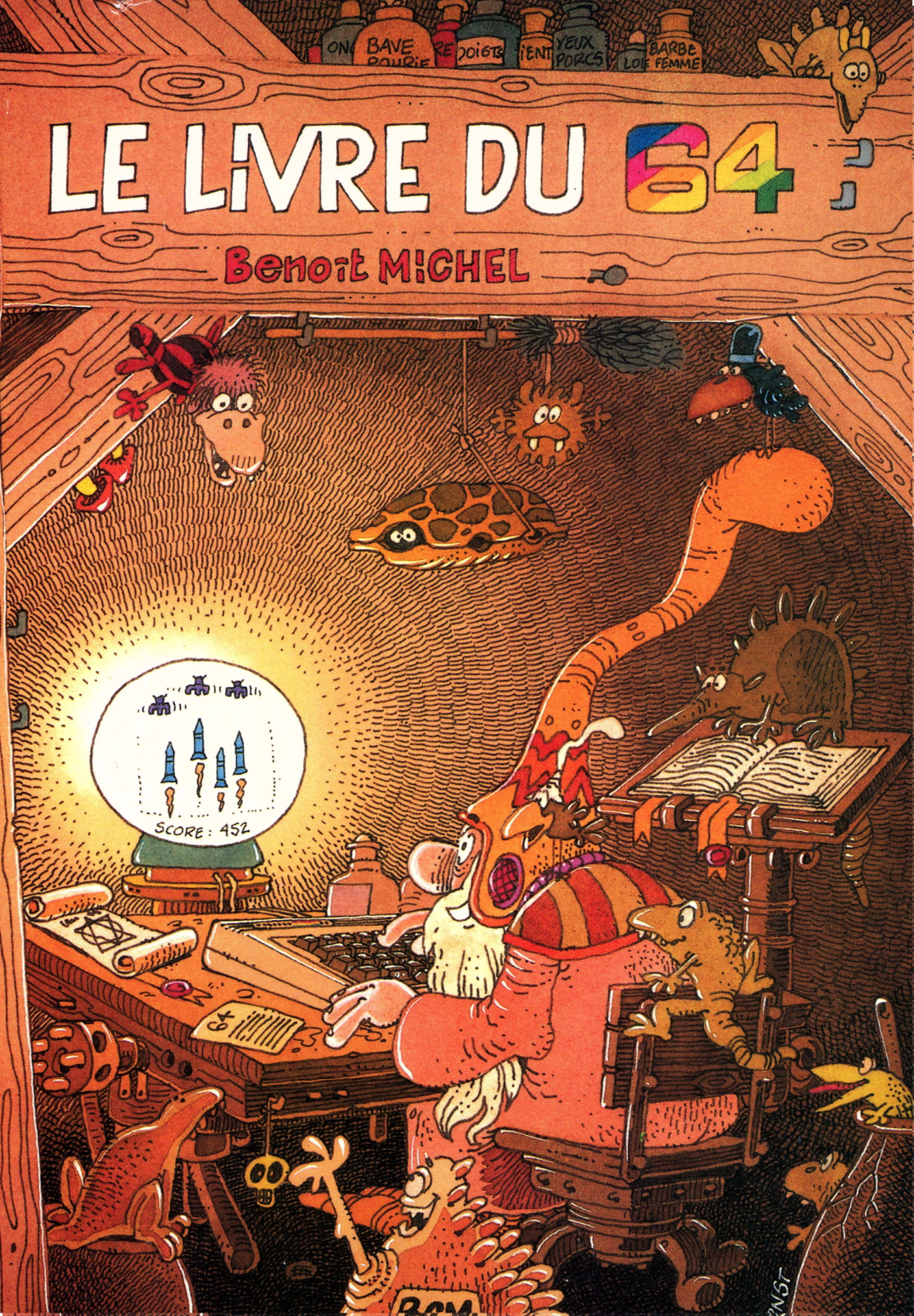
LE LIVRE DU 64

Benoit MICHEL



LE LIVRE DU 64

par
B. Michel

BCM
1984

AUTRES OUVRAGES EDITES CHEZ B.C.M. s.c.

Le Livre du VIC (épuisé)	par B. MICHEL
Le Livre du 64	par B. MICHEL
Le Livre du MSX	par D. MARTIN
Les Dessous du SPECTRAVIDEO	par D. MARTIN
Le Livre du MS/PC-DOS	par F. PIETTE
Le Livre de l'AMSTRAD	par D. MARTIN et P. JADOUL
Ecrire en dBASE	par C. MICHEL

DISQUETTES DISTRIBUEES PAR B.C.M. s.c.

Ces disquettes contiennent les programmes des livres concernés

Le Disque du 64 (format 1541)
Le Disque du MS/PC-DOS
Le Disque de dBASE
Le Disque de l'AMSTRAD
Le Disque du MSX

EN PREPARATION :

Le Livre de l'AMSTRAD Tome 2 (Disques et périphériques)
Le Livre de l'ATARI 520 ST
Le Livre de l'AMIGA
Le Livre du C

3ème édition Mai 1986
Imprimé en Belgique par BON TON - 4600 CHENEE
Dépot légal : D/1984/3827/2

ISBN : 2-87111001-80

Distribué par
PSI Diffusion, B.P. 86, F - 77042 LAGNY s/MARNE cedex FRANCE
PSI BENELUX, 17, rue du Doyenné, B - 1050 BRUXELLES,BELGIQUE

I N T R O D U C T I O N

Cet ouvrage est consacré à la programmation du COMMODORE 64.

Fruit d'une étroite collaboration avec les membres du club liégeois d'informatique MICRORDI, il a pour but de combler les nombreuses lacunes qui subsistent dans les informations connues à ce jour sur le CBM 64. Dès qu'un utilisateur assimile les rudiments du langage BASIC, il se sent frustré car l'accès aux splendides possibilités graphiques et autres lui semblent toujours aussi inacessibles.

Si certains passages de ce livre peuvent sembler ardus, les concepts qu'ils développent restent toujours aisément utilisables grâce aux nombreux exemples prêts à fonctionner.

Les spécificités les plus méconnues du CBM 64 sont ici développées au grand jour. Citons entre autres le mélange de plusieurs modes graphiques dans un même écran, l'emploi des touches de fonctions, l'impression sur papier d'images graphiques, la reprogrammation du clavier en AZERTY, et bien d'autres encore ....

Pour maîtriser totalement son ordinateur, il ne suffit jamais de s'équiper du mode d'emploi. Cet ouvrage rencontre une partie des demandes de l'utilisateur débutant. Cependant, le programmeur décidé à se lancer dans l'emploi du langage ASSEMBLEUR ou du langage FORTH devra se procurer les manuels correspondants. On trouvera en fin de volume une bibliographie couvrant ces sujets.

Bon voyage à bord du livre du 64 !!

CHAPITRE 1

LE SYSTEME

Le CBM 64 est un micro-ordinateur dans la plus pure tradition COMMODORE. Il conserve en effet de ses ancêtres, le PET et le VIC, la structure générale. Le langage est toujours le langage BASIC MICROSOFT version II et le noyau du systeme (le KERNAL) est toujours compatible avec celui des PET et VIC. Si certaines fonctions ont été ajoutées, aucune n'a été supprimée ni déplacée.

Le CBM 64 est un ordinateur BASIC. DèS l'allumage de la machine, le message 'READY' bien connu est affiché. Comme pour de nombreuses autres machines, le langage BASIC et le système opératoire -Le KERNAL en terminologie COMMODORE- sont intimement mêlés.

Regardons de plus près les programes internes du CBM 64: deux mémoires de 8 K (8192 octets) contiennent tout le logiciel nécessaire au fonctionnement de la machine, soit 9 K de BASIC et 7 K de KERNAL. L'entièreté du BASIC et une grande partie du KERNAL sont identiques à la version existant dans le VIC à l'exception de la gestion directe des périphériques, légèrement différents. L'écran par exemple, possède 25 lignes de 40 caractères au lieu de 22 seulement dans le VIC.

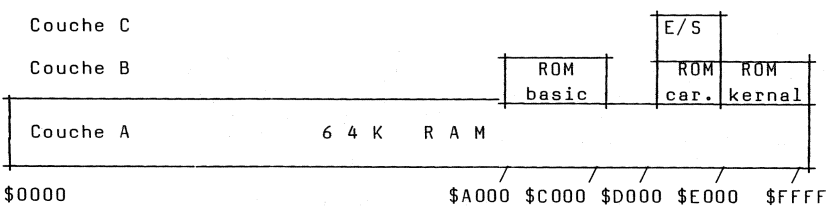
L'architecture interne du CBM 64 est, elle, totalement originale. Il contient en effet 64 K de mémoire RAM (mémoire que l'on peut lire et écrire à volonté, mais qui s'efface si le courant est coupé) et 20 K de mémoire ROM (READ ONLY MEMORY ou mémoire que l'on peut lire mais pas modifier ni effacer).

Ajoutons à cela les périphériques qui occupent une zone de 4K et le CBM 64 se retrouve propriétaire de 88 K octets à contrôler. Or, un microprocesseur classique, comme le 6510 qui équipe le CBM 64 n'adresse que 64 K octets simultanément.

Comment procède-t-il alors? La solution du problème passe par l'emploi d'une série d'interrupteurs commandés par le microprocesseur et qui mettent en service, tour à tour,l'une ou l'autre mémoire.

Nous allons, théoriquement du moins, construire un CBM 64. Prenons huit circuits 4164. Ce sont des mémoires de 65536 x 1 bit. Mises en parallèle, elles nous donnent 64 K octets de mémoire (1 K = 2 à la puissance 10,soit 1024 octets).

Ajoutons un microprocesseur 6510, un contrôleur d'écran VIC-II et une mémoire morte ROM de 4 K octets qui contient le générateur de caractères. Cette dernière mémoire ne contient donc pas un programme, mais bien le dessin de chacun des 512 caractères différents possibles, à l'usage du contrôleur d'écran.



En l'état actuel, notre CBM 64 ne possède aucun programme en mémoire, et aucun moyen d'en transférer un depuis une source extérieure.

Les périphériques qui permettront la communication avec le monde extérieur, nous allons tout simplement les déposer par dessus la mémoire RAM. De façon à ce que le microprocesseur, regardant par au dessus son espace mémoire, voit les périphériques et non plus la mémoire à cet endroit. Ici, nous faisons intervenir le jeu d'interrupteurs dont nous parlions ci-dessus. Ils permettront de faire passer soit la mémoire, soit les périphériques sur le dessus de la pile.

L'électronique digitale d'un ordinateur travaille en "tout ou rien" et le système binaire basé sur l'emploi de 2 valeurs 0 ou 1, s'y impose donc. Les adresses d'une mémoire sont transmises par 16 fils qui conduisent (1) ou ne conduisent pas (0) du courant électrique. C'est pourquoi les adresses de différents blocs

de mémoire commencent toujours (très souvent en tout cas) à des valeurs qui sont des puissances entières de 2. Les puissances de 2 exprimées en décimal sont: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768,65536.

Très souvent, les électroniciens et informaticiens travaillent avec des nombres en base 16, en hexadécimal. La raison en paraît évidente si l'on regarde la table des puissances de deux en base 16 : 1, 2, 4, 8, 10, 20, 40, 80, 100, 200, 400, 800, 1000, 2000, 4000, 8000, 10000.

La notation HEXA comporte les chiffres 0 à 9 suivis des lettres A, B, C, D, E, F qui portent les valeurs décimales de 10,11, 12, 13, 14 et 15.

Nous repérerons tout au long de ce livre les valeurs HEXA en les faisant précéder du symbole "$".

On trouvera en annexe le programme 'HEXA' qui vous familiarisera avec cette notation.

LES PERIPHERIQUES

Les périphériques vont être déposés par dessus la mémoire RAM aux adresses $D000 et suivantes, jusqu'en $DFFF.

Ils occupent donc $E000 - $D000 =$1000 octets soit 4096 octets ou 4 K. Il y a dans le CBM 64 4 périphériques principaux: le VIC-II, contrôleur d'écran, le SID ou SOUND INTERFACE DEVICE, contrôleur sonore et deux CIA ou COMPLEX INTERFACE ADAPTER. Ces derniers sont essentiellement constitués de portes d'entrée/sortie bidirectionelles. Chaque CIA possède deux portes paralèles à 8 bits, quelques signaux de contrôle pour gérer les mouvements par les portes, deux minuteries et une horloge.

Le contrôleur d'écran VIC-II occupe 48 adresses de $D000 à $D02F. Ce sont ces adresses qui permettent le dialogue entre le VIC-II et le microprocesseur. Mais quand le VIC-II affiche une image, il va chercher en mémoire RAM les informations nécessaires à créer l'image: Le numéro du caractère et la couleur du caractère. Le dessin du caractère, lui, provient d'une zone mémoire qui peut être en RAM et c'est ce qu'on appelle les caractères programmables. Mais le plus souvent, comme c'est le cas à l'allumage de la machine, le dessin provient du générateur de caractères, cette mémoire ROM que nous avons évoquée précédemment. Cette ROM est à l'adresse $D000.

Mais, direz-vous, nous venons de mettre à cette adresse nos périphériques !!

Eh bien, reconstruisons notre empilage: Sur la RAM, nous posons le générateur de caractères (c'est la couche B de la figure) et par dessus, les périphériques(la couche C). Le microprocesseur, sans basculer d'interrupteur, peut ainsi voir ses périphériques et manipuler le clavier, la cassette, etc...

Il n'a pas souvent besoin d'accéder au générateur de caractères. Le VIC-II, quant à lui, accède le générateur 200.000 fois par seconde et pour lui le problème de basculer l'interrupteur n'existe pas. Le VIC-II, en effet n'accède jamais aux périphériques.

Il existe cependant un cas où il est bien intéressant de pouvoir lire le contenu du générateur de caractères. Pour créer un nouveau jeu de caractères, on peut l'introduire en mémoire à partir d'un périphérique; mais lorsqu'on veut seulement en modifier une partie, la solution la plus simple est de recopier la ROM générateur de caractères en RAM. Et pour cela, il nous faut manipuler les interrupteurs. Les interrupteurs de pagination de la mémoire (en anglais: BANKING) sont situés à l'adresse 1 de la mémoire. Cette adresse est en fait celle d'une porte d'entrée/sortie à 8 bits qui fait partie intégrante du microprocesseur lui-même.

Basculons donc l'interrupteur:

```
POKE 1,51
```

Le nom officiel de cet interrupteur est '**CHAREN**' (de l'anglais 'character enable' ou autorisation des caractères). Nous constatons que notre ordinateur refuse dès lors tout service: Il est comme mort. Eteindre et rallumer la machine est la seule solution. Que s'est-il passé ?

Notre micro-ordinateur, même quand il affiche 'READY' et attend une commande, effectue un programme. Ce programme s'appelle "**ROUTINE D'INTERRUPTION**" et se produit tous les soixantièmes de seconde. Son rôle est de lire le clavier et de mettre à jour le compteur d'horloge **TI$**, entre autres.

Cette routine fait de nombreux appels aux périphériques. Or, en basculant l'interrupteur, nous avons mis hors service les périphériques sans prévenir le microprocesseur.

Il nous faut donc écrire un programme pour mettre en service le générateur de caractères sans se servir des périphériques. En mmode direct, cela est impossible, le clavier n'étant jamais accessible en même temps que le générateur de caractères.

