCT+SHIFT+"=" affiche VISUHIRES" qu'il faut compléter pour déclencher l'affichage des écrans HIRES que l'on aura indiqués (nouvelle commande pré-definie n°27 = #1B).

NB: DOKE#2F5, #2F9 et #2FC sont les vecteurs de !,] et &(). Les touches ESC, CTRL, SHIFTg, ←, ↓, espace, ↑, →, FUNCT, SHIFTd, RETURN et DEL ainsi que les touches restantes (; ' , . /) reçoivent le code de re-definition #00 soit rien. FUNCT+RETURN affiche le numéro de la ligne BASIC suivante (commande NUM).

SEDORIC3D.KEY et TABLE “REDEF” POUR DÉVELOPPEURS

Les tables KEYDEF, PREDEF et REDEF telles qu'elles sont décrites ci-dessus sont présentes non seulement dans le NOYAU, mais aussi dans le fichier SEDORIC3N.KEY (N pour Normale). Le fichier SEDORIC3D.KEY (D pour Développeurs) contient également les mêmes tables à l'exception de la table REDEF qui a été changée pour avoir:

Touche	FUNCT	Cde n°	
0	espace (=rien)	#00	pour les touches pas encore attribuées par KEYDEF
1	POKE#26A,(PEEK(#	#01	suivie de l'une des 2 commandes suivantes
2	26A)AND#FE)	#02	pour forcer le curseur à OFF (invisible)
3	26A)OR#01)	#03	pour forcer le curseur à ON (visible)
4	PRINTCHR$(18);	#04	pour valider la commande curseur ON/OFF
5	POKE#BBA3,#0	#05	pour effacer le hideux CAPS de la ligne service
6	FORI=#BB80TO#BBA	#06	suivie de la commande suivante
7	7:POKEI,32:NEXTI	#07	pour effacer toute la ligne service
8	POKE#BB80,	#08	suivie de la commande suivante
9	PEEK(#26B)	#09	pour couleur PAPER ligne service = PAPER écran
- £	POKE#BB81,	#0A	suivie de la commande suivante
= +	PEEK(#26C)	#0B	pour couleur INK ligne service = INK écran
\ |	POKE#20C,#FF	#0C	pour forcer en mode MAJUSCULE
/ ?	POKE#20C,#7F	#0D	pour forcer en mode minuscule
[{	?HEX$(PEEK(#	#0E	pour afficher le contenu hexadécimal d'un octet
]}	?HEX$(DEEK(#	#0F	pour afficher le contenu hexadécimal de 2 octets

Exemple: Vous êtes en train de taper un programme BASIC et vous voulez effacer le curseur. Au lieu de l'habituelle bascule PRINT CHR$(17), vous voulez taper POKE#26A,(PEEK(#26A)AND#FE) qui force à OFF indépendamment de l'état précédent. Pour cela, il suffit de taper FUNCT+1 puis FUNCT+2. Si nécessaire il faut ajouter un PRINTCHR$(18); pour valider la commande précédente: Tapez simplement FUNCT+4. Rappel: le tableau ci-dessus est à photocopier et à placer dans votre Manuel ou près de votre ORIC.

Pour les utilisateurs désireux de ne rien changer à leurs habitudes, le fichier SEDORIC1.KEY (1 pour version 1) contient les tables KEYDEF, PREDEF et REDEF correspondant au clavier de la version 1.006.

TABLE MOTS-CLÉS SEDORIC (codes 32 à 127)

Les commandes DELETE et USING, ont été supprimées dans leur version "en minuscules" et remplacées par CHKSUM et VISUHIRES (en MAJUSCULES seulement). En fait dans la nouvelle table, DELETE "en MAJUSCULES" est remplacé par CHKSUM, DELETE "en minuscule" est supprimé et remplacé par DELETE "en MAJUSCULES", USING est remplacé par UNPROT et UNPROT est remplacé par VISUHIRES. Les 17 octets modifiés figurent en gras dans le dump hexadécimal.

L'initiale de chaque commande (indiquée en gras souligné) est implicite. Le code de fonction de chaque commande a été indiqué, ainsi que l'adresse d'exécution (voir plus loin la table des adresses d'exécution).

Les mots-clés qui comportent un token BASIC sont codés de deux manières différentes. Lorsque les commandes SEDORIC sont tapées en MAJUSCULES, les mots-clés du BASIC sont reconnus et les lettres correspondantes sont remplacées par le token BASIC, l'encodage est réalisé par la ROM en ECB9. Au contraire, si l'utilisateur tape les mots de SEDORIC en minuscules, chaque lettre du mot sera 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 RAM overlay, et donc l'exécution, en sont accélérées. De plus, le nombre de bogues affectant les commandes tapées en minuscules était si élevé dans la version 1.0 que cette possibilité n'est plus prise en charge dans la version 3.0 de SEDORIC.

Dump hexadécimal	Commande	Code n°	Adresse	Remarque
C9DE- 50 50 80 00	**A**PPEND	#20=032	FE07	;#80 token BASIC "END"
C9E2- 50 50 45 4E 44 00	**A**PPEND	#21=033	FE07	
C9E8- 5A 45 52 54 59 00	**A**ZERTY	#22=034	EBDE	
C9EE- 43 43 45 4E 54 00	**A**CCENT	#23=035	EB91	
C9F4- 4F 58 00	**B**OX	#24=036	F0DE	
C9F7- 41 43 4B 55 50 00	**B**ACKUP	#25=037	F151	
C9FD- 55 49 4C 44 00	**B**UILD	#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-  48 4B 53 55 4D 00       CHKSUM      #2C=044   E9FF

CA24-  45 96 45 00             DELETE      #2C=045   F142 ;#96 token BASIC "LET"
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 token 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 token 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 token 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 token 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 token BASIC "SIN"
CACC-  92 00                  LINPUT      #51=081   EC94 ;#92 token 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

CAEE-  55 4D 00               NUM         #59=089   EB25

CAF1-  55 54 00               OUT         #5A=090   E71F
CAF4-  4C 44 00               OLD         #5B=091   E0AF
CAF7-  50 45 4E 00            OPEN        #5C=092   FA50

CAFB-  55 54 00               PUT         #5D=093   F9CB
CAFE-  52 4F 54 00            PROT        #5E=094   E9F6
CB02-  52 00                  PR          #5F=095   E7C0
CB04-  4D 41 50 00            PMAP        #60=096   F990

CB08-  55 49 54 00            QUIT        #61=097   E7F5
CB0C-  57 45 52 54 59 00      QWERTY      #62=098   EBE1
```

```
CB12-  45 53 55 4D 45 00      RESUME      #63=099   E9BB
CB18-  53 45 54 00            RSET        #64=100   FC75
CB1C-  45 57 49 4E 44 00      REWIND      #65=101   FABB
CB22-  45 4E 55 4D 00         RENUM       #66=102   F14E
CB27-  45 4E 00               REN         #67=103   E537
CB2A-  D1 4F 4D 00            RANDOM      #68=104   E796 ;#D1 token BASIC "AND"
CB2E-  41 4E 44 4F 4D 00      RANDOM      #69=105   E796
CB34-  45 53 54 4F 52 45 00   RESTORE     #6A=106   E7D9
CB3B-  45 53 45 54 00         RESET       #6B=107   E7B8

CB40-  57 41 50 00            SWAP        #6C=108   EA3B
CB44-  45 45 4B 00            SEEK        #6D=109   F154
CB48-  54 52 55 4E 00         STRUN       #6E=110   E853
CB4D-  54 98 00               STRUN       #6F=111   E853 ;#98 token BASIC "RUN"
CB50-  59 53 54 45 4D 00      SYSTEM      #70=112   E9FC
CB56-  54 41 54 55 53 00      STATUS      #71=113   E9F3
CB5C-  41 56 45 55 00         SAVEU       #72=114   DD4D
CB61-  41 56 45 4D 00         SAVEM       #73=115   DD4A
CB66-  41 56 45 4F 00         SAVEO       #74=116   DD53
CB6B-  41 56 45 00            SAVE        #75=117   DD50
CB6F-  45 41 52 43 48 00      SEARCH      #76=118   E5FC
CB75-  59 53 00               SYS         #77=119   F15A
CB78-  4D 41 50 00            SMAP        #78=120   F996

CB7C-  4B 45 4E 00            TKEN        #79=121   E89D
CB80-  41 4B 45 00            TAKE        #7A=122   F8DF
CB84-  59 50 45 00            TYPE        #7B=123   FE98
CB88-  52 41 43 4B 00         TRACK       #7C=124   F130

CB8D-  53 45 52 00            USER        #7D=125   EA7F
CB91-  4E 54 4B 45 4E 00      UNTKEN      #7E=126   E8E1
CB97-  E3 47 00               USING       000       EE99 ;#E3 token BASIC "SIN"
CB9A-  4E 50 52 4F 54 00      UNPROT      000       E9F9

CBA0-  49 53 55 A2 00         VISUHIRES   000       E9F0 ;#A2 token HIRES
CBA5-  55 53 45 52 00         VUSER       000       F121
```

```
CBAA-  49 44 54 48 00          WIDTH       000       E740
CBAF-  49 4E 44 4F 57 00       WINDOW      000       F210

CBB5-  9A 00                   RESTORE     000       E7D9 ;#9A token "RESTORE"

CBB7-  5D 00                   ]           000       EC04 ;#5D token ASCII "]"

CBB9-  FF 00                   255         000       E83E
```

TABLE DES INITIALES DES MOTS-CLÉS SEDORIC

	Dump Hexadécimal	reg A	1 ère 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 **06**	02	C	CA02	07	06
CBC7-	**24** CA **0D 0B**	03	D	CA24	13	11
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 **04**	14	U	CB8D	93	04
CC0F-	**A0** CB **61 02**	15	V	CBA0	97	02
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 SEDORIC

Les adresses (-1) sont regroupées par initiale des mots-clés, sous la forme LL puis HH)
Les 12 octets modifiés de cette table sont indiqués en gras dans le dump hexadécimal.

CC27- A 06FE/06FE/DDEB/90EB

CC2F- B DDF0/50F1/DFFE

CC35- C 47F1/8CFB/56F1/4CDE/BBF9/**FEE9**

CC3F- D 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/**F5E9**/BFE7/8FF9

CCA9- Q F4E7/E0EB

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

CCBF- S 3AEA/53F1/52E8/52E8/**FBE9**/**F2E9**/4CDD/49DD/52DD/4FDD/FBE5/59F1/95F9

CCD9- T 9CE8/DEF8/97FE/2FF1

CCE1- U 7EEA/E0E8/98EE/**F8E9**

CCEB- V **EFE9**/20F1

CCED- W 3FE7/0FF2

CCF1- autre D8E7/03EC/3DED

TABLE NOM ET EXTENSION PAR DÉFAUT

CCF7-	43 4F 4D	COM extension courante
CCFA-	42 41 4B	BAK extension pour SAVEU
CCFD-	43 4F 4D	COM extension par défaut
CD00-	3F 3F 3F 3F 3F 3F 3F 3F 3F	????????? Joker * ou nom_de_fichier_ambigu omis
CD09-	42 41 4B	BAK (ne semble pas utilisée)

TABLE DE CONSTANTES DIVERSES

CD0C-	28 50 35 5D	valeurs par défaut pour la commande WIDTH
CD10-	00 00 01 01 FA BF 23 34 36 37 FF	utilisé par la commande STRUN (CALL#467 #FF)
CD1B-	7B 0E FA 35 10	utilisé par les commandes LINE et BOX (0,174532925)
CD20-	81 C9 0F DA A2	utilisé par la commande LINE et BOX (-1,57079633)
CD25-	C6 C9 88 02 88 02	utilisé par la commande OPEN (initialisation de **FI**)
CD2B-	4F 46 46	OFF
CD2E-	53 45 54	SET
CD31-	C7 81 C2 82 45 D3 66 A5 C8 A3 8F D2 42 B5 98 E0	non identifié

TABLE DE CONVERSION QWERTY / AZERTY

CD41-	B1 BE AE AA 82 93	codes de touche pour ; M Z A W Q
CD47-	AE AA B1 BE 93 82	codes de touche pour M ; W Q Z A

TABLE DE CONVERSION ACCENT OFF / ACCENT SET

```
CD4D-  40 10 08 1C 02 1E 22 1E 00     code ASCII #40 @ -> à
CD56-  5C 00 00 1E 20 20 20 1E 04     code ASCII #5C \ -> ç
CD5F-  7B 04 08 1C 22 3E 20 1E 00     code ASCII #7B { -> é
CD68-  7C 10 08 22 22 22 26 1A 00     code ASCII #7C | -> ù
CD71-  7D 10 08 1C 22 3E 20 1E 00     code ASCII #7D } -> è
CD7A-  7E 1C 22 1C 22 3E 20 1E 00     code ASCII #7E ▌-> ê
```

TABLE DE CONSTANTES DIVERSES

CD83-	41 58 59 50 B8	A X Y P et DEF pour la commande USER
CD88-	0A 64 E8 10	LL des valeurs 10, 100, 1000 et 10000
CD8C-	00 00 03 27	HH des valeurs 10, 100, 1000 et 10000
CD90-	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)

CD94-	00	45 4E	EN	Error Number (mise à jour après une erreur)
CD96-	02	45 4C	EL	Error Line (mise à jour après une erreur)
CD98-	04	49 4E	IN	mise à jour par INstr (position de la sous-chaîne dans la chaîne)
CD9A-	06	4F 4D	OM	Output Mode (mode de sortie de LINPUT)
CD9C-	08	53 4B	SK	SeeK (nombre d'occurrences de la chaîne trouvé par SEEK)
CD9E-	0A	46 54	FT	File Type (type fichier chargé) (mis à jour par LOAD)
CDA0-	0C	45 4F	EO	variable non identifiée
CDA2-	0E	52 41	RA	affiche Registre A du microprocesseur (commande USER)
CDA4-	10	52 58	RX	affiche Registre X du microprocesseur (commande USER)
CDA6-	12	52 59	RY	affiche Registre Y du microprocesseur (commande USER)
CDA8-	14	52 50	RP	affiche Registre d'état du microprocesseur. (commande USER)
CDAA-	16	45 46	EF	Existing File (si le fichier existe EF = 1 , sinon EF = 0)
CDAC-	18	53 54	ST	STart adress (adresse de début du fichier) (LOAD)
CDAE-	1A	45 44	ED	EnD adress (adresse de fin du fichier) (LOAD)
CDB0-	1C	45 58	EX	EXécution adress (adresse d'exécution du fichier) (LOAD)
CDB2-	1E	43 58	CX	Curseur X (abscisse du curseur TEXT ou HIRES) (HCUR et LCUR)
CDB4-	20	43 59	CY	Curseur Y (ordonnée curseur TEXT ou HIRES) (HCUR et LCUR)
CDB6-	22	46 50	FP	Free Piste (n° de piste du secteur libéré par PRESEC)
CDB8-	24	46 53	FS	Free Secteur (n° du secteur libéré par PRESEC)
CDBA-	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 degrés. 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 D'ERREURS SEDORIC (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**

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

CE06-06 57 49 4C 44 43 41 52 44 28 53 29 20 4E 4F 54 20 41 4C 4C 4F 57 45 **C4**
WILDCARD(S)_NOT_ALLOWE**D**

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

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

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

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

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

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

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

CE7D-0E 46 49 4C 45 20 4E 4F 54 20 4F 50 45 **CE**
FILE_NOT_OPE**N**

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

CE9B-10 45 4E 44 20 4F 46 20 46 49 4C **C5**
END_OF_FIL**E**

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

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

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

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

AUTRES MESSAGES SEDORIC (ZONE CEE7)

(Le numéro d'ordre X est indiqué à gauche sous l'adresse).
Deux de ces messages ont été modifiés afin de différencier la version 3.0 (14 octets différents): "_(Master)_" devient "_V3_(Mst)_" et "_(Slave_)_" devient "_V3_(Slv)_".

CEE7- 0A 0D 54 52 41 43 4B **BA**
01 LFCRTRACK**:**

CEEF- 20 53 45 43 54 4F 52 **BA**
02 _SECTOR**:**

CEF7- 20 57 52 49 54 45 20 46 41 55 4C 54 **A0**
03 _WRITE_FAULT**_**

CF04- 20 52 45 41 44 20 46 41 55 4C 54 **A0**
04 _READ_FAULT**_**

CF10- 0A 0D 20 42 52 45 41 4B 20 4F 4E 20 42 59 54 45 20 **A3**
05 LFCRBREAK_ON_BYTE_#

CF22- 0D 0A 44 72 69 76 65 **A0**
06 CRLFDrive**_**

CF2A- 20 56 33 20 28 4D 73 74 29 **A0**
07 _V3_(Mst)**_**

CF34- 20 73 65 63 74 6F 72 73 20 66 72 65 65 20 **A8**
08 _sectors_free **(**

CF43- 20 46 69 6C 65 73 **A0**
09 _Files**_**

CF4A- 20 49 53 20 50 52 4F 54 45 43 54 45 **C4**
0A _IS_PROTECTE**D**

CF57- 20 28 59 29 65 73 20 6F 72 20 28 4E 29 6F **BA**
0B _(Y)es_or_(N)o**:**

CF66- 20 44 45 4C 45 54 45 44 0D **8A**
0C _DELETED CR **LF**

CF70- 49 4E 53 45 52 54 20 4D 41 53 54 45 52 **A0**
0D INSERT_MASTER**_**

CF7E- 41 4E 44 20 50 52 45 53 53 20 27 52 45 54 55 52 4E **A7**

0E AND_PRESS_'RETURN'

CF90-0F 20 41 4C 52 45 41 44 59 20 45 58 49 53 54 53 0A **8D**
_ALREADY_EXISTSLF**CR**

CFA1-10 20 2D 2D 3E **A0**
_-->_

CFA6-11 55 53 45 52 **A0**
USER_

CFAB-12 20 56 33 20 28 53 6C 76 29 **A0**
_V3_(Slv)_

CFB5-13 20 28 54 79 70 65 **BD**
_(Type=

CFBC-14 29 **A0**
)_

CFBE-15 20 44 49 53 43 20 49 4E 20 44 52 49 56 45 **A0**
_DISC_IN_DRIVE_

Rappel: _ = simple espace matérialisé par ce caractère de soulignement
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

(**R**ead / **W**rite **T**rack / **S**ector)

*** Cette partie a été écrite grâce aux informations qui m'ont été fournies par Fabrice Francès
*** Sans lui, je n'aurais jamais compris comment l'ORIC accède aux lecteurs de disquettes.
*** Qu'il en soit cordialement remercié.

En entrée, X contient le n° de commande pour le FDC.
En sortie, Z = 1 si pas d'erreur, sinon Z = 0
DRIVE (C000), PISTE (C001), SECTEUR (C002) et RWBUF (C003/C004) doivent être à jour.

Variables et registres utilisés

0310 à 0313 registres du FDC 1793 (**F**loppy **D**isc **C**ontroller)

0310- En lecture, c'est le "Status Register" (**registre d'état** du FDC).
En écriture, c'est le "Command Register" (**registre de commande** du FDC) qui reçoit X, le code de commande (voir ci-dessous)

0311- C'est le "Track Register" (**registre de piste** du FDC). Il indique le numéro de piste et est automatiquement mis à jour par les commandes de déplacement de tête et utilisé lors des comparaisons avec les numéros de pistes enregistrés dans les en-têtes de secteur dans les commandes de lecture / écriture.

0312- C'est le "Sector Register" (**registre de secteur** du FDC). Il est utilisé pour spécifier le secteur à lire / écrire et lors des comparaisons avec les en-têtes de secteurs lus/écrits.

0313- C'est le "Data Register" (**registre de données** du FDC) registre tampon lors des opérations de lecture / écriture. Utilisé aussi pour programmer la piste voulue lors d'une commande de déplacement de tête..
Pour plus d'information, voir en ANNEXE.

0314 à 0317 MICRODISC (1 registre par lecteur? Seul 0314 semble utilisé)

En écriture, c'est un registre (LS273) regroupant différents bits de configuration de l'électronique du MICRODISC :

b0 : masquage de l'interruption IRQ du FDC (0: masquée, 1: autorisée)
b1: activation de la ligne ROMDIS (0: ROM interne désactivée)
b2 : sélection du type de séparation de donnée
b3: simple/double densité (0: double, 1: simple)
b4: sélection face (0: face 0, 1: face 1)
b5,b6: sélection lecteur (0 à 3)
b7: sélection EPROM MICRODISC (0: sélectionnée, 1: inhibée)

En lecture, c'est seulement l'état de la ligne IRQ du FDC dans le bit 7

0318-031B ligne DRQ (Data ReQuest) du FDC (1 registre par lecteur? Seul 0318 semble utilisé)

dans le bit 7, lecture seulement

04FB- mémorise la valeur à destination du registre 0314 du MICRODISC (code DRIVE , FACE etc.)

C005- X = code de commande à destination du FDC.

Les codes de commande du FDC se répartissent en 4 catégories (de type I à IV). Voici les principaux:

I	Restore (#08)	positionne la tête sur la piste #00
I	Seek (#18)	positionne la tête sur piste requise et met à jour le "Track Register"
I	Step (#38)	avance la tête d'une piste dans la même direction et idem
I	Step-In (#58)	avance la tête d'une piste vers les numéros croissants et idem
I	Step-Out (#78)	avance la tête d'une piste vers la piste 0 et idem
II	Read Sector (#80)	lit un secteur sans tester le n° de face
II	Read Sector (#90)	lit plusieurs secteurs sans tester le n° de face
II	Write Sector (#A0)	écrit un secteur sans tester le n° de face
II	Write Sector (#B0)	écrit plusieurs secteurs sans tester le n° de face
III	Read Address (#C0)	lit le prochain champ ID (n° de piste, n° de face, n° de secteur, taille du secteur, CRC1 et CRC2), vérifie la validité
III	Read Track (#E0)	lecture piste, tous les octets de gaps, en-têtes et data sont lus
III	Write Track (#F0)	écriture piste, formate une piste
IV	Force Interrupt (#DX)	interruption du processus en cours

Pour chacun de ces codes de commandes, un ou plusieurs bits sont optionnels. Les valeurs données entre parenthèses ne sont que des exemples (**voir en ANNEXE pour de plus amples détails**).

C006- 1 ou 2 nombre de tentatives autorisées en cas de secteur non trouvé
C007- 8 nombre de tentatives autorisées en cas d'erreur de transfert
C017- résumé de certaines erreurs de l'I/O: 0 si pas d'erreur sinon mise à 1 des bits suivants:
b6: protection en écriture
b4: secteur non trouvé
b3: erreur de CRC (code de redondance cyclique)
b2: perte de donnée
Nota: il manque le bit 5 ! (qui correspond à une erreur d'écriture, cette erreur n'est donc pas détectée !)

C060 à C065 buffer pour la lecture d'un en-tête secteur

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	

Entrée secondaire pour rebouclage

CFEB-	8C 06 C0	STY C006	nombre de tentatives autorisées en cas de secteur non trouvé
CFEE-	A0 08	LDY #08	
CFF0-	8C 07 C0	STY C007	nombre de tentatives autorisées en cas d'erreur de transfert

Point d'entrée réel: exécution de la commande X

CFF3-	48	PHA	sauvegarde de la valeur de A sur pile (valeur “pokée” en 030E)
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 la valeur correspondante dans la table CD90. Cette table contient les 4 octets à destination du port 0314 suivant le lecteur sélectionné (b5 et b6 de 0 à 3), avec toujours la face 0 sélectionnée (b4 = 0), L'EPROM MICRODISC activée (b7 = 1) et la ROM interne désactivée (b1 = 0), la double densité MFM (b3 = 0), et l'interruption du FDC masquée (b0 = 0). 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.
CFFD-	2C 01 C0	BIT C001	teste b7 du n° de PISTE (à 1 si deuxième face)
D000-	10 02	BPL D004	saute ligne suivante si première face visée
D002-	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)
D004-	8D FB 04	STA 04FB	la valeur à destination de 0314 est mémorisée en 04FB parce que le port est en écriture seulement
D007-	CC 0B C0	CPY C00B	le DRIVE demandé est-il le drive actif?
D00A-	F0 0A	BEQ D016	si oui, on continue en D016
D00C-	8C 0B C0	STY C00B	sinon, C00B est mis à jour pour activation
D00F-	20 EA D0	JSR D0EA	lit le numéro de piste sous la tête
D012-	90 02	BCC D016	si C = 0 (opération réussie), continue en D016
D014-	68	PLA	sinon, récupère A et retourne
D015-	60	RTS	

Le numéro de piste a correctement été obtenu

D016-	AD 03 C0	LDA C003	
D019-	AC 04 C0	LDY C004	
D01C-	85 F3	STA F3	F3/F4 mis à jour avec RWBUF

D01E-	84 F4	STY F4	
D020-	78	SEI	interdit les interruptions
D021-	A9 20	LDA #20	masque pour décodage de la commande, on teste dans l'ordre les bits 7, 6 et 5 pour différencier les commandes de type I, II, III et IV. En fait, après optimisation par rapport au code d'ORIC DOS, la seule chose qui intéresse Broche ici est de savoir s'il faut rajouter une commande de déplacement de tête avant d'effectuer la commande (c'est le cas des commandes de lecture/écriture piste et secteur, mais inutile dans le cas des commandes de déplacement de tête ou de détermination de la piste sous la tête)
D023-	2C 05 C0	BIT C005	teste b5, b6 et b7 du code de commande