Pour prévenir le microprocesseur que la routine d'interruption ne doit pas fonctionner, il faut introduire le programme suivant :

```
10 POKE 56333,127    :REM Pas d'interruptions
20 POKE 1,51         :REM Enlève les périphériques
30  GOSUB 1000       :REM
40 POKE1,55         :REM Rétablit les périphériques
50 POKE 56333,129    :REM rétablit les interruptions
60 END
```

Le sous-programme que nous allons écrire à la ligne 1000 peut maintenant lire le générateur de caractères.

```
1000 X = PEEK(53288)
1999 RETURN
```

Une chose que nous avons négligé jusqu'ici est toutefois primordiale: Il faut un programme à un ordinateur pour fonctionner. Ce programme, c'est le KERNAL, qui, avec le BASIC, donne au CBM 64 sa configuration habituelle. Ces programmes sont répartis dans les deux mémoires ROM placées dans la couche B par dessus la RAM aux adresses $A000 à $BFFF et $E000 à $FFFF.

Ici encore le microprocesseur voit normalement les mémoires ROM mais peut contrôler leur présence grâce à deux interrupteurs. On pourrait croire que chaque interrupteur contrôle la présence d'une des deux mémoires mortes, or ce n'est pas le cas.

Nous avons vu que le bit 2 de l'adresse 1 (valeur 2^2 = 4) contrôlait les périphériques. Le bit 1 de l'adresse 1 (valeur 2^1 = 2) contrôle la présence des deux mémoire BASIC et KERNAL. Le nom officiel de cet interrupteur est **HIMEM**. Mettre ce bit à 0 enlève de la vue du microprocesseur les deux mémoires.

Comme précédemment, à cause des interruptions, l'ordinateur tombe mort et il faut l'éteindre et le rallumer. Mais ici, plus question de s'en tirer avec un programme écrit en BASIC, car il n'y a plus de BASIC du tout! Donc, seuls les programmes entièrement écrits en langage machine peuvent se permettre d'ôter ainsi les mémoires mortes du CBM 64.

L'autre interrupteur est représenté par le bit 0 de l'adresse 1 (valeur 2^0 = 1). Bien plus utilisé, cet interrupteur ne met hors service que la ROM $A000 qui contient 80 % du BASIC. Son nom officiel est **LOMEM** .

Il devient flagrant que l'interrupteur HIMEM n'aurait aucune utilité en ne masquant que la ROM KERNAL. La ROM $A000 seule n'est en effet d'aucune utilité sans l'autre, puisque le BASIC est en fait réparti dans les deux mémoires.

Enlever la ROM $A000 et conserver le KERNAL est par contre très intéressant: il nous reste un ordinateur avec son système d'exploitation et 56 K de mémoire, ce qui est idéal pour des programmes comme le traitement de texte, les tableaux de calcul "....**CALC**" ou d'autres langages comme le **FORTH**, le **C**,le **LOGO**, etc...

Nous avons vu que la présence de la mémoire ROM devant la RAM empêche de lire cette dernière. Mais une mémoire ROM ne peut, par définition, être modifiée. Donc écrire dans une mémoire ROM est impossible et inutile; c'est pourquoi l'électronique du CBM 64 dérive les instructions d'écriture - les POKE - adressées à une mémoire ROM vers la mémoire RAM située derrière elle.

Cette particularité très intéressante est véritablement unique parmi les micro-ordinateurs usuels. Elle est la clé de nombreuses possibilités de modification du BASIC ou du KERNAL. Il suffit de lire tous les octets des deux ROM et de les réécrire à la même adresse, puis de mettre les deux mémoires ROM hors service. Il nous reste à ce moment un BASIC et un KERNAL en RAM et donc aisément modifiables à souhait.

On peut voir des exemples de ceci avec les programmes **'BASIC-FR'**et **'AZERTY'** en annexe. On trouvera également en fin de volume le programme **'COPIEROM'** écrit en langage machine dans un souci de rapidité d'exécution qui recopie les 16 K de ROM en RAM et effectue le basculement des interrupteurs nécessaires.

A titre d'exemple, nous allons corriger une erreur de l'interpréteur BASIC. La commande :

```
PRINT ASC("")
```

donne comme résultat le message suivant :

```
?ILLEGAL QUANTITY ERROR
```

Dans tout autre BASIC, la réponse aurait été '0'. Corrigeons cette erreur,et pour cela rendons le BASIC modifiable en le recopiant en RAM avec 'COPIEROM' ou par le programme BASIC suivant, qui est équivalent:

```
10 FOR I = 40960 TO 49151
20 POKEI,PEEK(I) : NEXT I :REM IL Y EN A POUR UNE MINUTE
```

Nous avons donc copié la ROM $A000 en RAM.

```
30 POKE 46991,5 :REM AU LIEU DE 8 DANS LA ROM ORIGINALE
```

Ceci corrige l'interprétation du mot BASIC **ASC**.

```
40 POKE 1,54
```

Ceci met hors service la ROM BASIC, et nous pouvons maintenant essayer notre BASIC modifié.

Pour expérimenter ce domaine, il est à conseiller de reprendre le programme 'COPIEROM', d'y ajouter les POKE de correction et de les sauver sur cassette ou sur disquette avant d'essayer la modification. Le risque de ne pas récupérer un ordinateur en bon état de fonctionnement est élevé.

Il faut alors couper l'alimentation ou pousser sur le bouton de RESET (installez le vite, si ce n'est pas encore fait !) pour récupérer la situation. Le programme étant,même dans ce cas souvent perdu, on comprend tout de suite l'avantage de l'avoir sauvegardé avant de l'essayer!

Une zone intéressante à expérimenter est le début de la ROM BASIC en $A000 qui contient les messages et les mots-clé du BASIC: Voir l'exemple du programme BASIC-FR. Voici un autre exemple:

POKE 41122,65 transforme le mot-clé 'LIST' en 'LAST' !!

Bien sûr, LIST ne donne plus qu'un bien connu "SYNTAX ERROR"
On peut toujours récupérer le BASIC d'origine avec un :

POKE 1,55

ou en enfonçant simultanément STOP et RESTORE.

Signalons ici l'existence de l'excellent ouvrage "**WHAT'S REALLY INSIDE THE COMMODORE 64**" publié par DATACAP et qui donne le texte complet des ROM BASIC et KERNAL.

Il existe encore deux autres interrupteurs de pagination nommés EXROM et GAME. Ces deux interrupteurs ne sont pas accessibles par programme: il faut connecter physiquement un interrupteur entre les broches 8 (**GAME**), 9 (**EXROM**) et la broche 1 (MASSE) du connecteur d'extension ROM. Ces interrupteurs font toujours partie des cartouches d'extension ROM comme FORTH ou SIMON'S BASIC.

Normalement, GAME et EXROM valent 1. Quand ils sont tous deux à 0, les 8 K de mémoire RAM de $8000 à $AFFF sont recouverts par une mémoire ROM externe présente dans la cartouche d'extension si le reste de la mémoire est normal. (LOMEM = 1 et HIMEM = 1). Si la ROM BASIC est hors service (LOMEM = 0), et que la cartouche externe bascule à 0 les interrupteurs GAME et EXROM, les adresses $A000 à $BFFF sont occupées par une mémoire ROM dans la cartouche externe. Ceci permet à des cartouches extérieures qui n'utilisent pas de BASIC de voler 16 K d'espace mémoire contigu au 64.

La dernière configuration possible est celle où GAME = 0 et EXROM = 1. Elle enlève de l'espace d'adressage toutes les mémoires RAM et ROM des adresses $1000 à $FFFF, ne laissant en service que 4 K de mémoire RAM dans le CBM 64 et 16 K de mémoire ROM dans la cartouche de $8000 à $9FFF et de $E000 à $FFFF.

Ceci transforme le CBM 64 en une machine identique a l'"**ULTIMAX**" le jeu vidéo de bas de gamme que COMMODORE a étudié puis abandonné au cours de l'année 1982. Cette configuration ne possède aucun intérêt pratique. Le but recherché était de permettre l'usage des cartouches de jeu de l'ULTIMAX sur le CBM 64.

Voici un tableau récapitulatif des situations possibles:

	N.CONFIGURATION	1	2	3	4	5	6	7	8	9	10
	INTERRUPTEURS				*	*				*	
	POKE 1,	55	53	54	52	52	55	54	55	53	X
	GAME	1	1	1	1	X	0	0	0	0	0
	EXROM	1	X	X	X	0	0	0	0	0	1
TAILLE DU BLOC	ADRESSES HEXA	NATURE DES MEMOIRES									
8 K	$E000 - $FFFF	ROM	RAM	ROM	RAM	RAM	ROM	ROM	ROM	RAM	CAR
4 K	$D000 - $DFFF	I/O	I/O	I/O	RAM	RAM	I/O	I/O	I/O	I/O	I/O
4 K	$C000 - $CFFF	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	NIL
8 K	$A000 - $BFFF	ROM	RAM	RAM	RAM	RAM	ROM	CAR	CAR	RAM	NIL
8 K	$8000 - $9FFF	RAM	RAM	RAM	RAM	RAM	CAR	RAM	CAR	RAM	CAR
16 K	$4000 - $7FFF	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	NIL
12 K	$1000 - $3FFF	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	NIL
4 K	$0000 - $0FFF	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM	RAM

RAM = Mémoire vive
ROM = Mémoire morte
CAR = Mémoire morte dans une cartouche externe
I/O = Zone des périphériques
NIL = RIEN DU TOUT

* Pour remplacer la zone I/O par la ROM générateur de caractères, il faut mettre à l'adresse 1 la valeur reprise au tableau à laquelle on aura soustrait la valeur 4. Ceci est toutefois impossible pour les configurations 4,5 et 9.

RESUME DES CONFIGURATIONS MEMOIRE DU CBM 64

CONFIGURATION 1 : état normal du CBM 64 à l'allumage sans cartouche d'extension. 38 K accessibles au BASIC.

CONFIGURATION 2 : Périphériques et 60K RAM . Il faut écrire ses propres routines d'entrée-sortie ou changer de configuration pour accéder au KERNAL.

CONFIGURATION 3 : 52 K RAM contigus, périphériques et ROM KERNAL.C'est la configuration utilisée par les langages autres que BASIC qui se chargent en mémoire RAM. Par exemple, le CP/M.

CONFIGURATION 4 : 64 K RAM contigus. Il faut bien entendu changer de configuration pour tout accès à un périphérique.

CONFIGURATION 5 : Comme ci-dessus mais dans le cas où une cartouche est présente.

CONFIGURATION 6 : Configuration où on conserve le BASIC et où l'on ajoute une cartouche d'extension BASIC comme SIMON'S BASIC par exemple.

CONFIGURATION 7 : 40 K RAM contigus et 8 K ROM dans une cartouche externe pour les applications n'ayant pas besoin de BASIC (cartouches de traitement de texte, par exemple).

CONFIGURATION 8 : 32 K RAM contigus et 16 K ROM en cartouche. Comme ci-dessus.

CONFIGURATION 9 : Comme la configuration 2, mais avec une cartouche externe; le générateur de caractères est inaccessible dans cette configuration.

CONFIGURATION 10 : C'est celle du jeu video ULTIMAX qui n'a jamais vu le jour.

LES PROGRAMMES INTERNES DU CBM 64

Les programmes présents d'office dans le CBM 64 se subdivisent en deux parties: le **BASIC** et le **KERNAL**.

Le BASIC ne contient aucune facilité particulière pour manipuler les fonctions graphiques ou sonores du CBM 64. Par contre, la liaison existante entre le BASIC et le système d'exploitation KERNAL est très intelligemment orchestrée. Les portes restent donc ouvertes à de nombreuses additions. Les programmes SIMON'S BASIC, SUPEREXPANDER, SUPER GRAPHIC 64 et autres ne se privent d'ailleurs pas de modifier les connexions BASIC-KERNAL.

Les portes ouvertes les plus connues sont les instructions SYS et USR qui donnent accès à des programmes écrits en langage

machine et extérieurs au BASIC. Les quatre autres possibilités d'altération du BASIC sont les suivantes:

- Modifier les vecteurs en page 3
- Modifier les routines d'interruptions
- Modifier la routine CHRGOT
- Recopier en RAM et modifier le BASIC ou le KERNAL

Voici qui introduit des notions qui nous sont étrangères. Le système d'exploitation KERNAL est un ensemble de programmes qui gèrent les périphériques: écrire un caractére à l'écran, lire le clavier, lire et écrire sur la cassette, sur la porte série IEEE pour les lecteurs de disquettes ou l'imprimante, sur la porte série RS232 pour la télécommunication, etc...

Chacun de ces programmes contient une instructions de saut (JMP), l'équivalent en langage machine du GOTO, vers une des 13 lignes de la table des vecteurs en page 3. Avant d'étudier le pourquoi de la chose, voyons le 'comment': Dès l'allumage de la machine, le KERNAL remplit la table des vecteurs avec la liste des adresses de ses 13 routines principales. A partir de là, chaque routine du KERNAL fait un petit détour en mémoire RAM à chaque exécution. Mais ces vecteurs étant en mémoire RAM, sont donc modifiables à volonté.

C'est ainsi que le programme "PRINTER" intercepte les vecteurs CLOSE, OPEN et CHROUT pour dévier la sortie normale de caractères vers une imprimante non prévue par le système.

Les routines d'interruptions sont au nombre de trois: **IRQ**, **BRK** et **NMI**. Ce sont de petits programmes très courts qui s'exécutent périodiquement lorsque surviennent certains évènements précis: IRQ s'exécute 60 fois par seconde et s'occupe du clavier, de l'horloge et du moteur de la cassette. BRK ne survient que si le microprocesseur exécute l'instruction $00 ,appellée aussi BREAK. Elle a pour rôle d'effacer l'écran et de remettre en service les pages mémoire, les modes graphiques et sonores dans l'état où ils sont quand on allume la machine.

NMI est une routine qui réagit le plus souvent comme BRK; c'est le cas quand on enfonce les touches STOP et RESTORE simultanément. Mais c'est aussi NMI qui s'occupe de la gestion des périphériques série IEEE,RS232 et CASSETTE.

La modification de ces routines est bien sûr la clé des programmes comme ARROW, par exemple. (ARROW est une cartouche ROM qui fait fonctionner la cassette à une vitesse sept fois supérieure à la normale, ajoute de nombreueses fonctions au BASIC, etc..)

La routine **CHRGOT** est la dernière clé qui ouvre une très large porte sur le fonctionnement du BASIC. Jusqu'ici, à part la

méthode de recopie et de modification directe des programmes en ROM, nous n'avons ouvert des portes que sur le KERNAL.

CHRGOT est une très courte routine qui est située en RAM aux adresses $0073 et suivantes. C'est la portion de l'interpréteur BASIC qui prend un caractère dans le texte BASIC et le transfère à l'interpréteur proprement dit. Le fait que cette routine est en mémoire vive est une nécessité car il s'agit d'un programme qui se modifie lui-même. En effet, il contient l'adresse du caractère à lire, il lit le caractère, puis il ajoute 1 à l'adresse du caractère à lire et se modifie ainsi lui-même.

Si bien qu'à l'appel suivant de CHRGOT, il renvoie le caractère suivant. Cette méthode a été choisie car c'est la plus rapide. C'est une des principales causes de la grande rapidité des interpréteurs BASIC COMMODORE. Quant à nous, nous y trouvons un avantage : CHRGOT est aisément modifiable.