D026-	10 29	BPL D051	si b7=0, commande de type I
D028-	50 02	BVC D02C	si b7=1 et b6=0, commande de type II, il faut déplacer d'abord la tête
D02A-	F0 25	BEQ D051	si b7=b6=1 et b5=0, commande Read Address ou Force Interrupt, inutile de déplacer la tête

<u>On a donc ici une commande de lecture/écriture piste/secteur</u>

D02C-	AD 01 C0	LDA C001	compare la PISTE demandée
D02F-	CD 0C C0	CMP C00C	avec la piste active
D032-	F0 06	BEQ D03A	continue en D03A si identiques
D034-	48	PHA	sinon, empile PISTE pour ajouter
D035-	8A	TXA	l'indicateur V (vérification du numéro de piste)
D036-	09 04	ORA #04	à la commande de lecture/écriture
D038-	AA	TAX	pour se prévenir d'un mauvais positionnement de la tête
D039-	68	PLA	récupère PISTE
D03A-	29 7F	AND #7F	force à 0 le b7 de PISTE (à 1 si deuxième face)
D03C-	CD 11 03	CMP 0311	la PISTE demandée est-elle sous la tête?
D03F-	F0 10	BEQ D051	si oui, continue en D051
D041-	8A	TXA	sinon, sauve le code de commande en A
D042-	A2 18	LDX #18	exécute la commande n° #18, c'est à dire
D044-	20 F3 CF	JSR CFF3	Seek Track + indicateur h: engagement de la tête et déplacement
D047-	8D 05 C0	STA C005	remet le code de commande précédent dans C005
D04A-	AA	TAX	et dans X
D04B-	AD 13 03	LDA 0313	recopie piste programmée dans le registre piste
D04E-	8D 11 03	STA 0311	(inutile puisque la commande Seek l'a mis à jour)

<u>PISTE est en place sous la tête</u>

D051-	AD 01 C0	LDA C001	PISTE visée -> piste active
D054-	8D 0C C0	STA C00C	
D057-	29 7F	AND #7F	programme le numéro de piste à atteindre
D059-	8D 13 03	STA 0313	(sans conséquence pour les commandes de lecture/écriture)
D05C-	AD 02 C0	LDA C002	programme le numéro de secteur désiré
D05F-	8D 12 03	STA 0312	(sans conséquence pour les commandes de déplacement)
D062-	A0 00	LDY #00	Y = #00 préset pour délai
D064-	8A	TXA	A reçoit le code de commande
D065-	30 03	BMI D06A	délai si X positif (commande de déplacement de tête), sinon continue en D06A

Delai de 1ms avant de déclencher la commande de déplacement (inutile selon F. Francès)

D067-	88	DEY	
D068-	D0 FD	BNE D067	pause: reboucle tant que Y ne revient pas à 0
D06A-	AD FB 04	LDA 04FB	récupère la programmation MICRODISC
D06D-	09 01	ORA #01	masque 0000 0001 force b0 à 1
D06F-	8D 14 03	STA 0314	et autorise l'interruption IRQ du FDC
D072-	8E 10 03	STX 0310	envoie la commande au FDC
D075-	8A	TXA	teste les 4 bits forts de X:
D076-	29 F0	AND #F0	1111 0000 force à 0 les b0 à b3
D078-	C9 E0	CMP #E0	est-ce que la commande est Read Track ?
D07A-	58	CLI	autorise les interruptions
D07B-	F0 04	BEQ D081	si oui, continue en D081
D07D-	29 20	AND #20	distingue les commandes de lecture de celles d'écriture (toutes les commandes de lecture ont le bit 5 à 0 sauf Read Track, d'où le test précédent...). Nota: ce test envoie aussi les commandes de type I sur les routines de lecture ou d'écriture, mais heureusement elles restent bloquées sur une attente désespérée du signal DRQ, jusqu'à ce que l'IRQ de complétion de commande arrive...
D07F-	D0 12	BNE D093	si écriture, continue en D093

Lecture sur disquette

D081-	AD 18 03	LDA 0318	attente du signal DRQ (un octet complet reçu)
D084-	30 FB	BMI D081	reboucle tant qu'il ne passe pas à 0 (actif à l'état bas à cause d'une porte NAND)
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 sous-programme se fait sur ordre du contrôleur (interruption), à une adresse spécifiée par ailleurs

Ecriture sur disquette

D093-	AD 18 03	LDA 0318	attente du signal DRQ (le FDC demande un octet à écrire sur disquette)
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 sous-programme se fait sur ordre du contrôleur (interruption), à une adresse spécifiée par ailleurs

Handler d' IRQ

(Sous-programme vectorisé en FFFE)

D0A5-	2C 14 03	BIT 0314	vérifie si l'interruption vient du FDC
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	dépile le contexte de l'interruption,
D0AE-	68	PLA	on est donc de nouveau dans la routine XRWTS
D0AF-	68	PLA	
D0B0-	AD FB 04	LDA 04FB	réécrit la valeur courante de 0314 en masquant l'IRQ FDC
D0B3-	8D 14 03	STA 0314	
D0B6-	18	CLC	
D0B7-	AD 10 03	LDA 0310	lit le registre d'état du FDC (ce qui efface l'IRQ)
D0BA-	29 5C	AND #5C	récupère les conditions d'erreur (sauf l'erreur d'écriture !)
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 la commande n'est pas de type I, garde les conditions d'erreur précédentes
D0C2-	A0 00	LDY #00	sinon n'en tient pas compte
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	retourne une erreur si on a tenté d'écrire sur une disquette protégée en écriture
D0CB-	98	TYA	reprend le "bilan" précédent dans A
D0CC-	29 10	AND #10	vérifie qu'on n'a pas une erreur "secteur non trouvé"
D0CE-	F0 0D	BEQ D0DD	continue en D0DD si pas une erreur "secteur non trouvé"
D0D0-	CE 06 C0	DEC C006	décrémente le compteur de tentatives "secteur non trouvé"
D0D3-	F0 05	BEQ D0DA	et renvoie une erreur si le nombre de tentatives est écoulé
D0D5-	20 EA D0	JSR D0EA	sinon, vérifie que l'on peut lire un en-tête de secteur sur la piste
D0D8-	90 0D	BCC D0E7	si oui, saute pour retenter la lecture/écriture
D0DA-	38	SEC	positionne carry pour indiquer une erreur
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: vérifie qu'il n'y a pas eu d'erreur de transfert (mauvais CRC ou donnée perdue)
D0E0-	F0 F9	BEQ D0DB	et renvoie "pas d'erreur" (carry=0) dans ce cas
D0E2-	CE 07 C0	DEC C007	sinon décrémente le compteur de tentatives "erreur de transfert"
D0E5-	F0 F3	BEQ D0DA	et renvoie une erreur après écoulement du nombre de tentatives
D0E7-	4C F7 CF	JMP CFF7	retente la lecture/écriture

Test de la piste sous la tête

Cette routine a deux raisons d'être: la première, c'est déterminer si une piste est vierge (aucun en-tête lisible, on ne peut donc ni lire, ni écrire de secteur). La deuxième, c'est que le FDC ne peut garder trace que d'une seule position de tête, et il faut bien jongler avec les positions des têtes des 4 lecteurs (il serait beaucoup plus efficace de mémoriser ces positions en mémoire... mais on ne badine pas avec la sécurité)

D0EA-	8A	TXA	
D0EB-	48	PHA	
D0EC-	AD 03 C0	LDA C003	sauvegarde X et RWBUF
D0EF-	48	PHA	
D0F0-	AD 04 C0	LDA C004	
D0F3-	48	PHA	
D0F4-	A9 60	LDA #60	
D0F6-	A0 C0	LDY #C0	buffer positionné en C060, on va récupérer les 6 octets d'un en-tête secteur
D0F8-	8D 03 C0	STA C003	
D0FB-	8C 04 C0	STY C004	
D0FE-	AD 06 C0	LDA C006	sauve dans A le compteur de tentatives "secteur non trouvé"
D101-	A2 C0	LDX #C0	commande Read Address pour le FDC: cherche un en-tête de secteur quelconque
D103-	A0 01	LDY #01	une seule tentative (on fait quand même 5 fois le tour de la disquette...)
D105-	20 EB CF	JSR CFEB	exécute commande X
D108-	8D 06 C0	STA C006	régénère l'ancienne valeur de C006
D10B-	68	PLA	
D10C-	8D 04 C0	STA C004	
D10F-	68	PLA	récupère RWBUF initial
D110-	8D 03 C0	STA C003	
D113-	B0 06	BCS D11B	erreur si aucun en-tête trouvé, on propage l'erreur (carry)
D115-	AD 12 03	LDA 0312	sinon, en fin de commande Read Address, le registre secteur du FDC contient en fait le numéro de piste lu dans l'en-tête
D118-	8D 11 03	STA 0311	que l'on écrit donc dans le registre piste pour le mettre à jour
D11B-	68	PLA	
D11C-	AA	TAX	récupère X et le copie en C005 (code commande)
D11D-	8E 05 C0	STX C005	
D120-	60	RTS	

Handler NMI (bouton NMI sous l'ORIC)

Sous-programme vectorisé en FFFA

D121-	AD FB 04	LDA 04FB	récupère la programmation MICRODISC
D124-	8D 14 03	STA 0314	masque l'interruption IRQ du FDC (bit 0 à 0)
D127-	AD 10 03	LDA 0310	teste le bit Busy du FDC
D12A-	4A	LSR	en le poussant dans C
D12B-	90 05	BCC D132	à 0 si aucune commande en cours
D12D-	A9 D0	LDA #D0	sinon, interrompt la commande en cours
D12F-	8D 10 03	STA 0310	avec une commande Force Interrupt
D132-	38	SEC	met C à 1
D133-	4C F8 04	JMP 04F8	et continue en 04F8 (NMIRAM)

Sous-programme affichage "LFCRBREAK_ON_BYTE_#"

En entrée X/F2 contiennent l'adresse de l'instruction suivant le "BREAK". En sortie, réinitialisation de la pile et retour au Ready après affichage du message "LFCRBREAK_ON_BYTE_#" et de l'adresse.

D136-	86 F3	STX F3	sauve valeur de X dans F3 (LL de l'adresse suivante)
D138-	A2 04	LDX #04	indexe le message "LFCRBREAK_ON_BYTE_#"
D13A-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
D13D-	38	SEC	prépare une soustraction
D13E-	A6 F3	LDX F3	récupère dans X le LL de l'adresse suivante
D140-	A5 F2	LDA F2	récupère dans A le HH de l'adresse suivante
D142-	E9 02	SBC #02	calcule l'adresse du “BREAK” (A = LL - #02)
D144-	B0 01	BCS D147	saute l'instruction suivante si pas de retenue
D146-	CA	DEX	sinon décrémente aussi HH de l'adresse suivante
D147-	48	PHA	sauve LL, le résultat de la soustraction
D148-	8A	TXA	HH passe dans A pour être affiché en premier
D149-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
D14C-	68	PLA	récupère LL dans A pour l'afficher en second
D14D-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
D150-	58	CLI	autorise les interruptions
D151-	A2 FF	LDX #FF	réinitialise la pile (transfère #FF dans S)
D153-	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 sous-programme D5D8 en RAM overlay, 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 sous-programme 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 mémoire vers le haut

En CE/CF adresse du premier 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     pour le message "DISP_TYPE_MISMATCH_ERROR" (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 selon le n° en 33/34 à partir du début

Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le premier octet de lien)

D19C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D19F- DE C6 B3 C6 adresse ROM 1.0 adresse ROM 1.1
D1A3- 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 premier octet de lien)

D1A4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1A7- EE C6 C3 C6 adresse ROM 1.0 adresse ROM 1.1
D1AB- 60 RTS

Place TXTPTR au début du programme BASIC

D1AC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1AF- 65 C7 3A C7 adresse ROM 1.0 adresse ROM 1.1
D1B3- 60 RTS

Exécute la commande "LIST" simplifiée

D1B4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1B7- 99 C7 6C C7 adresse ROM 1.0 adresse ROM 1.1
D1BB- 60 RTS

ROM 1.0: Simple RTS, ROM 1.1: Met l'imprimante en service

D1BC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1BF- 40 C8 16 C8 adresse ROM 1.0 adresse ROM 1.1
D1C3- 60 RTS

Met l'imprimante hors service

D1C4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1C7- 3D C8 2F C8 adresse ROM 1.0 adresse ROM 1.1
D1CB- 60 RTS

Exécute la commande "RESTORE" du BASIC

D1CC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1CF- 1F C9 52 C9 adresse ROM 1.0 adresse ROM 1.1
D1D3- 60 RTS

"UNDEF'D_STATEMENT_ERROR" (GOSUB)

D1D4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1D7- F1 C9 23 CA adresse ROM 1.0 adresse ROM 1.1
D1DB- 60 RTS

Calcule le déplacement à l'instruction suivante, met à jour TXTPTR en ajoutant Y

D1DC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1DF- 1C CA 4E CA adresse ROM 1.0 adresse ROM 1.1
D1E3- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1E6- 0D CA 3F CA adresse ROM 1.0 adresse ROM 1.1
D1EA- 60 RTS

Exécute la commande "IF"

D1EB- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D1EE- 41 CA 73 CA adresse ROM 1.0 adresse ROM 1.1
D1F2- 60 RTS

XCRGOT + évalue le numéro de ligne à TXTPTR (résultat en 33/34)

D1F3- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z

= 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

```
D1F6-  20 D8 D5   JSR D5D8   XROM exécute à partir de la RAM une routine ROM
D1F9-  98 CA E2 CA           adresse ROM 1.0 adresse ROM 1.1
D1FD-  60         RTS
```

Affecte un nombre à une variable

```
D1FE-  20 D8 D5   JSR D5D8   XROM exécute à partir de la RAM une routine ROM
D201-  EF CA 39 CB           adresse ROM 1.0 adresse ROM 1.1
D205-  60         RTS
```

Va à la ligne

```
D206-  20 D8 D5   JSR D5D8   XROM exécute à partir de la RAM une routine ROM
D209-  9F CB F0 CB           adresse ROM 1.0 adresse ROM 1.1
D20D-  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
```

Exige 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
```

Exige 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 SEDORIC (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     XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
                                et RTS
```

Place dans AY, B6/B7 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)

```
D24C-  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D24F-  17 D2 A9 D2               adresse ROM 1.0 adresse ROM 1.1
D253-  60          RTS
```

Transfère le nombre de AY dans ACC1 (signé)

```
D254-  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D257-  ED D3 99 D4               adresse ROM 1.0 adresse ROM 1.1
D25B-  60          RTS
```

Interdit le mode direct

```
D25C-  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D25F-  19 D4 D2 D4               adresse ROM 1.0 adresse ROM 1.1
D263-  60          RTS
```

Réserve une place en mémoire pour une chaîne de longueur A

Sauvegarde la longueur en D0 et l'adresse en D1/D2

```
D264-  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D267-  F0 D4 AB D5               adresse ROM 1.0 adresse ROM 1.1
D26A-  60          RTS
```

"STRING_TOO_LONG_ERROR"

```
D26C-  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D26F-  C7 D6 82 D7               adresse ROM 1.0 adresse ROM 1.1
D273-  60          RTS
```

Vérifie s'il y a une chaîne à TXTPTR

Retourne longueur dans A et adresse dans XY et dans 91/92

```
D274-  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)
```

D277- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D27A- 15 D7 D0 D7 adresse ROM 1.0 adresse ROM 1.1 (longueur -> A avec N et
D27E- 60 RTS Z selon cette longueur, adresse -> XY et 91/92)

Evalue un nombre entier à TXTPTR et le retourne dans X

D27F- 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

D282- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D285- 10 D8 CB D8 adresse ROM 1.0 adresse ROM 1.1 (prend entier dans X)
D289- 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

D28A- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D28D- 6B D8 26 D9 adresse ROM 1.0 adresse ROM 1.1
D291- 60 RTS

Prend 2 coordonnées xy à TXTPTR et les retourne dans #2F8(x) et X(y)

D292- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D295- 96 D9 22 DA adresse ROM 1.0 adresse ROM 1.1
D299- 60 RTS

Effectue AY - ACC1 -> ACC1 (soustraction)

D29A- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D29D- 80 DA 0B DB adresse ROM 1.0 adresse ROM 1.1
D2A1- 60 RTS

Additionne le contenu de ACC1 (floating point accumulator) et la valeur pointée par AY et replace le résultat dans ACC1

D2A2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2A5- 97 DA 22 DB adresse ROM 1.0 adresse ROM 1.1
D2A9- 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 vers les adresses XY à 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)

```
D30A- 20 D8 D5    JSR D5D8     XROM exécute à partir de la RAM une routine ROM
D30D- C7 EC E0 ED              adresse ROM 1.0 adresse ROM 1.1
D311- 60          RTS
```

Effectue la commande "DRAW"

```
D312- 20 D8 D5    JSR D5D8     XROM exécute à partir de la RAM une routine ROM
D315- 79 F0 10 F1              adresse ROM 1.0 adresse ROM 1.1
D319- 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.

```
D31A- 20 D8 D5    JSR D5D8     XROM exécute à partir de la RAM une routine ROM
D31D- 94 F4 EF F4              adresse ROM 1.0 adresse ROM 1.1
D321- 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.

```
D322- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D325- 35 F5 90 F5               adresse ROM 1.0 adresse ROM 1.1
D329- 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

```
D32A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D32D- CB F7 01 F8               adresse ROM 1.0 adresse ROM 1.1
D331- 60          RTS
```

Régénère le jeu de caractères normaux (descend de la ROM dans la RAM)

```
D332- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D335- 3E F9 82 F9               adresse ROM 1.0 adresse ROM 1.1
D339- 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.

```
D33A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D33D- E2 00 E2 00               adresse ROM 1.0 adresse ROM 1.1
D341- 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.

```
D342- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
D345- E8 00 E8 00               adresse ROM 1.0 adresse ROM 1.1
D349- 60          RTS
```

ROUTINES SEDORIC D'USAGE GENERAL

Copie NOM et EXT de la table CCF7 dans BUFNOM

D34A-	A0 09	LDY #09	Y = #09 copiera 3 octets
D34C-	2C A0 00	BIT 00A0	continue en D34F
D34D-	A0 00	LDY #00	Y = #00 copiera 12 octets
D34F-	BD F7 CC	LDA CCF7,X	lecture dans table CCF7 selon X à l'entrée
D352-	99 29 C0	STA C029,Y	écriture dans BUFNOM selon Y (ci-dessus)
D355-	C8	INY	Y suivant (de 0 à 12 ou de 9 à 12)
D356-	E8	INX	X suivant
D357-	C0 0C	CPY #0C	
D359-	D0 F4	BNE D34F	et reboucle tant que Y est < 12
D35B-	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'écrire à la bonne place dans BUFNOM.

Affiche (X+1) ème 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 externes contenue dans EXTER
D362-	D0 12	BNE D376	et continue en D376

XAFSC affiche le (X+1) ème 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 premier message: "LFCRTRACK:")
D36A-	D0 0A	BNE D376	et continue en D376

Affiche le (X+1) ème 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 premier message)
D370-	D0 04	BNE D376	et continue en D376

Affiche le (X+1) ème 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 premier 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 premier
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 report 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	XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A

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

XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A

D3A1-	08	PHP	sauve P, c'est à dire les indicateurs initialisés 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 SEDORIC

RECHERCHE L'ADRESSE D'EXÉCUTION D'UN MOT-CLÉ SEDORIC

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 première lettre mot-clé

- Lecture de l'adresse premier 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 SEDORIC (mis en X)

- Examen de la correspondance avec l'un des mots-clés ayant la même initiale
  si trouve: exécution en D417
  sinon: continue en D428

D3AE-	A2 00	LDX #00	
D3B0-	8E FD 04	STX 04FD	mise à zéro de ERROR (numéro de l'erreur SEDORIC)
D3B3-	BA	TSX	sauve pointeur de pile en SAUVES
D3B4-	8E 23 C0	STX C023	pour sortie éventuelle par erreur
D3B7-	A5 E9	LDA E9	
D3B9-	A4 EA	LDY EA	
D3BB-	8D 1F C0	STA C01F	sauvegarde de TXTPTR en C01F/C020 (SVTPTR)
D3BE-	8C 20 C0	STY C020	
D3C1-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
D3C4-	E9 41	SBC #41	teste si #40<A<#5C (A contient-il une lettre)
D3C6-	A0 1A	LDY #1A	#1A = valeur finale si le caractère dans A n'est pas une lettre
D3C8-	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)

D3CA-	C9 1A	CMP #1A	on compare en réalité à #41 + #1A = #5B
D3CC-	B0 04	BCS D3D2	A>=#5B pas une lettre, donc A = #1A par défaut
D3CE-	A8	TAY	#00 <= Y <= #19 pour un caractère de A à Z

D3CF-	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
D3D2-	98	TYA	A de #00 (A) à #19 (Z) ou #1A (autre caractère)
D3D3-	0A	ASL	calcule index Y en multipliant A par 4
D3D4-	0A	ASL	car on va indexer avec Y une sous-table en CBBB
D3D5-	A8	TAY	constituée de groupes de 4 octets
D3D6-	B9 BB CB	LDA CBBB,Y	lit 2 premiers octets (adresse dans table principale
D3D9-	85 18	STA 18	(C9DE-CBBA) où commencent mots-clés de même
D3DB-	B9 BC CB	LDA CBBC,Y	initiale) et place ces 2 octets en 18/19
D3DE-	85 19	STA 19	(travail encodage/décodage mots clés)
D3E0-	B9 BE CB	LDA CBBE,Y	lit le nombre de mots-clés de même initiale
D3E3-	85 F2	STA F2	et le place en F2 (TRAV0)
D3E5-	BE BD CB	LDX CBBD,Y	lit n° d'ordre mot-clé SEDORIC, le garde en X
D3E8-	C6 F2	DEC F2	décrémente le nombre de mots-clés de même initiale
D3EA-	30 3C	BMI D428	si négatif: branche en D428 ("non trouvé")
D3EC-	A0 FF	LDY #FF	pour chaque mot-clé examiné consulte la table
D3EE-	C8	INY	principale à l'adresse où commence la liste des
D3EF-	B1 18	LDA (18),Y	mots-clés de même initiale (privés de la première
D3F1-	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"
D3F3-	85 F3	STA F3	chaque lettre, placée en F3, sera comparée à
D3F5-	B1 E9	LDA (E9),Y	la lettre correspondant à TXTPTR et chargée en A
D3F7-	C9 61	CMP #61	mise en MAJUSCULE: compare à "a"
D3F9-	90 06	BCC D401	inutile de continuer si lettre < "a"
D3FB-	C9 7B	CMP #7B	compare à "é" (lettre qui suit "z")
D3FD-	B0 02	BCS D401	inutile de continuer si lettre >= "é"
D3FF-	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)
D401-	C5 F3	CMP F3	le caractère pointé par TXTPTR est-il égal à celui de la table?
D403-	F0 E9	BEQ D3EE	si oui, on reboucle pour examiner le suivant
D405-	C8	INY	sinon, cherche la fin de ce mauvais mot-clé
D406-	B1 18	LDA (18),Y	dans la table (fin marquée par #00), ce qui évite de vérifier le mot-clé dans son entier
D408-	D0 FB	BNE D405	lorsque fin mot-clé trouvée,
D40A-	E8	INX	augmente X (n° d'ordre mot-clé SEDORIC)
D40B-	38	SEC	prépare addition de Y à 18/19 pour
D40C-	98	TYA	mise à jour de l'adresse du mot-clé suivant,
D40D-	65 18	ADC 18	Y = A = nombre de lettres du mot-clé
D40F-	85 18	STA 18	testé infructueusement (Super!)
D411-	90 D5	BCC D3E8	LL en 18 et report de
D413-	E6 19	INC 19	l'éventuelle retenue sur HH en 19
D415-	B0 D1	BCS D3E8	reboucle pour explorer ce nouveau mot-clé

Sous-programme "exécution"

D417-	8A	TXA	multiplie n° d'ordre mot-clé SEDORIC 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 SEDORIC (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 SEDORIC (#00 ou ":" ou paramètres de la commande)
D425-	4C 9E D3	JMP D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés et surtout **RTS** à l'adresse empilée juste avant

Sous-programme "non trouvé"

D428-	AD 1F C0	LDA C01F	entrée du sous-programme "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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
D435-	F0 12	BEQ D449	RTS si A = #00 (ce sous-programme sert probablement à d'autres fins) Ce RTS retourne au sous-programme 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)

Sous-programme "sortie vers LOAD"

D44A-	A9 00	LDA #00	initialise A = #00 pour commande LOAD
D44C-	4C F9 DF	JMP DFF9	(caractérise passage par le sous-programme "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 premier 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'adresse 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 SEDORIC, 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"

NB2: 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 sous-programme "non trouvé" de l'interpréteur (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 le sous-programme "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** 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 premier caractère), l'extension par défaut sera utilisée.