On peut ainsi modifier CHRGOT pour intercepter de nouveaux mots-clé BASIC. CHRGOT étant modifiable à volonté (mais avec prudence),il est donc facile après avoir pris un caractère dans le texte BASIC, de vérifier s'il appartient à un nouveau mot-clé (un de ceux que l'on a rajouté et pas un de ceux du BASIC d'origine). Sinon, on continue normalement. Si le caractère ne désigne pas un mot-clé BASIC , mais bien un de ceux que nous avons ajouté au système, nous exécutons la tâche appropriée. Et le retour au BASIC normal ne se fait qu'avec le premier caractère suivant la commande spéciale.

La cartouche 'ARROW' fonctionne suivant ce principe: Les mots-clé spéciaux commencent tous par une flèche à gauche, ce qui rend le test ajouté à CHRGOT très simple et rapide. Un autre exemple est le programme "DOS SUPPORT" qui reconnaît les commandes simplifiées de la disquette. Les programmes qui modifient CHRGOT sont appellés "WEDGE" en anglais.

LE CBM 64 DU POINT DE VUE DE L'ELECTRONICIEN

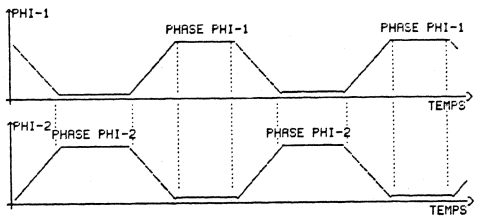
Le microprocesseur 6510 fait partie de la famille du 6502 et son fonctionnement est tout à fait similaire. Le rythme des opérations dans tout l'ordinateur dépend d'une horloge à quartz de précision rigoureuse. L'horloge oscille à la fréquence de 17,73 MHz (17 millions de fois par seconde) pour les CBM 64 européens, tandis qu'aux U.S.A., elle est de 14,31 MHz. La raison en est que cette fréquence est nécessaire au fonctionnement des téléviseurs qui servent d'écran. L'europe utilise le système **PAL** et les U.S.A. le système **NTSC**.

Ces fréquences étant trop élevées pour le microprocesseur, on les divise par 18 en système PAL et par 14 en système NTSC, ce qui nous donne les fréquences de fonctionnement de 1,0227 MHz aux U.S.A. contre 0,985 MHz en europe.

La différence de vitesse de 4% entre les deux types d'ordinateurs pose parfois des problèmes, entre autres à cause du positionnement différent des lutins (voir chapitre 4) qui en résulte, ainsi que des différences de timbre du générateur sonore obtenues entre les deux continents avec un programme identique.

La présence de cette introduction à l'électronique du CBM 64 se justifie par les conclusions de ce genre que l'on peut en tirer. D'autres problèmes vont par ailleurs se trouver éclairés d'un jour nouveau:

L'horloge maîtresse fonctionne donc à 1 MHz environ. Elle distribue à l'ensemble de l'ordinateur deux signaux PHI 1 et PHI 2, qui passent alternativement de 0 à 5 volts (on dit aussi: du niveau 0 au niveau 1).



Chaque fois que PHI 2 vaut 1, nous obtenons une période d'une durée de 500 nanosecondes que nous appellerons PHASE 2. De même, quand PHI 1 vaut 1, nous aurons la phase 1.

A chaque phase 2, le microprocesseur utilise les BUS (groupes de fils) d'adresses et de données pour effectuer ses programmes. L'autre moitié du temps, pendant les phases 1, le contrôleur d'écran VIC-II occupe les deux bus pour lire les informations dont il a besoin pour maintenir à jour l'image sur le téléviseur. La seconde tâche dont le VIC-II s'occupe se produit aussi durant les phases 1. Il s'agit du rafraîchissement des mémoires. Les mémoires 6164 de notre ordinateur ont la mémoire courte. Elles s'effacent en effet après deux millièmes de seconde seulement.

Heureusement, pour éviter ces amnésies indésirables, il existe une méthode fort simple. Il suffit de lire ces mémoires et elles retrouvent une nouvelle période de deux millisecondes de stabilité.

Si les mémoires du CBM 64 nous paraissent stables, c'est que le VIC-II s'occupe de les relire successivement sans relâche.

Jusqu'ici, pas de problèmes entre le microprocesseur et le VIC-II: A chacun sa phase.

Mais les choses se compliquent quand on apprend que, en plus du générateur de caractères et de la mémoire à rafraîchir, le VIC-II doit aussi, toutes les 8 lignes de balayage de l'écran, afficher 40 nouveaux caractères. Il lui faut donc effectuer 40 lectures de la mémoire d'écran toutes les 8x64 microsecondes (une ligne de balayage TV dure environ 64 microsecondes). Or, il est déja occupé lors de toutes les phases 1 disponibles.

Dernier problème: l'affichage des lutins nécessite aussi un accès à la mémoire à chaque ligne de balayage pour lire l'adresse du lutin et trois accès à la mémoire par ligne à afficher pour générer le dessin du lutin lui-même.

Pour effectuer ces accès supplémentaires, le VIC-II 'vole' des phases 2 au microprocesseur et lui interdit ainsi de temps en temps l'accès à la mémoire.

Pour l'utilisateur, cela signifie qu'il n'est pas possible de se fier à une boucle de comptage dans un programme pour connaître exactement la durée d'un évènement précis. Nous comprenons maintenant pourquoi l'écran est éteint pendant les accès à la cassette. Les utilisateurs de CBM 64 qui possèdent un lecteur de disquettes se sont probablement tous rendus compte des problèmes que peut poser la présence de lutins sur l'écran pendant les opérations de LOAD ou de SAVE sur disquette.

En conclusion, un bon conseil: Effacez toujours de l'écran les lutins avant de travailler avec un périphérique raccordé sur le bus série IEEE !!!

C H A P I T R E 2

LES PROGRAMMES INTERNES DU 64

L'INTERPRETEUR BASIC

A l'achat d'un CBM 64, un seul programme est fourni par COMMODORE: c'est l'interpréteur BASIC. C'est aussi et de loin, le plus utilisé de tous les programmes existant sur le CBM 64. A un point tel d'ailleurs que bien des utilisateurs de micro-ordinateur ne savent même pas que BASIC est un programme. On cite en général BASIC en tant que langage. Or aucun ordinateur, pas même le 64, ne peut comprendre ce langage. La langue maternelle du 6510, cerveau du CBM 64, est ce qu'on appelle le " langage machine ". L'interpréteur BASIC est le programme qui traduit les commandes BASIC et les convertit en ordres exécutables par le 6510. Le fait que le BASIC est avant tout un programme et non avant tout un langage est extrêmement important: c'est en connaissant bien le programme que l'on pourra efficacement utiliser le langage.

Les premiers langages de haut niveau comme FORTRAN étaient traduits (compilés) d'un bloc en langage machine avant d'être exécutés.

Les défauts de cette méthode sont nombreux : elle nécessite une très grosse mémoire et une procédure fort longue à manipuler. Avec BASIC, une nouvelle génération de langages a vu le jour: les langages interprétés, c'est-à-dire traduits et exécutés simultanément ; on traduit une instruction, on l'exécute puis on passe à la suivante et ainsi de suite. C'est peut être plus lent comme procédé mais c'est simple et efficace.

Le programme BASIC est remarquablement court au vu de son efficacité. Dans le 64, le BASIC occupe la mémoire ROM aux adresses $A000 à $C000 et $E000 à $E4EB, soit 9451 octets. Le noyau principal de l'interpréteur BASIC est ce qu'on appelle l'interpréteur de commande.

L'interpréteur de commande est un programme relativement court, qui exécute indéfiniment une boucle : attendre une commande au clavier de l'ordinateur, puis agir en conséquence, et recommencer sans fin. L'interpréteur de commande est actif dès l'allumage du 64, et en général, à tous les moments où l'écran affiche le message 'READY', qui signifie prêt à recevoir une commande. Détaillons les deux éléments de l'interpréteur de commande : Attente et Action.

L'ATTENTE

Une commande BASIC peut être simple (exemple : RUN) ou compliquée (exemple : FOR I=0 TO 60 : PRINT MID$ (TI$,4,2) : NEXTI). Il faut donc permettre au programmeur de voir ce qu'il écrit, de corriger une faute éventuelle avant de transmettre la commande à l'interpréteur BASIC. C'est pourquoi l'interpréteur de commande fait appel 60 fois par seconde à un programme de lecture de clavier, et parallèlement, à un programme appelé EDITEUR D'ECRAN (ces deux programmes font partie du KERNAL). Le premier met en mémoire dans le **tampon de clavier** les caractères successivement frappés au clavier, le second les recopie dans l'écran, sauf les caractères dits d'édition qui lui sont destinés : déplacements de curseur, insertion, effacement, changements de couleurs, etc... dont il tient compte immédiatement pour donner au message en cours de frappe l'aspect à l'écran désiré par le programmeur.

Un problème se pose évidemment : comment transmettre à l'interpréteur BASIC, les caractères spéciaux de curseur, codes couleur, etc..? C'est ici qu'intervient un des éléments les plus déroutants pour le débutant : le mode "GUILLEMETS". En effet, après avoir reçu un nombre impair de fois le caractère ", l'éditeur d'écran n'interprète pas pour son propre compte les caractères spéciaux, mais les transmet directement (sous forme d'un caractère graphique) à l'écran. Le mode "GUILLEMETS" est également activé après la frappe du caractère spécial "Insertion".

Ensuite, deux autres caractères sont reconnus de façon toute particulière par l'éditeur d'écran : le "RETURN" et le "SHIFT-RETURN". Ils signifient tous deux que la commande sur laquelle le curseur se trouve est terminée. Avec RETURN, la commande entrée au clavier est transmise à la phase ACTION de l'interpréteur de commande.Avec SHIFT-RETURN,la commande entrée est aussitôt oubliée et l'interpréteur de commande redémarre à zéro la phase ATTENTE.

Il existe de toute façon une autre méthode de terminer à tout moment l'action de l'interpréteur de commande, comme de l'interpréteur BASIC, ou de tout autre programme : la touche STOP ou la combinaison STOP-RESTORE, qui sont interprétées par les routines de décodage clavier et redémarrent le programme en cours à son début: en règle générale, il s'agit de l'interpréteur de commande BASIC.

L'ACTION : A la fin de chaque commande, le contrôle passe à la seconde partie de l'interpréteur de commande : la commande est là dans l'écran et maintenant il faut agir. Première phase de l'action : prendre le texte de la commande et le recopier dans une zone mémoire (TAMPON D'ENTREE) qui servira de bloc-notes pour les opérations suivantes. Ce tampon, en mémoire à l'adresse $0200, a une longueur de 88 caractères.

Cette longueur est donc la longueur limite d'une commande BASIC quelle qu'elle soit. Une fois le transfert exécuté, le vrai travail commence. La deuxième phase de l'action consiste à TASSER la commande pour en réduire la longueur. A cette fin, l'interpréteur de commande passe en revue le texte de la commande caractère par caractère et, s'il rencontre au début de la commande un ESPACE , il le supprime en redécalant tous les caractères d'une position vers la gauche.

De plus, par souci d'efficacité les MOTS-CLES BASIC qui forment l'ensemble du vocabulaire BASIC, sont encodés sous la forme d'un seul caractère. Il existe 76 mots-clés dans le CBM 64. Or, dans les parties de texte qui ne sont pas entre guillemets, il n'y a dans le texte d'une commande normalement aucun caractère sur fond inversé. Ceci réduit donc de moitié le nombre de caractères possibles dans ces zones et libère 128 codes (les caractères de 128 à 255) pour la représentation des mots-clés. S'il lit un caractère qui n'est ni alphabétique ni entre guillemets, l'interpréteur de commande commence tout d'abord par remettre à zéro le bit 7 du code caractère pour supprimer un éventuel fond inversé.

Ensuite s'il s'agit d'un caractère alphabétique, l'interpréteur recherche dans la table des mots-clés BASIC (en mémoire ROM en $A09E ou 41118), le premier mot-clé commençant par la même lettre. S'il en trouve un, il compare le caractère suivant du tampon d'entrée avec le caractère suivant du mot-clé. Ce processus est répété jusqu'à la découverte d'une différence ou jusqu'à la fin du mot-clé. (La fin d'un mot-clé se reconnaît parce que le bit 7 du dernier caractère de chaque mot-clé vaut 1 dans la table). Dans le cas d'une différence, l'interpréteur de commande continue la recherche dans la table jusqu'à la découverte du mot-clé correct ou jusqu'à la fin de la table.

Dans ce dernier cas, il en conclut que le caractère analysé ne fait pas partie d'un mot-clé. Ce peut être un nom de variable, par exemple. Si le caractère testé fait bien partie d'un mot-clé, l'interpréteur remplace tous les caractères du mot-clé par un seul caractère dont le bit 7 vaut 1 (code compacté ou TOKEN en anglais). La valeur de ce caractère est simple à trouver : il s'agit de la position du caractère dans la table (en comptant à partir de 0) à laquelle on ajoute $80 pour mettre le bit 7 à 1 et ne pas avoir de conflit avec un caractère alphanumérique. Comme il y a 76 mots-clés BASIC dans le CBM, les codes TOKEN sont donc compris entre 128 et 203 inclus (de $80 à $CB).

A la fin de cette opération de compactage sur tout le tampon d'entrée, la taille d'une commande est réduite à son optimum. La commande BASIC est maintenant sous sa forme utilisable. Une dernière constatation à propos de la méthode de recherche des mots-clés dans la table s'impose : l'interpréteur de commande détecte la fin d'un mot-clé BASIC en faisant la différence du code caractère dans le tampon d'entrée et dans la table en $A09E. Si la différence vaut $80, c'est que les codes caractères sont identiques, sauf les bits 7 qui sont différents .

Or il peut y avoir deux causes à cette différence : ou bien le bit 7 du caractère dans la table vaut 1 et c'est la fin du mot-clé dans la table, ou bien le bit 7 du caractère dans le tampon d'entrée vaut 1 et c'est parce que ce caractère a été frappé avec la touche SHIFT enfoncée. Ce peut ne pas être le dernier caractère du mot-clé dans la table. Ceci explique pourquoi l'interpréteur BASIC accepte que l'on frappe des abréviations aux mots-clés comme R, shift-U au lieu de RUN. Dans le cas où deux mots-clés commencent par les deux mêmes caractères (READ et RESTORE par exemple), l'abréviation en deux caractères donne une comparaison avec succès dès le premier des deux mots-clés rencontré dans la table (dans notre exemple, READ). L'abréviation du second peut donc s'obtenir en 3 ou 4 caractères comme R, E, shift-S, ou R, E, S, shift-T pour RESTORE.

La dernière phase de l'action de l'interpréteur de commande consiste à regarder dans le tampon d'entrée si le premier caractère est numérique ou non. S'il est numérique, il en déduit que la commande, commençant par un numéro, doit être considérée comme une ligne de programme à ranger en mémoire. Dans le cas contraire, il s'agit d'une commande à interpréter directement et le contrôle est transféré à l'interpréteur BASIC proprement dit. Avant d'analyser le fonctionnement de l'interpréteur BASIC proprement dit, il convient de bien connaître ce qu'il doit interpréter.