Il en est de même pour un **nom_de_fichier_ambigu**, mais les caractères autorisés comptent en plus * et ?, en outre un nom_de_fichier_ambigu peut être omis (exemple DIR) ou se réduire à la seule lettre indiquant le drive à utiliser (exemple DIR A).

Certains groupes de caractères, correspondant à un mot-clé BASIC, peuvent avoir été codés et ont été remplacés par le token correspondant (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 sous-programme "non trouvé" de l'interpréteur (b7 de F4 à 1), le nom_de_fichier_non_ambigu (chargement direct) ou l'indication de changement de drive (d-) ne comportent pas de "", de même, lorsqu'un nom_de_fichier_ambigu est requis (C = 0) et que ce nom_de_fichier_ambigu est omis ou réduit à une lettre (de A à D), il n'y a pas de "". La fin du nom_de_fichier_non_ambigu ou du nom_de_fichier_ambigu 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 nom_de_fichier_non_ambigu ou aux nom_de_fichier_ambigu: ,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 un 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 nom_de_fichier_ambigu ou nom_de_fichier_ambigu 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 sous-programme "non-trouvé" de l'interpréteur SEDORIC (D428) (A = #00 et N = 0) (nom_de_fichier ou drive-) avec TXTPTR pointant sur le premier caractère de cette commande (nom_de_fichier_non_ambigu 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, ("nom_de_fichier_non_ambigu" 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 (+ indication. éventuelle PROT)

D45F-	A2 0B	LDX #0B	X = 11 soit index pour 12 copies (de 11 à 0)
D461-	A9 20	LDA #20	code ASCII de l'espace
D463-	85 F3	STA F3	#20 -> F3 (pour éviter erreur n° 3 prématurée)
D465-	9D 29 C0	STA C029,X	
D468-	CA	DEX	rempli d'espaces nnnnnnnnneee de BUFNOM
D469-	10 FA	BPL D465	
D46B-	28	PLP	lorsque X = #FF, récupère indicateurs dont C et N
D46C-	10 62	BPL D4D0	si N = 0 branche en D4D0, c'est à dire pour toute entrée venant du sous-programme "non trouvé" de l'interpréteur SEDORIC: chargement direct (nom_de_fichier_non_ambigu sans "") ou changement de drive actif (lettre de A à D suivie d'un "-")
D46E-	B0 3B	BCS D4AB	branche si nom_de_fichier_non_ambigu

Analyse si nom_de_fichier_ambigu 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) nom_de_fichier_ambigu omis

D470-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
D473-	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 nom_de_fichier_ambigu omis (par exemple DIR)
D475-	A9 0C	LDA #0C	l'analyse du nom_de_fichier_ambigu proprement dit est terminée
D477-	85 F2	STA F2	F2 = 12 (pointeur de fin de zone nom + extension)
D479-	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 premier "?" est écrit en C128 au lieu de C029! De plus au retour Z = 0 car le dernier DEX entraîne X = #0B (non nul).

D47C-	F0 03	BEQ D481	branche en D481 si retourne avec Z = 1 (bogue?)
D47E-	4C 03 D5	JMP D503	sinon valide drive et termine (jump forcé)
D481-	C9 2C	CMP #2C	est-ce une ","? (nom_de_fichier_ambigu omis suivi de paramètre)
D483-	F0 F0	BEQ D475	si oui reboucle en D475 (rempli de "?" et sort)
D485-	C9 C3	CMP #C3	est-ce le token TO? (COPY omis TO ...)
D487-	F0 EC	BEQ D475	si oui reboucle en D475 (rempli de "?" et sort)

2) nom_de_fichier_ambigu réduit à une lettre (drive)

D489-	38	SEC	prépare soustraction (rappel C = 0 au début sous-programme)
D48A-	E9 41	SBC #41	convertit première lettre en n° de drive (A = 0 etc.)
D48C-	A8	TAY	copie ce n° de drive potentiel en Y
D48D-	C9 04	CMP #04	teste si 0 à 3 (A à D)
D48F-	B0 1A	BCS D4AB	A >= 4, ce n'est pas une indication de drive
D491-	20 98 D3	JSR D398	s'il s'agit d'une lettre de A à D, XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
D494-	F0 08	BEQ D49E	branche si fin d'instruction (et avec C = 1)
D496-	C9 C3	CMP #C3	est-ce le token TO? (par ex COPY B TO ...)
D498-	F0 04	BEQ D49E	si oui, continue en D49E avec C = 1
D49A-	C9 2C	CMP #2C	est-ce une virgule? (cf COPY ... TO B ,C)
D49C-	D0 05	BNE D4A3	non, branche en D4A3 (oui, continue avec C = 1)
D49E-	8C 28 C0	STY C028	n° de drive -> C028 et rebouclage forcé en D475
D4A1-	B0 D2	BCS D475	(rempli nnnnnnnnneee de "?" et fin analyse nom_de_fichier_ambigu)

Non, ce n'était pas un nom_de_fichier_ambigu omis ou réduit
(C'est donc un "nom_de_fichier_ambigu" réel)

D4A3-	A5 E9	LDA E9	
D4A5-	D0 02	BNE D4A9	décrémente TXTPTR pour annuler le JSR XCRGET
D4A7-	C6 EA	DEC EA	(pointe à nouveau sur le premier caractère du nom_de_fichier_ambigu)
D4A9-	C6 E9	DEC E9	

Evalue la chaîne nom_de_fichier_non_ambigu ou nom_de_fichier_ambigu
(place sa longueur dans F3 et son adresse dans TXTPTR)

D4AB-	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 "":
D4AE-	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 premier 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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés. 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 première 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 première 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 le sous-programme "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 incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
D4F2-	20 98 D3	JSR D398	(avance au troisième 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 sous-programme "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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

Analyse du nom_de_fichier_non_ambigu ou nom_de_fichier_ambigu 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 correspondant.

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 (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
D514-	24 F6	BIT F6	teste b7 de F6 (à 0 par défaut si l'extension n'est pas trouvée)
D516-	30 12	BMI D52A	si b7 = 1 (extension déjà trouvée), continue en D52A
D518-	C9 2E	CMP #2E	le caractère lu est-il un "." (cherche extension)
D51A-	D0 0E	BNE D52A	sinon, continue en D52A
D51C-	66 F6	ROR F6	si oui (extension trouvée), C = 1 -> b7 de F6
D51E-	E0 0A	CPX #0A	X pointe-t-il plus loin que le caractère n°9 ?
D520-	B0 1B	BCS D53D	si oui, "INVALID_FILE_NAME_ERROR" X = 9 position maximale du "."
D522-	A9 0C	LDA #0C	sinon, OK, mais il faut mettre à jour F2 qui
D524-	85 F2	STA F2	doit viser la fin de zone extension (F2 = 12)
D526-	A2 08	LDX #08	X = #08 pour pointer sur dernier caractère du nom
D528-	D0 15	BNE D53F	branchement forcé en D53F

Recherche d'une éventuelle virgule précédant un paramètre
(et marquant la fin du nom_de_fichier_non_ambigu en l'absence de "")

D52A-	C9 2C	CMP #2C	le caractère lu est-il une ","? (début de paramètre)
D52C-	D0 06	BNE D534	sinon, continue en D534 (recherche token etc)
D52E-	24 F4	BIT F4	si "," teste b7 de F4 (0 si le sous-programme "" non trouvé)
D530-	10 27	BPL D559	si 0 OK, branche en D559 (car implique sans "")
D532-	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")
D534-	20 67 D5	JSR D567	remplace tout token par le mot-clé correspondant, remplace "*" par des "?" et vérifie si les caractères sont autorisés (A-Z, 0-9)
D537-	9D 29 C0	STA C029,X	écrit le caractère A dans BUFNOM selon pointeur X
D53A-	98	TYA	récupère le dernier caractère lu Y dans A
D53B-	E4 F2	CPX F2	F2 = pointeur de fin de zone nom ou extension
D53D-	B0 6D	BCS D5AC	si X >=F2 "INVALID_FILE_NAME_ERROR" (nom trop long)
D53F-	C6 F3	DEC F3	décrémente le nombre de caractères à analyser
D541-	F0 10	BEQ D553	s'il ne reste plus rien, vérifie l'extension et termine
D543-	E8	INX	sinon augmente position du pointeur dans BUFNOM
D544-	24 F5	BIT F5	teste b7 de F5 (à 1 si un token est en cours)
D546-	30 CC	BMI D514	si b7=1 reboucle pour analyse "2 ème" caractère lu
D548-	20 98 D3	JSR D398	si b7=0 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
D54B-	D0 C7	BNE D514	si pas fin instruction, reboucle analyse caractère lu
D54D-	24 F4	BIT F4	si fin instruction teste F4 (b7=0 "" non trouvé)
D54F-	10 08	BPL D559	s'il est nul, vérifie extension et termine
D551-	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 premier caractère de l'extension
(Si pas valide remplace les 3 caractères par l'extension par défaut et termine)

D553-	68	PLA	
D554-	85 EA	STA EA	
D556-	68	PLA	restaure TXTPTR
D557-	85 E9	STA E9	
D559-	AD 32 C0	LDA C032	lit le premier caractère de l'extension dans BUFNOM
D55C-	C9 20	CMP #20	est-ce un espace?
D55E-	D0 A3	BNE D503	sinon, c'est fini, sortie générale
D560-	A2 00	LDX #00	si oui, X = #00 (indexe extension par défaut)
D562-	20 4A D3	JSR D34A	lit "COM" dans table CCF7, le copie dans BUFNOM
D565-	F0 9C	BEQ D503	revient avec Z = 1: c'est fini, sortie générale

Token, *, ? lettres de A à Z, chiffres de 0 à 9

Remplace tout token par le mot-clé correspondant, remplace tout "*" par des "?" et vérifie si tous les caractères sont autorisés (?, A-Z et 0-9)

D567-	24 F5	BIT F5	teste b7 de F5 (à 1 si token en cours)
D569-	30 24	BMI D58F	1 = il y a encore des caractères du mot-clé à lire
D56B-	A8	TAY	caractère lu A -> Y et b7 de A -> N (à 1 si token)
D56C-	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 + extension

Recherche du mot-clé BASIC

D56E-	85 F5	STA F5	sauve A (qui contient donc un token) dans F5
D570-	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)
D572-	85 24	STA 24	A -> 24 (compteur pour trouver le bon mot-clé)
D574-	A9 E9	LDA #E9	table C0E9 en ROM contient la liste des mots-
D576-	A0 C0	LDY #C0	clés, chacun se terminant par un ASCII + #80
D578-	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 négatif. C'est la seule sortie de ce sous-programme, 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 négatif (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 sous-programme lit les caractères 2 par 2 (pour réduire les accès à la ROM). Mais seul le premier 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 deuxième des 2 n'est pas exploité (c'est la première lettre du token suivant), car le flag b7 de F5 est immédiatement baissé (pas de token en cours). Si le premier des 2 n'est pas le dernier, le deuxième 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 premier caractère
D591-	E6 16	INC 16	
D593-	D0 02	BNE D597	incrémente 16/17 (vise premier 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 (deuxième 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	récupère 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 caractère, 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 correspondant 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 SEDORIC D'USAGE GÉNÉRAL

XROM Exécute à partir de la RAM une routine ROM

Le 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

```
D5D8- 85 0C     STA 0C
D5DA- 84 0D     STY 0D      sauve AY en 0C/0D
D5DC- 08        PHP
D5DD- 68        PLA
D5DE- 85 27     STA 27      sauve P en 27
D5E0- 18        CLC
D5E1- 68        PLA         prend l'adresse pour RTS sur la pile
D5E2- 85 0E     STA 0E      (adresse pointant après le JSR XROM)
D5E4- 69 04     ADC #04
D5E6- A8        TAY         y ajoute 4 pour sauter les DATA
D5E7- 68        PLA         et empile la nouvelle adresse de retour
D5E8- 85 0F     STA 0F
D5EA- 69 00     ADC #00     l'adresse d'origine (celle des DATA)
D5EC- 48        PHA         est gardée en 0E/0F
D5ED- 98        TYA
D5EE- 48        PHA
D5EF- A0 01     LDY #01
D5F1- AD 24 C0  LDA C024    si ROM V1.0 alors A = #00 et Y = #01 (octets DATA 1 et 2)
D5F4- 10 02     BPL D5F8    si ROM V1.1 alors A = #80 et Y = #03 (octets DATA 3 et 4)
D5F6- A0 03     LDY #03
D5F8- B1 0E     LDA (0E),Y
D5FA- 8D F0 04  STA 04F0    lit l'adresse du sous-programme ROM à exécuter
D5FD- C8        INY         et la place en 04F0/04F1 (EXEVEC)
D5FE- B1 0E     LDA (0E),Y
D600- 8D F1 04  STA 04F1
D603- A4 0D     LDY 0D
D605- A5 27     LDA 27
D607- 48        PHA         restaure P, A et Y et exécute le sous-programme ROM
D608- A5 0C     LDA 0C      dont le RTS utilisera l'adresse empilée
D60A- 28        PLP         pointant sur la suite du programme appelant
D60B- 4C 71 04  JMP 0471
```

Convertit n° lecteur en lettre et l'affiche

```
D60E- 18        CLC         Retenue mise à zéro pour addition
D60F- 69 41     ADC #41     A = A + #41 (convertit n° lecteur en lettre)
D611- 50 17     BVC D62A    continue en XAFCAR (affiche le caractère ASCII contenu dans A)
```

XAFHEX Affiche en hexadécimal le contenu de A

```
D613- 48        PHA         sauve A sur la pile
```

```
D614-  4A          LSR          A est de la forme xy (de 00 à FF)
D615-  4A          LSR          élimine les 4 bits de poids faible
D616-  4A          LSR          reste 0x
D617-  4A          LSR
D618-  20 1E D6    JSR D61E     convertit en caractère 0 à 9 ou A à F et affiche
D61B-  68          PLA          récupère A
D61C-  29 0F       AND #0F      élimine les 4 bits de poids fort, reste 0y
D61E-  C9 0A       CMP #0A      est-ce un nombre de 0 à 9?
D620-  90 02       BCC D624     si oui (C = 0), branche en D624
D622-  69 06       ADC #06      sinon (C = 1), A = A + 6 + C (#0A devient #11 etc)
D624-  18          CLC          prépare addition suivante
D625-  69 30       ADC #30      A = A + #30 (#0A du début devient #41 = "A")
D627-  2C A9 20    BIT 20A9     continue en D62A (XAFCAR affiche le caractère ASCII contenu dans A)
D628-  A9 20       LDA #20      code ASCII de l'espace
```

XAFCAR affiche comme un caractère ASCII le contenu de A

```
D62A-  C9 0D       CMP #0D      le caractère est-il un CR?
D62C-  D0 06       BNE D634     sinon, continue en D20E (affiche le caractère présent dans A)
D62E-  A9 00       LDA #00      si oui, remet à zéro la position de
D630-  85 30       STA 30       curseur sur la ligne (écran ou imprimante)
D632-  A9 0D       LDA #0D      puis reprend A = CR et
D634-  4C 0E D2    JMP D20E     continue en D20E (affiche le caractère présent dans A)
```

XAFSTR affiche chaîne terminée par 0 d'adresse AY

```
D637-  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 maximale de chaîne 256 caractères)
```

Affiche "_DISC_IN_DRIVE_"lettre du lecteur actif"AND_PRESS_'RETURN'"
puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)

```
D648-  A2 14       LDX #14      indexe "_DISC_IN_DRIVE_"
D64A-  20 6C D3    JSR D36C     affiche (X+1) ème 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+1) ème 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 correspondant, 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	où se trouve un simple RTS
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
D6B8-  85 E9       STA E9
D6BA-  84 EA       STY EA    remet en place TXTPTR
D6BC-  AD 1F C0    LDA C01F
D6BF-  AC 20 C0    LDY C020  valeur du pointeur de tampon clavier
D6C2-  8D 21 C0    STA C021
D6C5-  8C 22 C0    STY C022  remet en place pointeur de tampon clavier
D6C8-  60          RTS
```

Affiche l'erreur, réinitialise la pile et retourne au "Ready"

```
D6C9-  20 0A D3    JSR D30A  JSR EDE0/ROM autorise IRQ (gestion clavier/curseur)
D6CC-  AE FD 04    LDX 04FD  variable ERROR (n° de l'erreur)
D6CF-  E0 04       CPX #04   teste si c'est l'erreur n°#04
D6D1-  D0 33       BNE D706  sinon, continue directement en D706
D6D3-  A2 00       LDX #00   si oui, indexe "LFCRTRACK:"
D6D5-  20 6C D3    JSR D36C  affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
D6D8-  AD 01 C0    LDA C001  PISTE
D6DB-  20 13 D6    JSR D613  XAFHEX affiche en hexadécimal le contenu de A
D6DE-  AD 05 C0    LDA C005  type d'erreur (b5 à 0 = "_WRITE_FAULT_", à 1 = "_READ_FAULT_")
D6E1-  29 F0       AND #F0   1111 0000 force à 0 les 4 bits faibles
D6E3-  49 F0       EOR #F0   1111 0000 inverse les 4 bits forts
D6E5-  F0 14       BEQ D6FB  continue en D6FB si le résultat est nul
D6E7-  A2 01       LDX #01   sinon, indexe "_SECTOR:"
D6E9-  20 6C D3    JSR D36C  affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
D6EC-  AD 02 C0    LDA C002  SECTEUR
D6EF-  20 13 D6    JSR D613  XAFHEX affiche en hexadécimal le contenu de A
D6F2-  A2 03       LDX #03   pour message n°4 "_READ_FAULT_"
D6F4-  AD 05 C0    LDA C005  type d'erreur (b5 à 0 = "_WRITE_FAULT", à 1 = "_READ_FAULT_")
D6F7-  29 20       AND #20   0010 0000 force à zéro tous les bits sauf b5
D6F9-  F0 02       BEQ D6FD  continue en D6FD si le résultat est nul
D6FB-  A2 02       LDX #02   indexe "_WRITE FAULT_"
D6FD-  20 6C D3    JSR D36C  affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
D700-  AD 17 C0    LDA C017  n° de l'I/O error
D703-  20 13 D6    JSR D613  XAFHEX affiche en hexadécimal le contenu de A
D706-  AE FD 04    LDX 04FD  reprend la variable ERROR (n° de l'erreur)
D709-  CA          DEX       et la décrémente
D70A-  20 06 D2    JSR D206  CBF0/ROM va à la ligne
```

D70D-	A9 3F	LDA #3F	A = "?"
D70F-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
D712-	E0 1A	CPX #1A	teste si X >= #1A
D714-	B0 05	BCS D71B	si oui, saute les deux instructions suivantes
D716-	20 72 D3	JSR D372	affiche (X+1) ème message de zone CDBF terminé par "caractère + 128"
D719-	30 20	BMI D73B	suite forcée en D73B (b7 dernier caractère à 1)
D71B-	E0 31	CPX #31	teste si X < #31 (les erreurs utilisateurs peuvent avoir un n° compris entre #32 (50) et #FF (255))
D71D-	90 15	BCC D734	(il y a ici une bogue, #32 aurait été mieux!) si oui, continue en D734
D71F-	A2 10	LDX #10	sinon, indexe "USER_" (même pour l'erreur n°49!)
D721-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
D724-	AD FD 04	LDA 04FD	reprend la variable ERROR (n° de l'erreur)
D727-	A0 00	LDY #00	
D729-	8C 4C C0	STY C04C	DEFAFF code ASCII devant les nombres décimaux
D72C-	A2 01	LDX #01	pour afficher sur 3 digits n° erreur utilisateur
D72E-	20 58 D7	JSR D758	affichage en décimal d'un nombre AY
D731-	4C 3B D7	JMP D73B	suite forcée en D73B
D734-	8A	TXA	
D735-	E9 19	SBC #19	calcule X = X - #19
D737-	AA	TAX	
D738-	20 5C D3	JSR D35C	affiche le (X+1) ème message d'erreur externe terminé par un C+128 (adresse zone en C00D/C00E)
D73B-	4C 78 D1	JMP D178	C496/ROM réinitialise la pile, affiche " ERROR" et va au "Ready"

XCURON rend le curseur visible (= vidéo inverse)

D73E-	38	SEC	C = 1 ira dans b0 de 026A (pour CURSEUR ON)
D73F-	24 18	BIT 18	et continue en D741

XCUROFF cache le curseur (= vidéo normale)

D740-	18	CLC	C = 0 ira dans b0 de 026A (pour CURSEUR OFF)
D741-	08	PHP	sauvegarde les indicateurs dont C
D742-	4E 6A 02	LSR 026A	éjecte le b0 en décalant tous les bits à droite
D745-	28	PLP	récupère les indicateurs dont C
D746-	2E 6A 02	ROL 026A	récupère C dans b0 en décalant tous les bits à gauche
D749-	A9 01	LDA #01	effectue 0000 0001 ET registre 026A
D74B-	4C 2A D3	JMP D32A	sous-programme 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

Affichage en décimal sur X + 2 digits d'un nombre AY (entrée générale)

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 correspondant. 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 première soustraction du premier 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 (première 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	récupère 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 soustraction 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

XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si 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)
D7B0-	60	RTS	ou avec C = 0 (joker trouvé, mais autorisé)
D7B1-	26 F2	ROL F2	récupère C = 0 ou 1 selon entrée
D7B3-	90 FB	BCC D7B0	simple RTS si C = 0 (jokers autorisés)
D7B5-	A2 05	LDX #05	pour "WILDCARD(S)_NOT_ALLOWED_ERROR"
D7B7-	2C A2 01	BIT 01A2	et traite cette erreur en D67E
D7B8-	A2 01	LDX #01	pour "DRIVE_NOT_IN_LINE_ERROR"
D7BA-	4C 7E D6	JMP D67E	et traite cette erreur en D67E

Vérifie si drive demandé est "on line" et le valide "actif", si non génère une erreur

D7BD-	AC 28 C0	LDY C028	premier octet de BUFNOM = n° du lecteur demandé

Vérifie si le drive Y est "on line", si oui le valide "actif", si non génère une erreur

D7C0-	8C 00 C0	STY C000	on le met dans DRIVE (n° du lecteur actif)
D7C3-	B9 39 C0	LDA C039,Y	vérifie dans TABDRV que ce lecteur est "on line"
D7C6-	F0 F0	BEQ D7B8	si #00, "DRIVE_NOT_IN_LINE_ERROR"
D7C8-	60	RTS	si lecteur est "on line" retourne

Recherche et met à jour les 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