L'interpréteur n'a accès qu'à deux zones distinctes en mémoire : le tampon d'entrée, où il interprète une commande en mode direct puis rend le contrôle à l'interpréteur de commande et la zone texte où le corps d'un programme est enregistré. C'est l'encodage du texte BASIC dans cette zone que nous allons examiner maintenant.

Le KERNAL initialise le CBM 64 avec la ZONE TEXTE débutant en $0801 . Détaillons le contenu de la mémoire dans laquelle ont été introduites les lignes BASIC suivantes :

```
10 PRINT"A"
20 GOTO 10
```

Dans ce cas, le CBM 64 contient en mémoire les valeurs suivantes :

Adresse	Valeur	
$0800	$00	Le caractère qui précède le texte BASIC doit toujours être un $00
$0801	$08	$0808 : adresse de la ligne BASIC suivante.
$0802	$08	
$0803	$0A	$000A =10 : numéro de ligne
$0804	$00	
$0805	$99	$80 + $19 : 25ème mot-clé de la table = PRINT
$0806	$41	= 'A' en ASCII
$0807	$00	signifie fin de ligne Basic.

$0808	$10	$0810 adresse de la ligne BASIC suivante
$0809	$08	
$080A	$14	$0014= 20 : numéro de ligne
$080B	$00	
$080C	$89	$89= $80+$9 : 9me mot-clé de la table = GOTO
$080D	$31	'1' en ASCII
$080E	$30	'0' en ASCII
$080F	$00	signifie fin de ligne BASIC
$0810	$00	Adresse de la ligne BASIC suivante. La valeur $0000 signifie fin de programme.
$0811	$00	

En annexe, on trouvera le programme SUPERLIST64 qui donne les numéros et les adresses des lignes BASIC en mémoire.

La nécessité des pointeurs de lignes est bien claire : l'interpréteur de commandes connaissant ainsi les adresses de début et de fin de chaque ligne en mémoire, peut insérer une ligne à sa place par ordre croissant de numéros. Il peut aussi remplacer une ligne par une autre, de longueur différente, et même supprimer une ligne.

Après avoir rangé une ligne BASIC à sa place en mémoire, l'interpréteur de commande revient à son point de départ. Par contre, s'il a reçu une commande directe, après compactage de la commande, il transfère le contrôle à l'interpréteur BASIC.

L'interpréteur BASIC est constitué de nombreuses routines différentes pour gérer les différentes commandes, fonctions, etc. Toutes ces routines, comme la routine principale d'interprétation ont très souvent besoin de prendre en mémoire un ou plusieurs caractères consécutifs pour les interpréter.

L'opération de prise d'un caractère est l'oeuvre du sous-programme CHRGOT qui est situé en mémoire RAM en page zéro (de $73 à $8A). Voir en annexe le texte de la routine CHRGOT. CHRGOT a deux fonctions : prendre un caractère en mémoire, ce qui implique que CHRGOT contienne un pointeur de caractère courant qui doit se modifier à chaque appel.

De plus, si le caractère est un espace, ($20) , CHRGOT l'ignore et prend le caractère suivant. Deuxièmement, le flag CARRY du 6510 est affecté par le type de caractère lu; le CARRY est mis à un si le caractère est alphabétique ou numérique. Le programme appelant peut ainsi voir en une seule instruction (BCC ou BCS) si le caractère que CHRGOT lui renvoie est alphabétique. CHRGOT est de très loin la routine la plus employée

de tout le 64, ce qui explique le soin extrême apporté à sa réalisation. La routine CHRGOT ne pourrait être aussi efficace et aussi courte (24 octets) ailleurs qu'en page zéro.

L'interprétation d'une commande BASIC commence toujours en $0200, début du tampon d'entrée. Cependant, certaines commandes comme RUN, GOTO et GOSUB modifient les variables de l'interpréteur BASIC de manière à diriger l'interprétation vers la ligne dont le numéro est spécifié dans l'instruction. Ces trois mots-clés sont les seuls à pouvoir faire sortir le pointeur d'interprétation de la zone du tampon d'entrée. Pour s'en convaincre, il suffit de lire la valeur de ce pointeur par des PEEK (aux adresses $7A,$B ou 122 et 123 en décimal) en mode direct (La valeur lue est toujours dans la zone $0200) puis par programme.

La routine principale d'interprétation est celle qui, à la fin de l'exécution du programme correspondant à une commande BASIC, décode le mot-clé lui-même et passe le contrôle au sous-programme correspondant.

L'interpréteur appelle CHRGOT, qui lui renvoie un caractère TOKEN de valeur comprise entre $80 et $CB : en déduisant $80, il trouve le numéro du mot-clé dans la table des mots-clés en $C09E. Mais il existe dans la mémoire morte du CBM 64 une autre table, qui contient (sur deux octets chaque fois) l'adresse de chacun des programmes correspondant aux mots-clés. Cette table est située en $A00C. Soit le mot-clé PRINT (code $99 ou 153): le numéro du mot-clé est 153-128 = 25. Chaque élément de la table ayant deux octets de long, l'adresse de la routine PRINT est située en ($A00C+1+2*25 = $A03E) et ($A00C+ 1+2*25+1=$A03F). La valeur lue en $A03E est $9F et en $C03F, on trouve $AA.

L'interpréteur ayant trouvé ainsi l' adresse $AA9F, la dépose dans la pile et par l'instruction RTS, fait un saut à l'adresse en question. En effet l'instruction RTS -retour de sous-programme- retourne à l'adresse qui suit celle qui avait été mise dans la pile par l'instruction JSR. Ceci explique que les adresses dans la table en $A00C ne sont pas les adresses des programmes, mais bien les adresses des programmes-1. Notez le dernier mot-clé de la table des mots-clés, GO, ce qui permet l'écriture de GO TO en deux mots.

On peut difficilement passer en revue par le détail tous les sous-programmes d'interprétation, mais la structure générale qu'ils emploient est toujours identique. Le transfert entre BASIC et KERNAL passe presque chaque fois par un vecteur en RAM, ce qui permet de modifier à volonté les entrées-sorties. En ce qui concerne le BASIC proprement dit, les différents sous-programmes ne s'appellent entre eux que de façon très restreinte.

Presque chaque fois, les arguments à transmettre entre sous-programmes sont stockés en mémoire RAM (en page 0 généralement). Cette méthode permet à chaque élément de l'interpréteur BASIC de faire appel à toute une série de repères: est-on en mode direct, en mode "guillemets", quel est le périphérique de sortie à un moment donné,etc...

La transmission des paramètres entre sous-programmes est le plus souvent systématisée comme suit : si un seul paramètre entier d'un seul octet de long est à transmettre, on emploie usuellement l'accumulateur du 6510 : le registre A. Si le paramètre est un nombre entier codé sur deux octets (valeur comprise entre 0 et 65535), on emploie les registres Y et A, l'octet de poids faible dans le registre A, l'octet de poids fort dans le registre Y. Exemple : pour imprimer une chaîne de caractères, il faut transmettre à la routine d'impression l'adresse mémoire du premier caractère de la chaîne. Donc, pour imprimer "**** COMMODORE 64 BASIC V2 ****",message présent en mémoire en $E479, on effectue le programme suivant :

```
LDA £$79
LDY £$E4      ;voir détail de cette routine
JMP $AB1E     ;en fin de chapitre .
```

Les opérations mathématiques, algébriques, logiques et trigonométriques sont toutes effectuées en notation flottante, c'est-à-dire que les variables sont toujours converties, dans le CBM 64, dans un format standard se rapprochant de ce que l'on nomme généralement la NOTATION SCIENTIFIQUE. Un nombre est ainsi codé sur 5 octets, qui représentent une valeur comprise entre +1 et -1 (la MANTISSE) et un octet qui représente la partie exponentielle (l'EXPOSANT).

Le bit de poids fort de la mantisse représente le signe de la mantisse (1=négatif, 0=positif). La mantisse est en notation "complément à 2". La valeur de l'exposant est égale à la puissance de 2 par laquelle il faut multiplier la mantisse. Dans l'exposant, le bit 7 représente le signe : Il vaut 1 pour un nombre négatif et 0 pour un nombre positif. $80 représente donc l'exposant nul qui est la marque d'un nombre nul. De nombreux exemples sont repris dans la description détaillée en fin de chapitre.

Comment décoder un nombre en format flottant ?
Tout d'abord l'exposant : déduire $81 de sa valeur. On obtient la puissance de 2 comprise dès lors entre +126 et -129 (-129 <=E<=126). La mantisse étant binaire et normalisée sur 32 bits, est comprise, en valeur absolue, entre 1 et 2 (1<= M <2). La valeur du bit 31 représente le signe. Les valeurs décimales fractionnaires de chaque bit sont reprises au tableau ci-après :

Octet MSB de poids fort:							
BIT31 =SIGNE	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24
Octet suivant:							
BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16
Octet suivant:							
BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Octet LSB de poids faible:							
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0

Exemples de nombres flottants

VALEUR	VALEUR FLOTTANTE	Valeur dans l'accu FLP (octet6=signe,bit31 mantisse =1)
	EXP MANTISSE	EXP MANTISSE SIGNE
1E38	FF 16 76 99 52	FF 96 76 99 52 00
1E10	A2 15 02 F9 00	A2 95 02 F9 00 00
8 = 2^3	84 00 00 00 00	84 80 00 00 00 00
6.28318531=2*PI	83 49 0F DA A2	83 C9 0F DA A2 00
1 = 2^0	81 00 00 00 00	81 80 00 00 00 00
0.5 = 2^-1	80 00 00 00 00	80 80 00 00 00 00
0	00 n'importe quoi	00 n'importe quoi 00
-0.5 = 2^-1	80 80 00 00 00	80 80 00 00 00 FF
-10	84 A0 00 00 00	84 A0 00 00 00 FF

FORMULE : VALEUR = MANTISSE * 2 ^ EXP

numéro du bit	Valeur binaire	Valeur décimale
31	Signe	1= négatif,0=positif
30	2^-1	0,5
29	2^-2	0,25
28	2^-3	0,125
27	2^-4	0,0625
26	2^-5	0,03125
25	2^-6	0,015625
24	2^-7	0,0078125
23	2^-8	0,00390625
22	2^-9	0,001953125
21	2^-10	0,0009765125
20	2^-11	0,0004882812
19	2^-12	0,0002441406
18	2^-13	0,0001220703
17	2^-14	0,0000610351
16	2^-15	0,0000305175
15	2^-16	0,0000152587
14	2^-17	0,0000076293
13	2^-18	0,0000038146
12	2^-19	0,0000019077
11	2^-20	0,0000009536
10	2^-21	0,0000004768
9	2^-22	0,0000002384
8	2^-23	0,0000001192
7	2^-24	0,0000000596
6	2^-25	0,0000000298
5	2^-26	0,0000000149
4	2^-27	0,0000000074
3	2^-28	0,0000000037
2	2^-29	0,0000000018
1	2^-30	0,0000000009
0	2^-31	0,0000000004

Pour trouver la valeur décimale d'une mantisse, il faut la transcrire en binaire, puis faire la somme de toutes les valeurs décimales de chaque bit qui porte la valeur 1. En ajoutant 1 à cette somme et en tenant compte du signe représenté par le bit 31, on obtient aisément la valeur de la mantisse complète.

Les calculs étant effectués en flottant, il est nécessaire d'avoir au moins deux zones pour stocker deux opérandes pour une multiplication par exemple. Ces deux zones s'appellent les accumulateurs flottants. (ACCU FLP1et ACCU FLP2). L'ACCU FLP1 occupe les adresses $61 à $66, l'ACCU FLP2 les adresses $69 à

$6E. Si un seul argument est nécessaire, (comme pour les fonctions trigonométriques SIN et COS), seul l'ACCU FLP1 est utilisé. Les accus FLP occupent 6 octets chacun: les 5 premiers représentent la valeur flottante, le sixième est une image du signe de la mantisse durant les différents calculs. Dans cet octet -$66 ou $6E - la valeur ne peut être que $00 = signe positif ou $FF = signe négatif.

Deux autres adresses sont chaque fois remises à jour par l'interpréteur, il s'agit de $6F qui contient le signe du produit (ou le produit des signes) des deux accus FLP, et de $70 qui contient un bit supplémentaire de poids faible permettant d'obtenir des arrondis corrects lors des conversions entier-flottant et réciproquement ainsi que dans les opérations trigonométriques. Ce bit est appelé en anglais GUARD-BIT ou bit de garde.

Le langage BASIC autorise l'emploi de variables entières de valeurs comprises entre -32768 et +32767 mais, dans le cas des ordinateurs COMMODORE, l'emploi de ces variables ralentit l'exécution du BASIC car elles nécessitent à chaque usage une conversion flottant-entier. L'intérêt de ces variables est, dans le cas des tableaux, un certain gain d'espace-mémoire.

Le principal intérêt de ce chapitre est de permettre, par la compréhension du fonctionnement de l'interpréteur, la création de programmes divers. Des sous-programmes en BASIC ou en langage machine et, pourquoi pas, de "simples" appels SYS à la ROM BASIC confèrent au 64 certaines propriétés de l'interpréteur non accessibles classiquement.

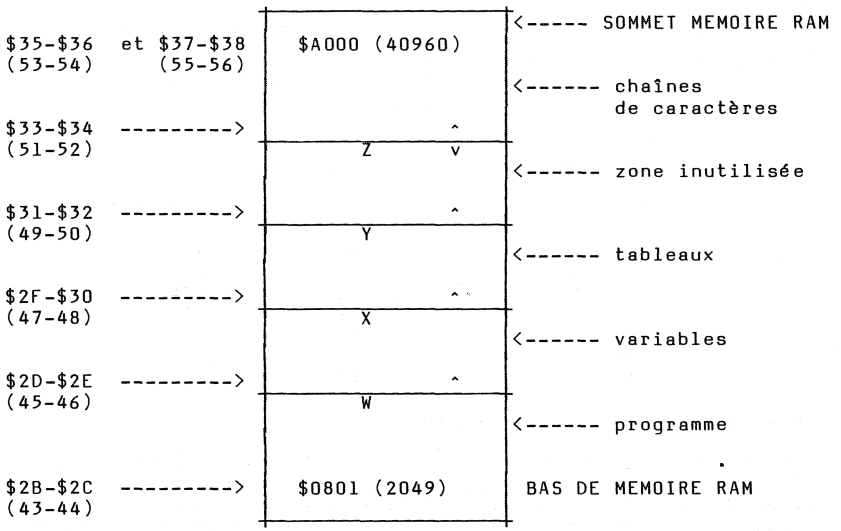
Mais, pour aller plus avant dans le fonctionnement de l'interpréteur, il est nécessaire de bien comprendre la manière dont les variables BASIC sont rangées en mémoire.

L'interpréteur BASIC exige un espace mémoire RAM d'un seul tenant. Le programme lui-même pourrait être en mémoire morte mais il doit être suivi de l'espace des variables, qui se modifie et impose donc l'usage de mémoires vives.

Le tableau de la page suivante détaille l'usage que fait BASIC de cette zone de mémoire RAM qui lui est accessible.

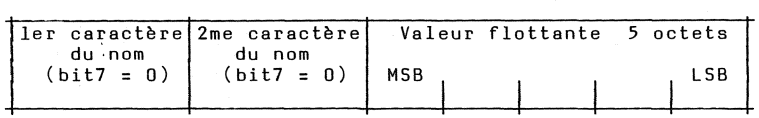
Les adresses situées à gauche du tableau sont celles des pointeurs en page 0 où sont sauvées et mises à jour en permanence les adresses de début de chacune des zones de mémoire. Les flêches indiquent le sens d'évolution des débuts de zones pendant l'exécution d'un programme.