D7C9-	A2 0E	LDX #0E	X = #0E et continue en D7EF pour RA
D7CB-	2C A2 10	BIT 10A2	
D7CC-	A2 10	LDX #10	X = #10 et continue en D7EF pour RX
D7CE-	2C A2 12	BIT 12A2	
D7CF-	A2 12	LDX #12	X = #12 et continue en D7EF pour RY
D7D1-	2C A2 14	BIT 14A2	
D7D2-	A2 14	LDX #14	X = #14 et continue en D7EF pour RP
D7D4-	2C A2 16	BIT 16A2	
D7D5-	A2 16	LDX #16	X = #16 et continue en D7EF pour EF
D7D7-	2C A2 06	BIT 06A2	

Place dans la variable OM le n° du mode de sortie

D7D8-	A2 06	LDX #06	X = #06 et continue en D7EF pour OM
D7DA-	2C A2 04	BIT 04A2	
D7DB-	A2 04	LDX #04	X = #04 et continue en D7EF pour IN
D7DD-	2C A2 00	BIT 00A2	

Place dans la variable EN le n° de l'erreur A

D7DE-	A2 00	LDX #00	X = #00 et continue en D7EF pour EN
D7E0-	2C A2 0A	BIT 0AA2	
D7E1-	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)

D7E3-	2C A2 1E	BIT 1EA2	
D7E4-	A2 1E	LDX #1E	X = #1E et continue en D7EF pour CX
D7E6-	2C A2 20	BIT 20A2	
D7E7-	A2 20	LDX #20	X = #20 et continue en D7EF pour CY
D7E9-	2C A2 22	BIT 22A2	
D7EA-	A2 22	LDX #22	X = #22 et continue en D7EF pour FP
D7EC-	2C A2 24	BIT 24A2	
D7ED-	A2 24	LDX #24	X = #24 et continue en D7EF pour FS
			Les variables EL, SK, ST, ED, EX, EO sont indexées par Y:
D7EF-	A0 00	LDY #00	Y = #00 et continue en D803 pour toutes ces dernières
D7F1-	2C A2 02	BIT 02A2	

Place dans la variable EL le n° de ligne de l'erreur AY

D7F2-	A0 02	LDY #02	Y = #02 et continue en D803 pour EL
D7F4-	2C A2 08	BIT 08A2	
D7F5-	A0 08	LDY #08	Y = #08 et continue en D803 pour SK
D7F7-	2C A2 18	BIT 18A2	
D7F8-	A0 18	LDY #18	Y = #18 et continue en D803 pour ST

D7FA-	2C A2 1A	BIT 1AA2	
D7FB-	A0 1A	LDY #1A	Y = #1A et continue en D803 pour ED
D7FD-	2C A2 1C	BIT 1CA2	
D7FE-	A0 1C	LDY #1C	Y = #1C et continue en D803 pour EX
D800-	2C A2 0C	BIT 0CA2	
D801-	A0 0C	LDY #0C	Y = #0C et continue en D803 pour EO (bogue: semble inutilisée)
D803-	85 F2	STA F2	sauve A en F2 (LL de valeur de la variable)
D805-	BD 94 CD	LDA CD94,X	lit deux caractères
D808-	85 B4	STA B4	dans la table des variables
D80A-	BD 95 CD	LDA CD95,X	et les place en B4/B5
D80D-	85 B5	STA B5	(caractères significatifs d'une variable)
D80F-	98	TYA	Y précédemment 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 sous-programme 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 sous-programme, 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 LA ROUTINE EB78 EN ROM)

Afin de pouvoir traiter les touches de fonctions (voir le manuel SEDORIC page 54, 55 et 102), la gestion des touches a été modifiée. Sous SEDORIC, le vecteur JMP EB78 (en 023B/023D) est remplacé par JMP 045B. En 045B, SEDORIC passe sur RAM overlay, exécute la routine **D845** (XKEY prend un caractère au clavier)(incluant l'ancien EB78/ROM) et revient sur ROM.

Que fait donc le sous-programme **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 correspondant. Le sous-programme EB78 met ce code ASCII dans A, force N à 1 et remet 02DF à 0. Lorsqu'aucune touche n'a été pressée, le sous-programme EB78 retourne avec N = 0. X et Y ne sont pas touchés.

Que fait de plus le sous-programme **D845/RAM overlay** (XKEY prend un caractère au clavier)? 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 sous-programme 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 sous-programme 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 sous-programme 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 RAM overlay selon adresse en 16/17

D81C-	E6 16	INC 16	
D81E-	D0 02	BNE D822	incrémente l'adresse de lecture en 16/17
D820-	E6 17	INC 17	
D822-	A0 00	LDY #00	index Y = 0 pour lecture à l'adresse en 16/17
D824-	2C 48 C0	BIT C048	teste b6 de C048 (si 0, c'est une commande SEDORIC donc lecture en RAM overlay; si 1, commande BASIC donc lecture en ROM)
D827-	50 03	BVC D82C	si RAM overlay visée, saute la ligne suivante
D829-	4C 53 04	JMP 0453	si ROM visée, passe par page 4 (bascule sur ROM, effectue un LDA (16),Y puis bascule sur RAM overlay et RTS au point d'appel du sous-programme D81C)
D82C-	B1 16	LDA (16),Y	A = contenu cellule dont l'adresse est en 16/17
D82E-	F0 3F	BEQ D86F	branche si A = 0 (fin d'un mot-clé SEDORIC)
D830-	10 3F	BPL D871	simple RTS si le b7 de l'octet lu A est à 0
D832-	2C 48 C0	BIT C048	si le b7 de l'octet lu A est à 1, cela marque soit la fin d'une commande re-definissable ou pré-definie, soit un token BASIC incorporé dans un mot-clé SEDORIC, 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 re-definissable ou pré-definie écrite en toutes lettres, sans token ou à 1 si c'est un token BASIC incorporé dans un mot-clé SEDORIC).

D835-	10 38	BPL D86F	si commande re-definie ou pré-definie, continue en D86F
D837-	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
D839-	60	RTS	puis retourne

Teste si touche correspondant à n° de colonne et de ligne a été pressée

D83A-	8D 00 03	STA 0300	place le n° de ligne en 0300 (port B)
D83D-	A9 08	LDA #08	0000 1000 = masque pour tester b3 (réponse)
D83F-	2D 00 03	AND 0300	donne A = 0000 x000 avec x = 1 si touche correspondante
D842-	60	RTS	pressée, soit A = #08 et Z = 0

XLKEY prend un caractère au clavier, entrée appelée par LINPUT

D843-	38	SEC	C = 1, entrée utilisée uniquement par LINPUT
D844-	24 18	BIT 18	(F2 = longueur spécifiée) et continue en D846

XKEY prend un caractère au clavier, entrée générale principale

D845-	18	CLC	C = 0, entrée utilisée dans tous les autres cas: appels dérivés de la ROM (vecteur 045B), ainsi que WINDOW, TYPE et BUILD.
D846-	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
D849-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
D84C-	08	PHP	sauve P et notamment N, flag de touche pressée
D84D-	8D 46 C0	STA C046	sauve A = code ASCII correspondant 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 fonction 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 sous-programme appelé depuis LINPUT

Vérification spéciale pour les appels de la ROM

Les sous-programmes 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 (maximum pour la 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 fonction 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	mise à zéro de C04B et de C048 qui sont utilisés:
D877-	8D 4B C0	STA C04B	- pour sauver première lettre d'un mot-clé SEDORIC
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 re-definissable (code #00 à #0F) ou pré-definie (code #10 à #1F). Sinon, il s'agit soit d'une commande SEDORIC (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 correspondante 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 relais 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 SEDORIC page 104).

```
D886-  20 3A D8   JSR D83A     teste si la touche FUNCT a été pressée
D889-  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

```
D88B-  AD 46 C0   LDA C046     sinon, pas besoin de gérer touche de fonction,
D88E-  AE 47 C0   LDX C047     récupère A = code ASCII et X = nombre de caractères
D891-  2C 3D C0   BIT C03D     force V et N selon MODCLA (b6=ACCENT b7=AZERTY)
D894-  10 D9      BPL D86F     si QWERTY (mode par défaut), termine en D86F
D896-  AD 08 02   LDA 0208     si AZERTY il faut intervertir certaines lettres
D899-  A2 05      LDX #05      compare le code de touche saisi avec ceux de la
D89B-  DD 41 CD   CMP CD41,X   table CD41 (6 codes pour ; M Z A W et Q)
D89E-  F0 0C      BEQ D8AC     code trouvé, branche en D8AC pour conversion
D8A0-  CA         DEX          code pas trouvé, décrémente index X et examine
D8A1-  10 F8      BPL D89B     le précédent, tant qu'il en reste (X >= 0)
D8A3-  AD 46 C0   LDA C046     rien trouvé, récupère A = code ASCII de touche
D8A6-  AE 47 C0   LDX C047     récupère X = nombre de caractères dans buffer
D8A9-  4C 6F D8   JMP D86F     sort en D86F: ce n'était alors pas une lettre "critique"
```

Conversion QWERTY / AZERTY