DETAILS DE L'USAGE DE LA MEMOIRE VIVE PAR BASIC



Le détail de la zone programme a été étudié ci-dessus. Etudions maintenant la zone VARIABLES. La zone variables peut contenir quatre types de variables différentes : les flottantes, les entières, les chaînes et les fonctions. Dans un souci de simplification, toutes les variables sont codées sur 7 octets de long : 2 pour le nom de la variable, 5 pour la valeur. L'espace mémoire réservé au nom de la variable est de 2 octets. Or BASIC accepte des variables avec des noms de longueur quelconque. Mais, attention! Il ne reconnaît que les deux premiers caractères (le premier alphabétique, le second alphanumérique). Donc, pour BASIC, les variables FACTURE et FAMILLE ne forment qu'une : la variable FA. (Essayez FACTURE = 0: FAMILLE = 2 : PRINT FACTURE). Les caractères suivants des noms de variables sont présents dans le texte BASIC mais JAMAIS dans la zone variables.

1.Variable flottante



2.Variable entière

Les variables entières sont des valeurs codées sur deux octets en notation "complément à deux". Le bit 7 à zéro indique un nombre positif.

ler caractère du nom (bit7 = 1)	2me caractère du nom (bit7 = 1)	MSB (haut) bit7= signe	LSB (bas)	0 3 OCTETS	0 INUTILES	0

Exemples de valeurs entières:

Valeur décimale	MSB	LSB
0	$00	$00
256	$01	$00
4100	$10	$04
-1	$FF	$FF
-2	$FF	$FE
32767	$7F	$FF
-32768	$80	$00

3.Variables chaînes de caractères

Les chaînes de caractères ont des longueurs très différentes, mais toujours inférieures à 256 caractères (0<=L<256). Les chaînes sont donc sauvées dans la zone chaînes au sommet de la mémoire disponible et les retrouver reste aisé car l'adresse du premier caractère et la longueur de la chaîne suffisent pour les retrouver. Ceci nécessite 3 octets dans la variable elle-même. La variable chaîne ne contient jamais les caractères eux-même mais seulement un pointeur.

ler caractère du nom (bit7=0)	2me caractère du nom (bit7=1)	long.	LSB bas adresse de la chaîne	MSB haut	0 2 OCTETS INUTILES	0

4.Les fonctions FN

Les fonctions FN sont définies par l'instruction DEF.
Exemple :

```
DEFFNAA(X) = PEEK(X) + 256* PEEK (X+1) .
```

La valeur rendue dans B par B=FNAA(Y) est l'adresse (en décimal) qui est logée en mémoire sous la forme BAS, HAUT sur deux octets à l'adresse Y. Par exemple, l'adresse de la fonction USR en $0311 et $0312 (785 et 786) s'obtient par PRINT FNAA(785). Dans les cas simples où il n'y a qu'un paramètre à transmettre à une sous-routine, les fonctions FN sont très

efficaces et très rapides. Dans l'exemple ci-dessus la variable X n'est utilisée que dans la fonction et n'a pas d'autre usage. Lors de l'appel de la fonction par B=FNAA(Y), la valeur de la variable Y est transférée dans la variable locale pour l'exécution et réciproquement en fin de calcul de la fonction. Il faut noter que le pointeur de la variable locale pointe directement vers la valeur de la variable et non vers le premier caractère du nom de la variable.

ler caractère du nom de la fonction (bit7=1)	2me caractère du nom de la fonction (bit7=0)	LSB pointeur définition la fonction bas	MSB pointeur définition la fonction haut	LSB pointeur de la variable locale +$02 bas	MSB pointeur de la variable locale +$02 haut	premier caract. de la fonction

L'usage de la zone VARIABLES peut être rapide ou non suivant les programmes. En effet, la zone variables s'agrandit de 7 octets à chaque nouvelle création de variable, ce qui arrive à des moments divers dans le cours du programme. Bien sûr, CLR peut les effacer toutes, mais le procédé n'est guère élégant. L'ordre chronologique de création des variables détermine donc l'emplacement physique de celles-ci. Lors de la recherche d'une variable, l'interpréteur BASIC passe toute la table en revue. S'il ne la trouve pas, il la crée en fin de zone. On a donc tout intérêt à créer au début du programme les variables les plus couramment utilisées comme les indices des boucles FOR-NEXT, par exemple. Une simple ligne BASIC comme :

```
1  I=1 : J=1 : A$=""
```

peut améliorer la vitesse d'exécution d'un programme de plusieurs %. Pour lister les variables existantes en mémoire à un moment donné, on peut utiliser les pointeurs en $2D - début de variables - et en $2F - fin de variables - pour repérer la zone utile.

Voyez en annexe le programme LISTVARIABLES. Ce programme peut résider en mémoire en plus d'un autre programme (pour autant que la mémoire soit disponible). Il affiche la liste des variables en mémoire et leurs valeurs. On obtient également l'adresse des chaînes de caractères qui ont été utilisées.

Il faut surtout remarquer les sous-programmes en 63080, 63210, 63070 et 63290 qui peuvent être réutilisés à d'autres fins : ils réalisent la conversion des données du format binaire propre au 64 en caractères lisibles par tous. On remarquera le sous-programme 63070 qui affiche les deux caractères du nom d'une variable. La conversion des valeurs flottantes de binaire en décimal s'effectue à la ligne 63080. L'EXPOSANT ZE est calculé en 63080 et la MANTISSE ZJ en 63090. Le résultat décimal se retrouve dans ZJ en 63120.

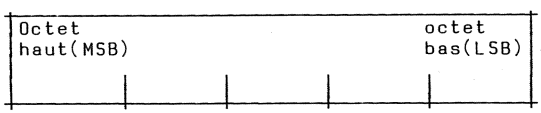
L'organisation de la ZONE TABLEAUX est fort similaire. Cependant, vu le grand nombre de valeurs que l'on peut être amené à utiliser, ici il n'est plus question de laisser des octets inutilisés. L'accès à une valeur est plus lent mais bien plus économe en espace-mémoire. Chaque tableau a une longueur différente, mais la structure est identique.

Octets 1 et 2	Mêmes caractéristiques que les deux premiers caractères des noms de variables simples. Seuls sont autorisés les types entier, flottant et chaîne.Noter qu' une variable et un tableau peuvent porter le même nom sans que la confusion soit possible : ils sont dans des zones différentes.
Octets 3 et 4	Longueur du tableau codée sur deux octets (bas,haut)depuis l'octet 1 inclus jusqu'au dernier octet du tableau (inclus).
Octet 5	Nombre d'indices (1-255) du tableau = N
Octets 6 et 7	Valeur maximale du premier indice. Noter que l'élément 0 est utilisable. L'indice varie donc de 0 inclus à la valeur stockée dans ces deux octets (inclue). Cette valeur est sous la forme (bas,haut).
Octets 8 à 5+2N	Valeur limite des indices suivants si N est supérieur à 1.
Octets 6+2N à fin	Eléments du tableau, rangés dans l'ordre croissant des indices. Exemple:A(0,0,0,); A(1,0,0); A(2,0,0); A(0,1,0); ETC...

Les éléments d'un tableau ont une longueur différente suivant le type de variable: Pour les nombres flottants 5 octets, pour les entiers 2 octets, pour les chaînes 3 octets.

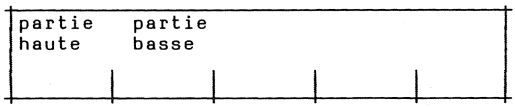
Format d'un élément de tableau :

Flottant



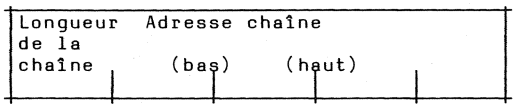
Voir la définition des nombres flottants ci-dessus.

Entier



Le bit 7 du premier octet représente le signe.

Chaîne



Pour rendre plus clair l'emploi de la zone tableaux, il est ici aussi possible d'utiliser le programme "LISTVARIABLES" en annexe. Notez bien que toutes les variables utilisées dans LISTVARIABLES doivent être déclarées avant d'entrer dans le programme ,sinon les pointeurs des différentes zones se modifient sans cesse.

Maintenant que la structure des variables est connue, voyons l'utilisation qu'en fait l'interpréteur BASIC. L'interpréteur BASIC est conçu pour exécuter un (et un seul) ordre à la fois. Le détail de chacun de ces ordres se trouve dans les 76 programmes dont les adresses sont dans la table en $A00C. Les 76 instructions BASIC sont à séparer en deux groupes distincts :Les 35 commandes BASIC et les 41 fonctions BASIC - les commandes sont les 35 premières instructions de la table en $A00C.

Les commandes BASIC représentent des ordres à effectuer, tandis que les fonctions représentent des valeurs, tout comme les variables. Les commandes BASIC peuvent nécessiter des arguments. Les arguments sont des valeurs, soit des variables, soit des constantes, soit des fonctions BASIC. Nous regrouperons les commandes en cinq groupes : sans argument, avec arguments, branchemment inconditionnel et conditionnel, commandes d'entrées/sorties.

1.LES COMMANDES SANS ARGUMENT sont au nombre de 8 :

END, REM, DATA, RESTORE, STOP, CONT, CLR, NEW.

Nous allons détailler ici leur fonction à l'intérieur de l'interpréteur BASIC, sans expliciter particulièrement l'usage de cette instruction en tant que mot de vocabulaire du langage BASIC. Cette approche plus directe de la réalité des choses aide par ailleurs à réviser nombre d'idées préconçues sur BASIC et son fonctionnement.

END et STOP Quand il rencontre une de ces instructions, BASIC teste, comme pour les 8 instructions de ce groupe, si le mot-clé est suivi d'un paramètre. S'il l'est, il exécute immédiatement la routine d'erreur. Ce test est simple. Si l'interpréteur BASIC est arrivé à la routine END avec le drapeau Z =0, c'est que celui-ci a été positionné par la routine CHRGOT qui a décelé que le premier caractère qui suit END est nul (fin de ligne). La routine d'erreur qui se situe en $A43A est alors exécutée.En entrant dans cette routine, le registre X doit contenir le numéro du message d'erreur à afficher. Voir la table des messages d'erreurs en $A19E. Si il n'y a pas de paramètre, BASIC teste si l'on est en mode direct. Si c'est le cas, il enlève l'adresse de retour (celle de l'interpréteur de commandes) de la pile et saute à l'adresse $A474.

Le message 'READY' s'affiche et l'interpréteur BASIC redémarre l'interprétation dans le tampon d'entrée en $0200. A noter que l'arrêt d'un programme par la touche STOP exécute l'instruction END après avoir affiché 'BREAK'.

CONT La routine END, avant de retourner au mode direct, a sauvé en $3D et $3E le pointeur d'interprétation de la routine CHRGOT. En rétablissant le pointeur en $7A, $7B, la commande CONT fait redémarrer l'interprétation là où elle s'est arrêtée, pour autant que l'on n'ait pas commis la moindre erreur entretemps, ni modifié le programme.

NEW et CLR Les pointeurs de fin de variables et de fin de tableaux sont rendus identiques au pointeur de fin de BASIC par la commande CLR.

Résultat : L'interpréteur, ne recherchant les variables dans la table qu'aux adresses comprises entre les pointeurs de fin de BASIC et fin de variables, ne trouvera plus aucune variable. Donc CLR n'efface pas les variables. Après un CLR involontaire, on peut récupérer les variables en "POKANT" une valeur adéquate en $2F et $30 (pointeur de fin de variables) ou en $31 et $32 (pointeur de fin de tableaux). Attention, prendre des valeurs plausibles, c'est-à-dire supérieures à la valeur du pointeur en $2D, $2E (pointeur début de variables). De même NEW remet le pointeur de fin de programme à la valeur du pointeur de début de programme. De plus, NEW remet à zéro les deux premiers octets de la zone texte BASIC (adresse de la seconde ligne BASIC). On peut donc récupérer le programme en rétablissant ces deux octets par POKE des valeurs correctes (si on les connaît, bien sûr!) et en rétablissant le pointeur de fin programme. (Voir programme DE-NEW). NEW effectue ensuite les commandes CLR et RESTORE.

RESTORE et DATA On sait que des valeurs (constantes) peuvent être gardées dans un programme par DATA. Lorsque l'interpréteur BASIC rencontre DATA il recherche un ':' ou la fin de ligne et passe à l'instruction suivante. Ce n'est que lors de l'exécution de READ qu'il s'en servira pour connaître quelle sera la prochaine donnée à lire. READ se sert d'un pointeur en $3F et $40 qui indique le numéro de ligne DATA en cours et d'un pointeur en $41, $42 qui indique physiquement l'adresse réelle de la prochaine donnée à lire. L'instruction RESTORE remet simplement ces pointeurs à leurs valeurs de départ, 0 pour le numéro de ligne et pour le pointeur l'adresse de début de programme -1. On voit tout de suite le parti à tirer de cette situation. Pour redémarrer un RESTORE à n'importe quelle ligne, il suffit de faire quelques POKE après avoir utilisé SUPERLIST 64 pour connaître les bonnes adresses!

REM est fort semblable à DATA pour l'interpréteur, mais ici, il ignorera toute la ligne même si elle contient ':', contrairement à DATA.

2.<u>LES COMMANDES BASIC AVEC ARGUMENTS</u> sont au nombre de six :

DIM, READ, LET, DEF, POKE et LIST.

D'autres commandes BASIC nécessitent des arguments mais sont caractérisées autrement (voir les types 3,4 et 5).La commande DIM est une commande d'initialisation qui crée un tableau de taille donnée au format défini plus haut. Les variables tableau sont rangées dans la mémoire par ordre chronologique de création. Conséquence de ceci : les tableaux déclarés les premiers par DIM sont les plus rapidement lus car BASIC recherche une variable ou un tableau dans la mémoire par ordre croissant d'adresses. On constate que les variables-tableaux occupent beaucoup d'espace-mémoire.

Essayez : B=FRE(0) : DIM A(4,4,4) : PRINT FRE(0)-B

Pour visualiser l'occupation-mémoire d'un tableau, ne pas oublier l'existence des éléments pour lesquels l'indice =0. On a vu ci-dessus que CLR efface toutes les variables et les tableaux. Il est possible de se créer soi-même une commande CLR-TABLEAUX qui n'efface que les tableaux :

```
POKE 49, PEEK(47) : POKE 50,PEEK (48)
```

Cette instruction modifie le pointeur de fin de tableaux de manière à annuler la longueur de la zone "tableaux".

La commande **READ** est la seule à utiliser les zones de données créees par la commande DATA. READ lit une valeur d'une zone de DATA et la range dans la variable spécifiée. L'adresse

de la valeur à lire est connue à tout instant car READ met à jour à chaque fois deux pointeurs : en 63 et 64 ($3F,$40) se trouve le numéro de la ligne BASIC contenant la commande DATA en cours. Une partie de la routine READ est partagée avec l'instruction INPUT : il s'agit de la recherche du paramètre et du transfert de la valeur dans la variable-paramètre. La différence est dans l'adresse de lecture de la valeur. Pour INPUT, elle est dans le tampon d'entrée; pour READ, elle est dans le corps du programme. A noter : les chaînes de caractères dans DATA ne doivent pas obligatoirement être entre guillemets.

La commande LET étant très fréquemment utilisée, les concepteurs de BASIC l'ont rendue optionnelle, c'est-à-dire que lors de la recherche d'un mot-clé dans la table, si le mot-clé n'est pas trouvé, on considère qu'il s'agit d'un nom de variable. Ce nom de variable est alors considéré comme l'argument de l'instruction LET, que LET ait été présent ou non dans le texte. BASIC teste ensuite la présence du signe '='. S'il est absent, il y a erreur de syntaxe. S'il est présent, BASIC évalue l'expression qui le suit et modifie ou crée la variable en lui affectant la valeur de l'expression. L'emploi de l'instruction LET explicite est avantageuse à un seul point de vue : la rapidité d'exécution. En effet, l'interpréteur ne scrute qu'une partie de la table des mots-clés (LET est en neuvième position).

La commande DEF a une fonction quasi-identique à LET : elle attribue une valeur à une variable de type 'FONCTION' .
Exemple : DEF FNAB (X) = SIN (X) crée la fonction AB. De plus DEF crée une variable (ici: X). Créer une fonction consiste à insérer dans la zone VARIABLES une zone de 7 octets contenant les pointeurs vers la variable et vers la zone où la fonction est définie (ici l'adresse du caractère 'SIN' dans le corps du programme. SIN est codé comme un seul octet, bien entendu). DEF utilise une variable locale qui n'existe pas vraiment. Son pointeur est remplacé par celui de la variable réellement utilisée lors de l'appel de la fonction. Le nom de cette variable n'a de signification qu'à l'intérieur de la définition de la fonction. Dans l'exemple ci-dessus, la variable X interne à la fonction peut coexister sans problèmes avec une variable X définie ailleurs dans le programme.

La commande **POKE** est excessivement simple à interpréter car très proche dans sa signification des instructions du 6510. POKE A,B est équivalent à : LDA £B;STA A. Le BASIC vérifie la présence des deux arguments puis convertit le premier en un entier sur deux octets : l'adresse. Le second est converti en entier sur un octet : la valeur.Cette valeur est ensuite rangée à sa place en mémoire.

La commande **LIST** est une commande particulière car ses arguments - ligne de début, ligne de fin - sont optionnels. Si un des paramètres est omis, BASIC le remplace par 0 pour le premier, 63999 pour le second. Avec un seul argument, LIST ne liste qu'une ligne, sauf pour LISTO qui affiche non la ligne

0 seule, mais tout le programme, comme LIST sans argument. Les mots-clés étant en mémoire sous une forme non directement lisible, une conversion est nécessaire. A cette fin la routine LIST se sert de la table des mots-clés en $A09E pour le transcodage. Une version BASIC de LIST est disponible en annexe (SUPERLIST 64). SUPERLIST 64 exécute les mêmes opérations de transcodage que LIST. Admirez la différence de vitesse entre BASIC et langage machine!

3. **LES COMMANDES DE BRANCHEMENT INCONDITIONNEL** permettent l'utilisation de sous-programmes et le retour à une branche unique de programme après un (ou plusieurs) branchement conditionnel. Il y a six commandes de ce type : GOTO et GO TO, RUN, GOSUB et RETURN, SYS. SYS ainsi que GOSUB et RETURN font appel à la notion de pile. Etudions d'abord les cas les plus simples.

GOTO et **GO TO** effectuent un branchement inconditionnel vers le début d'une ligne dont on spécifie le numéro. Curieusement l'argument ne peut pas être une expression ! L'exécution de GOTO est simple, en principe : il suffit de modifier le pointeur de la routine CHRGOT pour continuer l'interprétation plus loin. GOTO effectue ceci de manière assez efficace. Le numéro de ligne après calcul (il faut convertir la chaîne de caractères en entier sur deux octets) est rangé en mémoire aux adresses 20 et 21 ($14,$15) puis comparé au numéro de ligne en cours. S'il s'agit d'un saut en avant, BASIC cherche la ligne destination à partir du pointeur courant, c'est-à-dire à partir de la ligne qui suit celle du GOTO.

Si, par contre, il s'agit d'un saut en arrière (exemple : 10 GOTO 10), BASIC commence sa recherche à partir de la première ligne du programme. Si la ligne n'est pas trouvée, le message :'UNDEF'D STATEMENT ERROR' est généré. La possibilité de coder GOTO en deux mots GO et TO a été conservée pour le 64 car le BASIC MICROSOFT 2.0 était ainsi fait. Il s'agit de la version de l'interpréteur BASIC installée dans les PET et VIC depuis 1977 par MICROSOFT.

A l'exécution, il n'y a pas de différence entre ces deux GOTO, mais celui en deux mots occupe dans la mémoire deux octets de plus au moins que la version simple. Cela dépend du nombre de blancs entre GO et TO.

RUN effectue le branchement inconditionnel comme GOTO, mais après avoir réalisé CLR. La différence est très marquée car toutes les variables et tous les tableaux sont remis à zéro par RUN. RUN 30 est équivalent à CLR : GOTO 30.

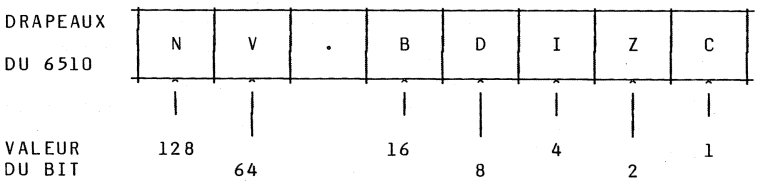
De plus, RUN referme tous les fichiers ouverts. En mode direct, on frappe usuellement RUN pour exécuter un programme, mais parfois GOTO N en mode direct peut s'avérer fort utile, entre autres, pour la mise au point des programmes.

SYS est une instruction de branchement inconditionnel vers un sous-programme écrit en langage-machine. En fin de sous-programme, l'interpréteur BASIC redémarre et passe à l'instruction suivante. L'adresse d'exécution de l'interpréteur

BASIC ne doit donc pas être perdue. Cette adresse est sauvée dans la PILE, d'où elle est récupérée par l'instruction RTS. La pile est la zone mémoire de 256 octets qui commence en 511 ($1FF) et décroît jusque 256($100) au maximum. Cette zone est réservée au stockage de valeurs temporaires grâce à un pointeur indiquant l'adresse du premier octet libre dans la pile. Ce pointeur est câblé à l'intérieur du 6510 lui-même : c'est le registre SP. La pile est mise à jour en respectant le principe LIFO-LAST IN FIRST OUT- c'est-à-dire que le premier entré est le dernier sorti. Cette structure de mémoire, bien connue des possesseurs de calculatrices HEWLETT-PACKARD, est particulièrement efficace pour la gestion des sous-programmes.

De très nombreux sous-programmes en langage-machine très utiles sont présents dans les ROM du CBM 64. Mais ils nécessitent quasi toujours la transmission de paramètres dans les registres A,X et Y du 6510. C'est pourquoi l'interpréteur BASIC, après avoir évalué l'expression derrière SYS - les parenthèses ne sont pas obligatoires -, charge les registres A, X et Y avec le contenu des adresses 780, 781 et 782. L'octet de STATUS du 6510 peut également être chargé (contenu en 783). Ces adresses $030C à $030F peuvent être chargées en BASIC par des POKE avant d'exécuter le SYS. Après le SYS, BASIC peut relire les valeurs contenues dans A, X, Y et le STATUS (drapeaux) du 6510 au moment du RTS qui rend le contrôle au BASIC par PEEK (780),...,PEEK(783).

Détaillons ici un point souvent resté OBSCUR :



Pour positionner un drapeau à 1, mettre la valeur correspondante dans la variable qui sera posée en 783. Exemple : pour positionner les drapeaux N et C à 1 et les autres à 0 :

```
FL = 128+1 : POKE 783,FL
ou   POKE 783, 128 OR 1
```

Pour appeler un sous-programme en langage machine avec transmission d'un paramètre en notation flottante, il faut employer la fonction USR et non la commande SYS.

GOSUB et RETURN : GOSUB est identique au GOTO avec un supplément ; l'adresse et le numéro de ligne de l'instruction en cours sont préalablement sauvés dans la pile pour être ensuite ré-utilisées par RETURN.

De même, RETURN est un GOTO avec le paramètre lu dans la pilee et non dans le corps du texte. La pile peut contenir, outre

des adresses de retour de sous-programmes en langage machine, deux types de données: Les blocs "FOR" et les blocs "GOSUB". Ce dernier est constitué de 5 octets:

Bloc "GOSUB"

ler octet	2ème	3ème	4ème	5ème
$80	numéro de ligne BASIC du GOSUB		pointeur du premieur caractère après le GOSUB	

L'instruction RETURN scrute la pile en remontant depuis la valeur SP vers le haut, puis trouvant le premier bloc qu'elle y rencontre (et donc le dernier qui y a été mis), l'en retire pour ensuite se rendre à cette adresse et y continuer l'interprétation.

4.LES BRANCHEMENTS CONDITIONNELS

Ils sont les commandes les plus fondamentales de l'informatique. En effet, toute l'intelligence des ordinateurs est concentrée dans ces seuls mots : FOR et NEXT, IF et THEN, ON et WAIT. Ces commandes sont d'ailleurs des variantes du même concept: une action est à réaliser seulement dans certains cas: les conditions. Sans les instructions conditionnelles, l'informatique d'aujourd'hui serait l'ombre d'elle-même : des machines à écrire, à trier,etc... mais pas des ordinateurs !

A tout seigneur, tout honneur : examinons IF, l'instruction numéro 1 de l'informatique.

Une instruction IF possède toujours la structure :

IF condition THEN commande....

bien que le BASIC du 64 accepte aussi la structure :

IF condition GOTO numéro de ligne.

Cette dernière version permet d'économiser temps et place mémoire, THEN est alors sous-entendu. La commande qui suit THEN peut être n'importe quelle commande BASIC, y compris IF et GOTO. La condition ne peut être qu'une valeur numérique. Le branchement est effectué vers l'instruction qui suit THEN si la condition n'est pas ZERO. Le branchement est effectué vers la ligne BASIC suivante si la condition est ZERO.

Une idée préconçue très courante : si A vaut 1 et que l'on écrit IF A<> 0 THEN PRINT, on sait que PRINT sera exécuté parce que la condition est vraie. Mais en fait, A<>0 est une expression dont la valeur est -1, d'où le branchement. En fait, IF A THEN PRINT est suffisant.

Une condition est une valeur. Suivant qu'elle est nulle ou non nulle, deux actions différentes sont entreprises : il est bien dommage de voir que seule (ou à peu près) la structure IF A=B THEN... est utilisée alors que des commandes telles que :

```
IF A THEN...
IF FNX (A) THEN...
```

ne se rencontrent que très peu. Tout ceci provient de la méconnaissance du fonctionnement de IF. Caractérisons-le:

```
IF -1 THEN PRINT "ceci est toujours imprimé"
IF 0 THEN PRINT "ceci ne s'écrira jamais"
```

(-1 peut être remplacé par une autre valeur non nulle.)

ON...GOTO et **ON...GOSUB** sont des versions plus sophistiquées de IF. La syntaxe à employer est similaire :

ON condition GOTO ou GOSUB No de ligne, No de ligne, No de ligne...Mais il y a plusieurs adresses de branchement possibles. La condition est une VALEUR supérieure à zéro. Le branchement est effectué vers le premier numéro de ligne si la condition vaut 1, et ainsi de suite. La condition de valeur N branchera vers le Nième numéro de ligne.

Exemple :

```
ON A GOTO 100, 200, 300
```

est équivalent à:

```
IF A=1 GOTO 100
IF A=2 GOTO 200
IF A=3 GOTO 300
```

Les branchements multiples étant relativement fréquents, ce type d'instruction est souvent utilisé, le gain d'espace mémoire et de temps d'exécution étant appréciable.

FOR...NEXT... est une instruction dite 'de boucle'. Il s'agit d'une instruction (NEXT) qui branche conditionnellement un certain nombre de fois vers une adresse connue, celle du FOR. Un élément supplémentaire a été ajouté qui augmente énormément la puissance de l'instruction **FOR** : l'indice de boucle, variable utilisée pour comptabiliser le nombre de boucles exécutées est accessible au programmeur : c'est une variable classique. Dans le 64 , l'indice de boucle doit être une variable flottante. Comme les variables d'indices de boucles sont très fréquemment utilisées, il est quasiment impératif de les déclarer en début de programme pour accélérer l'exécution des boucles. Les indices de boucles classiquement utilisés sont : I,J,K le plus souvent. Un programme BASIC optimisé commencera donc par :

```
0 REM TITRE
1 I=0 : J=0 : K=0 (:DIM.....)
```

Si vous comptez utiliser des boucles imbriquées, déclarez

en premier lieu l'indice des boucles les plus intérieures.
La syntaxe générale est :

```
FOR var = limite inf. TO limite sup. STEP inc.
          ...
          ...
NEXT var(,var...)
```

La variable indice de boucle doit être flottante, mais les valeurs limite peuvent être quelconques, de même que l'incrément.

Essayez :
```
10 FOR I =0 TO 0.1 STEP 0.05
20 PRINT I
30 NEXT I
```

On obtient sur l'écran les valeurs :

```
 5E-03
 .01
 .015
 ...
 ...
 ...
 .08
 .0850000001
```

Cette dernière valeur provient d'erreurs d'arrondi. Il faut donc faire attention à ne pas tester la variable par des égalités mais seulement par des inégalités :

```
Ne pas écrire  : IF I= 0.85 THEN...
mais écrire    : IF I>=0.85 THEN ...
```

pour éviter les problèmes dûs à ces erreurs d'arrondi.

Les valeurs de début de boucle, de fin de boucle et d'incrément sont évaluées comme expressions puis rangées dans la pile par l'instruction FOR... .TO...STEP, puis le contrôle passe à l'instruction BASIC suivante.

L'instruction NEXT recherche le bloc "FOR" dans la pile, puis ajoute l'incrément à la variable indice et enfin effectue le test :

```
(indice) < limite fin de boucle si inc. positif
(indice) > limite fin de boucle si inc. négatif
```

Si le test est passé avec succès, le contrôle passe à la ligne suivante, sinon l'exécution reprend à l'instruction qui suit le FOR.
Après être sorti d'une boucle, la variable indice a une valeur égale à (valeur de départ) + (incrément) * (nombre de

fois que la boucle a été parcourue) aux erreurs d'arrondi près. Cette valeur n'est donc pas égale à la limite de fin de boucle.

Essayez :

```
FOR I = 1 TO 15 STEP 0.2 : NEXT I : PRINT I
```

Un bloc FOR dans la pile contient 18 octets et il y a 240 emplacements utilisables environ dans la pile, ce qui explique que l'on ne peut imbriquer un nombre illimité de boucles avant de remplir la pile (OUT OF MEMORY ERROR).

Bloc FOR		
	1er octet	: $81
	octets 2-3	: pointeur variable indice de boucle
	octets 4-8	: incrément en flottant
	octets 9	; signe de l'incrément
	octets 10-14	: fin de boucle en flottant
	octets 15-16	: numéro de ligne FOR
	octets 17-18	: pointeur vers le premier caractère qui suit l'instruction FOR.