```
D8AC-  AD 08 02   LDA 0208     empile le code de la touche pressée (en 0208
D8AF-  48         PHA          se trouve le code des touches "normales")
D8B0-  BD 47 CD   LDA CD47,X   lit le code de touche correspondante en AZERTY
D8B3-  8D 08 02   STA 0208     et le met à la place du code touche pressée
D8B6-  20 1A D3   JSR D31A     JSR F4EF/ROM "Trouver le code ASCII"
D8B9-  AA         TAX          X = code ASCII correspondant au code de remplacement
D8BA-  68         PLA          récupère l'ancien code de touche
D8BB-  8D 08 02   STA 0208     et le remet en place
D8BE-  8A         TXA          A = code ASCII correspondant au code de remplacement
D8BF-  29 7F      AND #7F      masque 0111 111, force à 0 le b7 de A
D8C1-  10 E3      BPL D8A6     rebouclage forcé en D8A6 (récupère X et termine)
```

Teste si une touche SHIFT a aussi été pressée

D8C3-	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.
D8C5-	20 3A D8	JSR D83A	teste si la touche SHIFT DROIT a été pressée
D8C8-	D0 07	BNE D8D1	oui, branche en D8D1; non, teste l'autre shift
D8CA-	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
D8CC-	20 3A D8	JSR D83A	teste si la touche SHIFT GAUCHE a été pressée
D8CF-	F0 02	BEQ D8D3	sinon, saute la ligne qui suit (A = 0000 0000)
D8D1-	A9 40	LDA #40	si oui, A = 0100 0000 (FUNCT+SHIFT, met b6 à 1)
D8D3-	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

D8D6-	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 SEDORIC page 104):

Voici un tableau qui résume la correspondance entre les touches (dump du centre), les codes de touches correspondants (à gauche, valeurs de #80 à #BF) et les index de lecture dans la table **KEYDEF située en C800** pour un appui sur une touche + FUNCT ou sur une touche + FUNCT + SHIFT. Ces index de lecture sont la valeur à ajouter (de #00 à #7F) à l'adresse du début de la table #C800 pour trouver le code de fonction associé à cette combinaison de touches. Autrement dit, à chaque combinaison FUNCT+touche ou FUNCT+SHIFT+touche correspond un code de fonction choisit parmi 256. Cette correspondance est donnée par la table "KEYDEF" rappelée plus loin, qui liste les #80 codes correspondant aux #40 combinaisons FUNCT+touche et aux #40 combinaisons FUNCT+SHIFT+touche.

TABLE DES CODES DE TOUCHES

Codes pour touche seule	0 1 2 3 4 5 6 7 8 9 A B C D E F	Index de lecture dans la table KEYDEF pour: touche + FUNCT	touche + FUNCT+SHIFT
#80 à #8F	7 J M K _ U Y 8 N T 6 9 , I H L	#00 à #0F	#40 à #4F
#90 à #9F	5 R B ; . O G 0 V F 4 - ↑ P E /	#10 à #1F	#50 à #5F
#A0 à #AF	m m c m g f m s 1 e Z m ← d A r	#20 à #2F	#60 à #6F
#B0 à #BF	X Q 2 \ ↓] S m 3 D C ' → [W =	#30 à #3F	#70 à #7F

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT gauche, m touche manquante, r RETURN, s SHIFT droit, et _ SPACE. Exemple: le code de touche #80 correspond à la touche "7" et le code de touche #BF

à la touche "=". A chaque code de touche correspond un index de lecture dans la table KEYDEF. Mais certains codes (exemple #A0) ne correspondent à aucune touche.

Rappel de la table "KEYDEF":

Cette table est responsable de l'affectation de codes de fonctions (#00 à #FF) aux codes de touches (#80 à #BF). Les codes de fonctions sont décrits en ANNEXE. Les codes de touches sont représentés, sur l'image d'un clavier, dans le manuel SEDORIC, page 104. L'ordre de ces codes de touche est des plus mystérieux, puisque le premier (#80) est attribué à la touche "7", le second (#81) à la touche "J", le troisième (#82) à la touche "M", etc jusqu'au dernier (#BF) à la touche "=".

La table "KEYDEF" est une suite de 128 (#80) codes de fonctions, choisis parmi les 256 possibles et rangés dans le désordre de C800 à C87F. En fait ils sont rangés dans l'ordre des codes de touches: d'abord "7", puis "J", puis "M", ... jusqu'à "=". L'index d'entrée dans cette table va de #00 à #3F pour les combinaisons FUNCT+touche et de #40 à #7F pour les combinaisons FUNCT+SHIFT+touche. Il faut ajouter respectivement #80 ou #40 à cet index pour obtenir le code de fonction correspondant.

La table KEYDEF a été complètement revue pour intégrer des commandes SEDORIC dont l'utilisation était impossibles dans les versions précédentes. Ceci a été rendu possible grâce à la correction de la bogue de la routine "Prendre un caractère au clavier" en D907.

TABLE KEYDEF

FUNCT+touche (commandes SEDORIC)

	Codes de fonctions (voir en ANNEXE)		Touches (de #80 à #BF)	
C800-	**07 45 57 4B**	**00 18 07 08**	7 J M K	_ U Y 8
C808-	**59 7B 06 09**	00 **42 41 52**	N T 6 9	, I H L
C810-	05 **66 25** 00	00 **5B 27 00**	5 R B ;	. O G 0
C818-	**1B 3F 04 0A**	**00 5E 3D 0D**	V F 4 -	↑ P E /
C820-	00 00 00 00	00 00 00 00	m m c m	g f m s
C828-	**01 00 08** 00	**00 00 22** FF	1 e Z m	← d A r
C830-	**6D 62 02 0C**	**00 0F 72** 00	X Q 2 \	↓] S m
C838-	**03 31 29 00**	**00 0E 1E 0B**	3 D C '	→ [W =

FUNCT+SHIFT+touche (commandes BASIC)

	Codes de fonctions (voir en ANNEXE)		Touches (de #80 à #BF)	
C840-	**17 B2** A8 **F1**	00 **8C A6 18**	7 J M K	_ U Y 8
C848-	**90 C9 16 19**	00 92 **A2 BC**	N T 6 9	, I H L
C850-	**15** 9C **CA** 00	00 D2 9B **10**	5 R B ;	. O G 0
C858-	**EB 8D 14 1A**	**00 87 C8 1D**	V F 4 -	↑ P E /
C860-	00 00 00 00	00 00 00 00	m m c m	g f m s
C868-	**11** 00 **A5** 00	00 00 **D1** FF	1 e Z m	← d A r
C870-	**A4 9A 12 1C**	00 **1F CB** 00	X Q 2 \	↓] S m
C878-	**13 91** ED 00	**00 1E B5 1B**	3 D C '	→ [W =

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT gauche, m touche manquante (certains numéros de code ne correspondent à aucune touche), r RETURN, s SHIFT droit, et _ SPACE.

Ce nouveau tableau permet d'accéder aux fonctions **BASIC** avec **FUNCT+SHIFT+touche** et aux commandes **SEDORIC** avec **FUNCT+touche**. Et ceci en respectant autant que possible les initiales. Les commandes SEDORIC sans n° (UNPROT, USING, VISUHIRES, VUSER, WIDTH, WINDOW et !RESTORE) sont maintenant accessibles.

UNE TABLE KEYDEF UN PEU PLUS PRATIQUE

J'ai reclassé ce tableau de correspondance dans l'autre sens: A chaque touche (dans l'ordre alphabétique, de A à Z) correspond une commande SEDORIC (avec appui simultané sur FUNCT) ou une commande BASIC (avec appui simultané sur FUNCT+SHIFT). Les codes de fonction sont aussi indiqués.

Touche	FUNCT (SEDORIC)	Code n°	FUNCT+SHIFT (BASIC)	Token n°
A/Q	AZERTY	#22	AND	#D1
B	BACKUP	#25	NOT	#CA
C	COPY	#29	CHR$	#ED
D	**DIR**	**#31**	DATA	#91
E	ESAVE	#3D	ELSE	#C8
F	FIELD	#3F	FOR	#8D
G	CHANGE	#27	GOSUB	#9B
H	HCUR	#41	HIRES	#A2
I	INIT	#42	INPUT	#92
J	JUMP	#45	INK	#B2
K	KEYSAVE	#4B	KEY$	#F1
L	LINPUT	#52	**LIST**	**#BC**
M/?	MOVE	#57	MUSIC	#A8
N	NUM	#59	NEXT	#90
O	OLD	#5B	OR	#D2
P	PROT	#5E	PLOT	#87
Q/A	QWERTY	#62	RESTORE	#9A
R	RENUM	#66	RETURN	#9C
S	SAVEU	#72	STEP	#CB
T	TYPE	#7B	THEN	#C9
U	UNPROT	#18	UNTIL	#8C
V	VISUHIRES	#1B	VAL	#EB
W/Z	WINDOW	#1E	WAIT	#B5
X	SEEK	#6D	EXPLODE	#A4
Y	PAPER0:INK7	#07	PING	#A6
Z/W	CALL#F8D0+CR	#08	ZAP	#A5

Exemples: FUNCT+"D" affiche "DIR" (Code n°49 = #31, voir en ANNEXE), tandis que FUNCT+SHIFT+"L" affiche "LIST" (Token n° 188 = #BC, voir ANNEXE). Les touches A/Q, M/?, Q/A, W/Z et Z/W ont une double étiquette. Ceci correspond aux claviers AZERTY/QWERTY. La touche ;/M n'est pas utilisée, il en est de même pour les touches ' , . et /. qui toutes ont reçu le code #00. Il est possible de re-definir ces touches à l'aide de la commande KEYDEF.

Dans les versions précédentes de SEDORIC, le sous-programme traitant des codes correspondant aux mots-clés SEDORIC était complètement bogué en D9A0 et ne marchait pas. Il est clair qu'au dernier moment les codes SEDORIC avaient été remplacés par des #00 dans le tableau C800. Résultat: les commandes SEDORIC n'étaient pas accessibles!

Analyse du type de commande

Les codes de commandes (page 102 du manuel SEDORIC) sont de 4 types (voir en ANNEXE):

n°#00 à #1F: commandes re-definissables (#00 à #0F) et pré-definies (#10 à #1F)

n°#20 à #7F: Mots-clés SEDORIC. On ne peut accéder aux derniers mots-clés (voir page 103 du manuel SEDORIC). Ce n° est ramené de #00 à #5F en soustrayant #20 pour obtenir le n° d'ordre des mots-clés SEDORIC

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).

D8D8-	AA	TAX	X est un index pour
D8D9-	BD 00 C8	LDA C800,X	lire la table des codes de fonctions
D8DC-	A8	TAY	copie code lu dans Y (pour tester si #FE ou #FF)
D8DD-	C8	INY	incrémente Y pour tester si c'était #FF (car #FF + 1 = #00 et C = 1)
D8DE-	D0 03	BNE D8E3	sinon, saute la ligne suivante
D8E0-	4C 63 D9	JMP D963	si oui, continue vers le sous-programme NUM AUTO
D8E3-	C8	INY	incrémente Y pour tester si c'était #FE (car #FE + 2 = #00)
D8E4-	F0 6C	BEQ D952	si oui, continue vers le sous-programme DEL TAMPON
D8E6-	C9 20	CMP #20	met C à 0, si A < #20 ou C à 1 si A >= #20
D8E8-	6A	ROR	C->b7...b0->C (c'est à dire force b7 selon C)
D8E9-	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 RAM overlay (b6 à 0 pour la table des commandes re-definissables en C880 ou celle des commandes pré-definies en C980 ou encore celle des mots-clés SEDORIC en C9DE).
D8EC-	2A	ROL	C<-b7...b0<-C (restaure les valeurs initiales)
D8ED-	30 04	BMI D8F3	saute soustraction suivante si A>=#80 (token BASIC)
D8EF-	90 02	BCC D8F3	saute la soustraction suivante si A<#20 (commande re-definie ou pré-definie)
D8F1-	E9 20	SBC #20	restent les codes SEDORIC (#20 à #7F) dont on calcule le n° d'ordre (#00 à #5F) en retirant #20
D8F3-	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)
D8F5-	AA	TAX	copie dans X le n° d'ordre pour servir
D8F6-	A9 E9	LDA #E9	d'index plus loin

```
D8F8-  A0 C0     LDY #C0     AY = C0E9 (table des mots-clés BASIC en ROM)
D8FA-  2C 48 C0  BIT C048    positionne N et V selon b7 et b6 (type de code)
D8FD-  70 29     BVS D928    continue en D928 si b6 à 1 (token BASIC)
D8FF-  30 06     BMI D907    continue en D907 si b7 à 1 (mots-clés SEDORIC)
D901-  A9 7F     LDA #7F     restent codes de #00 à #1F (commandes re-definies et pré-definies)
D903-  A0 C8     LDY #C8     AY = C87F (table des commandes re-definies et pré-definies)
D905-  D0 21     BNE D928    branchement forcé en D928
```

Cherche la première lettre du mot-clé SEDORIC de n° d'ordre X

En D90A, correction de la bogue "LOVE" qui affectait la première lettre des commandes SEDORIC. Ainsi la commande MOVE devenait LOVE, c'est joli non? La mauvaise gestion de l'index X était la cause de la bogue! La place nécessaire pour corriger cette bogue étant insuffisante (de 2 octets), il a fallu faire un détour par un sous-programme supplémentaire en EA30 (voir à cet endroit). Par rapport à la version 1.0, les 20 octets modifiés ci-dessous sont indiqués en gras.

```
D907-  A5 F2     LDA F2
D909-  48        PHA         sauve F2 sur la pile
D90A-  20 30 EA  JSR EA30    détour pour insérer l'instruction manquante (manque de place ici)
D90D-  A0 00     LDY #00     index pour lire dans la table des mots-clés
D90F-  B9 BD CB  LDA CBBD,Y  lit un n° d'ordre
D912-  E8        INX         code ASCII suivant
D913-  C8        INY
D914-  C8        INY         Y = Y + 4
D915-  C8        INY         Y vise le n° d'ordre suivant
D916-  C8        INY
D917-  C5 F2     CMP F2      n° d'ordre lu < n° d'ordre cherché ?
D919-  90 F4     BCC D90F    si oui, pas dépassé, on reboucle en D90F
D91B-  8E 4B C0  STX C04B    sinon, trouvé ou dépassé, sauve code ASCII
D91E-  A6 F2     LDX F2      récupère X (n° d'ordre du mot-clé SEDORIC)
D920-  CA        DEX         nouvelle instruction: gestion de l'index déboguée!
D921-  68        PLA
D922-  85 F2     STA F2      et récupère F2 (longueur chaîne LINPUT)
D924-  A9 DD     LDA #DD     AY = C9DD (table des mots-clés SEDORIC)
D926-  A0 C9     LDY #C9     pour rechercher suit chaîne (terminée par #00) etc...
```

Cherche la fin de la X ème chaîne dans tableau AY

Exemple: si le n° d'ordre de la chaîne à chercher est X = 2, la première chaîne ayant le n° 0, il faut rechercher la fin de la deuxième chaîne pour être au début de la troisième qui est donc la bonne (n°2).

```
D928-  85 16     STA 16      adresse du tableau qu'il faudra explorer
D92A-  84 17     STY 17      (C0E9/ROM, C9DD/RAM overlay ou C87F/RAM overlay)
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 premier 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	premier 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 première lettre (cas d'un mot-clé SEDORIC)
D948-	38	SEC	force à 1 le b7 de C049, c'est à dire du
D949-	6E 49 C0	ROR C049	flag "code de fonction 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: sous-programme 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 du sous-programme 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: sous-programme 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

NOTES PERSONNELLES