La meilleure manière de sortir précocement d'une boucle FOR est de modifier l'indice et d'exécuter NEXT, ce qui supprime le bloc FOR de la pile. Sinon on s'expose à des erreurs. Notammentà la rencontre du prochain NEXT sans nom de variable !

Exemple :

```
10   FORI=0TO100
20   :
30   IFI>47THENI=101:NEXT:GOTO1000
40   NEXT I
...
...
```

WAIT est une instruction de branchement conditionnel particulière. Ou bien le branchement ne s'effectue pas et l'on passe à l'instruction suivante, ou bien le branchement s'effectue sur le début de l'instruction elle-même et il y a bouclage.

Attention, ce bouclage s'effectue entièrement en langage machine et si la condition ne se réalise pas, seul STOP/RESTORE permet de sortir de la boucle. La condition est une expression calculée à partir des trois paramètres de WAIT : WAIT A,B,C

Le premier argument est une adresse (valeur 0-65535)
Le deuxième et le troisième sont des octets (valeur 0-255)

WAIT attend que la condition se réalise, c'est-à-dire que la

valeur lue à l'adresse A soit nulle après passage par les opérations logiques XOR C (OU EXCLUSIF avec le troisième argument qui vaut 0 s'il n'est pas cité explicitement) suivi de AND B (ET avec le deuxième argument).

Par exemple, 10 WAIT 198,3,1 est semblable à :

```
10 IF ( PEEK (198) AND 3 OR 1) GOTO 10
```

L'usage de WAIT est simple : pour attendre une certaine configuration binaire d'un octet précis, on fournit en premier argument l'adresse de l'octet et en second argument, la valeur des bits que l'on veut tester. Dans l'exemple ci-dessus, AND 3 signifie : je ne m'intéresse qu'aux bits 0 et 1 car 3=00000011 en binaire et seuls les bits 0 et 1 valent 1.

La valeur du troisième argument permet de définir si le (ou les) bit(s) que l'on attend est (sont) un 1 ou un 0. Dans notre exemple, si les bits 1 et 0 doivent valoir 0 et 1 respectivement, on mettra le bit 1 à 0 et le bit 0 à 1 dans le troisième argument, les autres bits n'ayant pas d'importance car on les a "masqués" lors du AND: on les met à 0.

```
                              Valeurs                Resultats
                             en binaire              en  binaire
ler cas : WAIT 37164,48,16

      10 IF PEEK(37164)      01011110  (94)      01011110   (94)
            AND 48           00110000  (48)      00010000   (16)
            XOR 16           00010000  (16)      00000000   ( 0)
            THEN 10                              = 0 : on n'attend pas.

2me cas : WAIT 37164,48,16

      10 IF PEEK(37164)      01111110  (126)     01111110   (126)
            AND 48           00110000  ( 48)     00110000   ( 48)
            XOR 16           00010000  ( 16)     00100000   ( 32)
            THEN 10                              <>0 : on attend.
```

L'usage de l'instruction WAIT est en pratique limitée à l'attente de modification d'un bit d'entrée-sortie.

EXEMPLES:

attente bouton de tir enfoncé :

```
WAIT 56464,16,16
```

attente bouton relâché/enfoncé:

```
WAIT 65464,16
WAIT 65464,16,16
```

stopper un programme si on ne maintient pas une touche :

```
10 PRINT"*";:WAIT 197,64
20 GOTO 10
```

Avance pas à pas dans un programme :

```
10 PRINT "*";:WAIT 197,64,64
20 WAIT 197,64:GOTO 10
```

5. LES COMMANDES D'ENTREE-SORTIE.

Elles constituent le dernier bloc de commandes BASIC, mais non le moindre. Sans les entrées-sorties, BASIC ne pourrait ni communiquer les résultats de ses cogitations, ni gérer des fichiers, c'est-à-dire des données plus nombreuses que ce que la mémoire du 64 lui permet de stocker.

Il y a 11 commandes de ce type : INPUT£, INPUT, LOAD, SAVE, VERIFY, PRINT£, PRINT, CMD, OPEN, CLOSE et GET.
Les entrées-sorties dans les ordinateurs COMMODORE passent presque toutes par les deux mêmes points : la routine CHROUT de sortie en $FFD2 et la routine d'entrée CHRIN en $FFCF. Or, pour l'utilisateur, une entrée et une sortie, ce n'est pas suffisant: le KERNAL simule donc pour l'utilisateur diverses entrées et sorties. Ainsi l'utilisateur définit lui-même vers où se dirigent ses sorties et d'où viennent ses entrées : il les adresse par un numéro de fichier logique.

Il est plus souple d'utiliser un fichier logique qu'un périphérique physique comme sortie. Ainsi le même programme (identique) peut envoyer des données à l'écran ou à l'imprimante en modifiant simplement l'instruction OPEN.

Ainsi donc le système utilise des numéros logiques de fichiers, des numéros physiques de sorties et entre les deux un seul canal d'accès. Donc, le CBM ne peut accéder à deux périphériques de sortie (ou deux d'entrée) simultanément.

La liaison que BASIC et KERNAL réalisent entre fichier logique et périphérique physique est définie par l'utilisateur grâce à la commande OPEN.

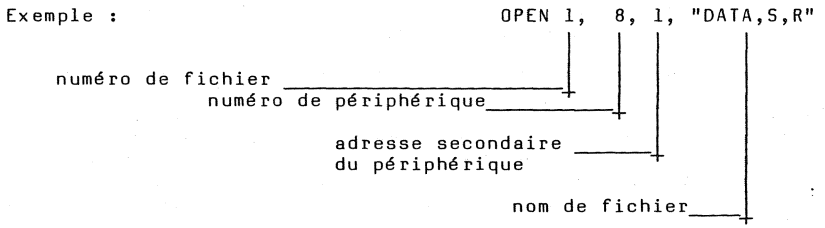
Les numéros de fichiers logiques peuvent être n'importe quel nombre entier compris entre 1 et 255. Il ne peut y avoir plus de 10 fichiers ouverts simultanément.

Les numéros de périphériques sont prédéfinis car ils dépendent physiquement des périphériques. Les périphériques internes ont leurs numéros définis dans les routines KERNAL qui les pilotent (DRIVERS). Ainsi le clavier porte le numéro 0, le lecteur de cassette le 1, l'interface série RS232 le 2, l'écran le 3. Les autres périphériques sont supposés se trouver sur le BUS IEEE série où les numéros autorisés vont de 4 à 31. Usuellement les imprimantes portent le numéro 4, le lecteur de disquettes le 8 (mais il est modifiable par programme), le modem téléphonique le 9.

Le canal d'entrée et le canal de sortie sont dédiés par défaut au clavier (0) et à l'écran (3), mais les commandes d'entrées/sorties peuvent les dévier vers d'autres. Quand aucun fichier n'est ouvert, le clavier et l'écran sont les seuls périphériques actifs et il n'y a donc pas lieu d'ouvrir un

fichier pour se servir des périphériques 0 et 3, bien que, parfois, cela représente des avantages, entre autres, une écriture de programmes générale et indépendante du périphérique en cours d'utilisation, ainsi qu'une suppression du '?' en entrée.

OPEN et **CLOSE**. Open ouvre un fichier, c'est-à-dire qu'il crée une correspondance entre un numéro logique utilisé par GET, PRINT, INPUT ou CMD et un périphérique physique.



Les deux premiers arguments sont obligatoires, les autres sont des options nécessaires pour dialoguer avec les périphériques les plus complexes. L'adresse secondaire est un paramètre retransmis au périphérique pour lui signaler une caractéristique spéciale du transfert. Les adresses secondaires n'ont de signification que pour le périphérique concerné.

Le lecteur de disques (8) peut utiliser les adresses secondaires 1, 2, 3,...,7 pour spécifier quel tampon mémoire RAM doit être utilisé à l'intérieur du lecteur de disques (on peut ouvrir plusieurs fichiers à la fois vers le lecteur). De même, pour le lecteur de disques, l'adresse secondaire 15 signifie "ceci est une commande pour le lecteur de disques".

Le lecteur de cassette se sert de l'adresse secondaire d'une façon particulière. L'adresse secondaire pour la cassette peut prendre les valeurs 0, 1, 2 ou 3. Pour les fichier, on n'utilise en général que 1 et 3.

1 signifie : bloc de données ordinaires.

3 signifie : bloc de données avec marqueur E.O.T., c'est-à-dire fin de bande. Lors de la recherche d'une information sur cassette, EOT signale au CBM-64 que la fin de bande est atteinte et que la recherche se termine.

Voir **LOAD** et **SAVE** pour les autres usages des adresses secondaires 0, 1, 2 et 3.

Le nom de fichier est un paramètre indispensable pour le lecteur de disques et très utile pour le lecteur de cassettes

car il permet de retrouver des fichiers par leur nom et non seulement par leur position physique réelle. Dans les autres cas, le texte est transmis au périphérique en question en lui laissant le soin de le gérer comme il l'entend.. Par exemple, pour le périphérique 2, c'est le KERNAL qui intercepte les deux caractères du nom de fichier pour initialiser les paramètres de transmission.

A la réception de la commande OPEN, l'interpréteur BASIC sauvegarde dans la table en mémoire en 601 ($259) les numéros de fichiers, les adresses primaires et les adresses secondaires. Ces valeurs permettront dans la suite d'aiguiller correctement les données vers le périphérique convenable.

La commande CLOSE n'a qu'un paramètre : le numéro de fichier. La commande CLOSE envoie sur le périphérique concerné les octets qui lui sont destinés et qui n'ont pas encore été transmis (cas d'une sortie) ou cesse de recevoir des octets (cas d'une entrée) puis retire de la table en 601 les paramètres relatifs au fichier refermé. Le pointeur en 152 ($98) qui indique à tout moment le nombre de fichiers ouverts est remis à jour.

Les instructions d'entrée GET, INPUT, INPUT£ s'adressent toutes à un périphérique d'entrée. GET et INPUT reçoivent directement des caractères du périphérique d'entrée par défaut (normalement le clavier) dont le numéro se trouve en 153($99). INPUT£ ou GET£ reçoivent des caractères en provenance du fichier ouvert dont le numéro est précisé comme argument.

GET ou GET£ prennent un seul caractère (ou aucun s'il n'y en a pas de disponible à cet instant) en provenance d'un périphérique. Pour savoir si un caractère est disponible au clavier avant d'effectuer un GET, il faut lire la valeur KBDPTR en 198 ($C6) qui indique le nombre de caractères disponibles. Pour les autres périphériques, il faut tester l'octet **ST-STATUS** en 144 ($90).

INPUT prend une suite de caractères au clavier jusqu'à la frappe d'un RETURN. INPUT£ fait de même sur un périphérique quelconque. Attention, par exemple avec INPUT£2,A$ si vous recevez des caractères d'un autre ordinateur par RS-232 : en effet, l'autre aura probablement émis en fin de ligne un **RETURN** suivi d'un LINEFEED c'est-à-dire que vos INPUT£2 vont rendre à partir de la deuxième fois des chaînes de caractères contenant un CHR$(10)-LINEFEED en première position. Dans le cas de INPUT£2,A$, ce n'est pas encore trop grave, mais pour des nombres, cela conduit à des résultats erronés.

Il faut noter que GET est dans la table des mots-clés, tandis que GET£ n'y est pas. L'instruction GET£ est donc codée sur deux octets. INPUT et INPUT£ sont codés chacun sur un seul octet.

A NOTER: L'instruction INPUT£ se bloque irrémédiablement si elle ne rencontre pas un caractère "RETURN"

(CHR$(13)) avant 80 caractères ! Faites donc attention à la taille des enregistrements contenus dans un fichier.

Les commandes PRINT et PRINT£ effectuent l'opération inverse de INPUT : envoyer des nombres ou des chaînes de caractères vers un périphérique de sortie. PRINT est destiné au périphérique de sortie par défaut : il s'agit normalement de l'écran mais CMD peut en décider autrement. PRINT convertit donc **tout** ce qu'il doit imprimer en caractères et les envoie sur le canal de sortie (routine CHROUT en $FFD2).

PRINT£ fait de même à quelques nuances près : le numéro de fichier précisé dans l'ordre PRINT est utilisé pour affecter le canal de sortie CHROUT au périphérique défini lors de l'OPEN du fichier en question. A la fin de l'instruction, le canal de sortie est réattribué au périphérique par défaut, normalement l'écran. Il faut bien noter la différence entre cette commutation du canal de sortie et l'instruction CLOSE : CLOSE enlève les caractéristiques d'un fichier de la table des fichiers ouverts en 601, tandis que la commutation du canal de sortie ne modifie pas cette table mais définit uniquement quel est l'actuel périphérique de sortie.

L'instruction PRINT envoie sur le canal de sortie la suite de caractères décrite dans l'argument de l'instruction. Cet argument s'appelle une LISTE.

Une LISTE est constituée de plusieurs éléments :

1. les expressions
2. la ponctuation
3. les fonctions de liste.

Les expressions sont des expressions BASIC évaluables par la routine EVAL en $AEF1. Exemple :

```
A$
A+10^4
FNA(X)+2
```

A l'impression, la valeur d'une expression est précédée d'un espace si sa valeur est positive et est toujours suivie d'un espace.

La ponctuation : les virgules et les point-virgules.Le rôle du point-virgule est de séparer deux éléments consécutifs de la ligne et ne provoque aucune impression.

```
PRINT;          ne sert à rien mais est valable.
PRINT A$B$C$    est valable.
PRINT A+2B+3    n'est pas valable.
PRINT A+2;B+3   est valable.
```

On voit que le ; reste obligatoire pour séparer les expressions non simples. Le ; en fin d'instruction signifie que la

liste n'est pas finie, la suite est dans le prochain PRINT. En l'absence de ; en fin de PRINT, BASIC envoie le caractère CHR$(13) (RETURN) et le caractère CHR$(10) (LINE FEED). Donc pour ne pas envoyer de CHR$(10) dans un fichier, il faut marquer la fin de la ligne ainsi :

```
PRINT£X,A$;B$;CHR$(13);
```

La virgule représente l'avance au tabulateur suivant : à sa rencontre, BASIC envoie un nombre d'espaces variable de façon à ce que le caractère suivant se trouve au début de la zone d'impression suivante, c'est-à-dire au 1er, 11eme, 21eme,31eme caractère de la liste,etc...Les fonctions de liste sont SPC(X) et TAB(X). SPC(X) représente X caractères espace. Cette fonction permet d'économiser de nombreux octets dans un programme. TAB(X) envoie au périphérique de sortie (usuellement l'écran) le nombre de <curseur à droite > nécessaire pour que le caractère suivant soit en position X dans la ligne d'impression.

X doit être compris entre 0 et 255 mais BASIC ne tient compte que de X modulo 40 c'est-à-dire le reste de la division entière de X par 40. Après avoir ainsi ramené la valeur entre 0 et 39, TAB fait avancer la position d'impression à cet endroit si c'est possible en avançant vers la droite. Sinon TAB passe à la ligne-écran suivante, puis avance ensuite du nombre de caractères demandé.

```
Exemple :  PRINT "?"; SPC(15); "?"
           PRINT "?"; TAB(15); "?"
```

La commande CMD définit quel est le périphérique de sortie par défaut . Au démarrage de l'interpréteur BASIC, l'équivalent de CMD3 est exécuté, c'est-à-dire que par défaut, le canal de sortie envoie les caractères à l'écran. Pour modifier la routine de sortie CHROUT et lui donner comme sortie un autre périphérique, il faut:

1. ouvrir un fichier déclarant ce périphérique avec OPEN X,...
2. attribuer à CHROUT le périphérique de ce fichier avec CMD X.

Exemple : lister sur l'imprimante :

```
OPEN1,4:CMD1:LIST
```

Exemple : lister sur un fichier disque :

```
OPEN 1, 8, 1, " 0:LISTING, SEQ, WRITE"
CMD 1 : LIST
```

Pour rétablir le périphérique par défaut comme étant

l'écran, il faut :

1. Vider le tampon de sortie par PRINT£X.

2. Refermer le fichier concerné par CLOSE X.

Si ce n'est pas exécuté, l'écran ne fonctionne pas correctement dans certains cas, en particulier après un listing sur les périphériques 2 ou 4.

Exemple : lister sur imprimante :

```
OPEN 1,4 : CMD 1 : LIST  <return>
PRINT£ 1 : CLOSE 1   <return>
```

Noter que LIST se terminant toujours par un retour au mode direct comme si on exécutait END, la seconde ligne de l'exemple ci-dessus doit donc être tapée en mode direct.

Les commandes **LOAD, SAVE, VERIFY** sont les commandes permettant le transfert des programmes d'un périphérique vers la mémoire du 64 et vice- versa. La syntaxe générale est identique pour les trois mots-clés : LOAD "PROG",8,1. Le premier argument est le nom de fichier, le second l'adresse primaire ou numéro du périphérique, le dernier l'adresse secondaire.

Le nom de fichier est obligatoire pour le disque et pas pour la cassette. Pour la cassette, lors de LOAD ou VERIFY , le nom de programme peut ne pas être complet.

Exemple : Si la cassette contient le programme "ESSAI", suivi du programme "EST",

LOAD"ES" chargera le programme ESSAI

LOAD"EST" chargera le programme EST.

En effet, le test d'identité des noms est seulement effectué sur le nombre de caractères présents dans l'instruction LOAD ou VERIFY.

Le numéro du périphérique peut être 1 ou un numéro entre 3 et 31. 1 est la cassette, les numéros supérieurs à trois doivent correspondre à des lecteurs de disquettes : un disque seul est usuellement en 8.

L'adresse secondaire peut être 0, 1, 2 ou 3. En pratique, il y a deux méthodes de chargement :

1. LOAD "E"

```
ou LOAD "E", 1          méthode 'normale'
ou LOAD "E", 1, 0

2. LOAD "E", 1, 1       méthode 'absolue'.
```

En mode normal, le programme est chargé en début de zone BASIC, à l'adresse qui est stockée en 43 et 44. Or sur la cassette, comme sur le disque, l'adresse de début de programme est inscrite au début du fichier programme. Il s'agit bien entendu de l'adresse de début du programme au moment du SAVE. En mode normal, BASIC charge le programme au début de sa zone de travail actuelle, sans tenir compte de Cl'adresse contenue dans le fichier-programme.

Par contre, en mode 'ABSOLU', le programme se chargera à l'adresse contenue dans le fichier, indépendamment de l'adresse de début de BASIC du 64 au moment du LOAD. Ceci est très utile pour sauver et recharger des images-écran, des générateurs de caractères, des programmes en langage machine.

Pour recharger un programme à l'adresse réelle où il était au moment du SAVE, il y a deux méthodes :

```
1. SAVE "TOTO", 8, 1
   POKE 44, adresse de chargement-partie basse
   POKE 45, adresse de chargement-partie haute
   LOAD "TOTO", 8

2. SAVE "TOTO", 8, 1
   LOAD "TOTO", 8, 1
```

On notera que la première méthode permet de modifier à volonté cette adresse de chargement.

Pour les SAVE sur cassette, les adresses secondaires 2 et 3 sont autorisées : 2 est semblable à 0 et 3 est semblable à 1, mais pour 2 et 3, le BASIC sauve les programmes avec un marqueur E.O.T ajouté à la fin, qui signifie 'fin de bande'. En conséquence tout LOAD, SAVE ou VERIFY qui découvre ce bloc E.O.T. arrête sa recherche, croyant atteindre la fin de bande physique.

L'instruction **VERIFY** est semblable au LOAD à l'exception de ceci : chaque octet lu, au lieu d'être rangé en mémoire, est comparé au contenu de la mémoire correspondante et l'octet ST-STATUS est mis à jour en fonction de la comparaison. A la fin du VERIFY, si le ST-STATUS est différent de 0, le message 'VERIFY ERROR' apparaît. Pour différencier les routines VERIFY et LOAD, le BASIC utilise un drapeau (flag) : l'adresse mémoire 147 contient 1 pour LOAD, 0 pour VERIFY.

LES FONCTIONS BASIC

Les fonctions BASIC représentent des valeurs, exactement au même titre que les variables. Les fonctions sont en fait des formules, des méthodes de calcul qui représentent une et une seule valeur définie par le (ou les) paramètre(s) de la fonction. Par souci d'uniformisation, toutes les fonctions ont au moins un argument, même si celui-ci n'est pas nécessaire, comme dans FRE. Les fonctions sont à regrouper en deux blocs : les fonctions numériques dont les valeurs sont en flottant et les fonctions chaînes dont les valeurs sont des chaînes de caractères. Les fonctions numériques ont toutes des arguments numériques sauf LEN, VAL et ASC qui ont un argument chaîne.

Les fonctions numériques ordinaires sont SGN, INT, ABS, SQR, LOG, EXP, COS, SIN, TAN et ATN. Ces fonctions ont un seul argument numérique (flottant ou entier) qui peut être une expression plus ou moins complexe. l'évaluation d'une expression est effectuée par la routine EVAL en $AEF1. Celle-ci est en fait la partie 'calculateur' de l'interpréteur BASIC. Les opérations élémentaires intervenant dans chaque expression sont évaluées dans l'ordre de priorité algébrique : parenthèses les plus intérieures d'abord, puis exposant, logarithmes, multiplication, etc...

Le résultat de chaque opération élémentaire est toujours présent dans l'accumulateur flottant FLP1 à la fin de la routine correspondante. Pour l'évaluation des opérations à deux arguments, l' ACCU FLP 2 est temporairement utilisé. Par exemple, SGN nécessite l'évaluation de la valeur de l'argument (BASIC vérifie qu'un et un seul argument est présent) à l'aide de la routine EVAL. Ensuite, la routine SGN teste la valeur de l'exposant de l'accu FLP1. Si celui-ci est nul, le résultat est 0 et la routine se termine avec 0 dans l'ACCU FLP1 que SGN laisse dès lors inchangé. Si l'exposant est différent de $00, la routine teste le bit de signe de la mantisse de l'ACCU FLP 1 et, suivant sa valeur, dépose +1 ou -1 dans l'ACCU FLP1. De même, la routine INT convertit la valeur en décimal, annule la partie fractionnaire et reconvertit le résultat en flottant. ABS met à 0 le bit 31 de la mantisse de l'accu FLP1.

Les 7 autres fonctions BASIC simples sont en fait 7 variantes d'une même routine. SQR, LOG, EXP, COS, SIN, TAN et ATN sont toutes évaluées par la méthode du polynôme. La fonction mathématique applique à l'expression évaluée dans l'accumulateur flottant un polynôme propre à chaque routine : chaque polynôme est de la forme AX+BX^2+CX^3+DX^4+...où X est la valeur de l'expression et A, B, C, D,... les constantes du polynôme (voir les adresses des constantes en fin de chapitre). Le premier octet de la table de valeurs d'un polynôme est l'ordre de ce polynôme, c'est-à-dire le nombre de termes. Le polynôme est évalué par la routine POLY en $E059. La méthode de calcul de COS et TAN est indirecte : en fait, BASIC n'évalue que le polynôme de sinus. Pour le cosinus, la routine COS, après évaluation de

l'expression, y ajoute la valeur pi/2, stockée en $E2E0 et évalue le sinus du résultat par SIN. Pour la tangente, BASIC exécute SIN, sauve le résultat dans une zone mémoire libre (en $4E), exécute ensuite COS puis divise la valeur du sinus en mémoire par celle contenue dans l'accu FLP1. Le résultat est à la fin de la routine dans l'accumulateur flottant et le tour est joué !

Il faut noter que, pour des raisons de précision, les mathématiques imposent l'usage de polynômes d'ordre impair.

La fonction arctangente, ATN, qui exige un nombre de termes important pour donner un résultat correct, possède l'ordre le plus élevé : Onze, ce qui explique que ce soit le mot-clé BASIC le plus lent à exécuter.

La fonction USR est le second interface après SYS, entre BASIC et langage machine. USR est une véritable fonction. Le paramètre de l'USR est évalué par EVAL et sa valeur est donc dans l'accu FLP1. Puis le contrôle est passé à la routine de l'utilisateur dont l'adresse (BAS-HAUT) doit être écrite en $311 et $312 (785 et 786). A la fin de la routine utilisateur. (RTS), la valeur contenue dans l'accu FLP1 est la valeur de la fonction. L'utilisateur peut donc créer sa propre fonction qui modifie l'ACCU FLP1.

La fonction FRE a un argument bidon (DUMMY) qui n'est là que par souci d'uniformisation. En effet, cette fonction n'a pas d'argument. Son but est d'évaluer la place disponible en mémoire pour le programme utilisateur; mais elle a aussi l'effet d'augmenter la mémoire effectivement disponible à un moment donné : elle provoque le "ramassage d'ordures" (GARBAGE COLLECTION). La fonction FRE vaut, après évaluation, le nombre d'octets libres en mémoire, soit la différence entre le pointeur bas de zone chaînes en $33 et le pointeur fin de tableaux en $31.

Le ramassage d'ordures effectué préalablement efface de la mémoire les zones chaînes inutilisées. Celles-ci sont nombreuses surtout si on a effectué de nombreuses concaténations.

EXEMPLE:

PROGRAMME	CONTENU DE LA ZONE CHAINES		
	BAS <-------MEMOIRE RAM------>		SOMMET
1 CLR			TOP
2 A$="COUCOU"			TOP
3 A$=A$		COUCOU	TOP
4 A$=A$+"!"	COUCOU !	COUCOU	TOP
5 A=FRE(0)		COUCOU !	TOP
6 END		COUCOU !	TOP

Dans l'exemple, la ligne 2 n'occupe aucune place dans la zone chaînes car la variable A$ contient un pointeur dirigé sur la chaîne de caractères contenue dans la ligne BASIC No2. La ligne 3 crée une nouvelle chaîne de caractères identique à la précédente mais située ailleurs (donc dans la zone chaînes). La ligne 4 crée une nouvelle chaîne plus longue, donc ailleurs que la précédente.

Notez que l'ancienne chaîne A$ est toujours présente mais le pointeur de A$ n'est plus dirigé sur elle : elle est inutilisée. La ligne 5 ramasse les ordures : elle supprime la chaîne inutile, remonte la chaîne utile vers le sommet de mémoire et remet à jour le pointeur de chaînes dans la variable A$. Après la fin d'exécution du programme, les variables encore utiles sont là. La taille totale des chaînes utiles peut être évaluée en notant la différence des pointeurs Haut et Bas de zone chaînes.

Taille des chaînes =

```
(PEEK(53)+256*PEEK(54))-(PEEK(51)+256*PEEK(52))
```

La fonction POS a également un argument bidon comme FRE. La valeur POS(0) est simplement la valeur de l'octet contenu en 211 ($D3). A noter que PEEK(214) ($D6) donne le numéro de la ligne-écran où se trouve le curseur, tandis que POS(0) ou PEEK (211) donne la position du curseur dans cette ligne. Il faut bien noter ici que par ligne on n'entend pas chaque rang horizontal de 40 caractères mais bien ligne BASIC qui , suivant les cas, occupe 1 ou 2 lignes de 40 caractères : une ligne BASIC peut avoir jusqu'à 80 caractères.

La fonction RND constitue un générateur pseudo-aléatoire, c'est-à-dire que le nombre 'au hasard' rendu par RND est en fait le résultat d'un calcul, d'un algorithme. Les valeurs RND ont une distribution statistique identique à celle d'un vrai tirage au hasard mais elles sont déterminées par un calcul et non un vrai hasard. Le paramètre d'entrée n'est pas une des données de l'algorithme de calcul mais plutôt un choix entre deux méthodes de calcul pour le point de départ de l'algorithme RND.

L'argument de départ de l'algorithme RND (SEED VALUE en anglais) est situé en 139 ($8B) , valeur en flottant sur 5 octets. Cette valeur est initialisée lors du premier appel à RND. Si le calcul de RND est effectué à partir de la valeur précédente RND, on dit que les valeurs font partie de la même séquence pseudo-aléatoire.

C'est ici qu'intervient l'argument de la commande RND(X). Avec X>0, la même séquence est exécutée à chaque allumage du 64. Avec X<0, l'argument de l'algorithme est recalculé et dépend de la valeur X, ce qui démarre une nouvelle séquence pseudo-aléatoire. Avec X=0, une nouvelle séquence est redémarrée mais l'argument de l'algorithme est une fonction de l'horloge interne du CBM64 (comptage en micro-secondes dans les registres 4,5,8 et 9 du CIA1 en $DC04,$DC05,$DC08 et $DC09).

Pour obtenir un effet de hasard réaliste dans un programme, il faut employer :

```
RND(0)    la première fois
RND(.5)   les fois suivantes.
```

Il est en effet faux de croire qu'un meilleur (?) hasard est obtenu en employant chaque fois une séquence pseudo-aléatoire différente. Chaque séquence pseudo-aléatoire donne 65536 nombres différents , donc les risques de reconnaître un morceau de séquence 'de mémoire' sont faibles.

La fonction PEEK lit le contenu d'une adresse mémoire. La valeur de PEEK(X) est donc comprise entre 0 et 255. Comme toutes les fonctions, sa valeur est donnée en flottant. Pour éviter des erreurs d'arrondi dues au mode flottant, les manipulations de valeurs OCTETS ont intérêt à être rangées dans des variables entières. En effet, BASIC dans ce cas effectue les arrondis nécessaires avec exactitude.

```
Exemple : A%=PEEK(144) : PRINT "ST="A%
```

NOTE : PEEK ne lit que les mémoires en service au moment de la commande. Pour lire la mémoire RAM sous les ROMS BASIC ou KERNAL, il convient d'utiliser le programme mettant les ROM hors service au bon moment (voir le programme USR-PEEK).

LEN, VAL et ASC sont trois fonctions numériques dont les arguments sont des variables-chaînes. LEN(A$) donne la longueur de la chaîne argument lue dans le descripteur de la variable, c'est le 3me octet des 7 qui définissent la chaîne. LEN est toujours compris entre 0 et 255 inclus.

ASC(A$) prend le premier caractère de la chaîne et donne sa valeur numérique c'est-à-dire son code ASCII . Attention, la chaîne argument ne peut pas être vide ! VAL(A$) convertit une chaîne de caractères représentant un nombre en une valeur flottante. Pour être certain d'éviter la chaîne nulle, on peut utiliser:

```
A=ASC(A$+CHR$(0))
```

Exemples de chaînes pouvant convenir POUR VAL :

```
10
10E-3
-2E-05
+10
.2E4
```

Les fonctions chaînes ont des valeurs chaînes : leur nom se termine donc par un $, comme pour les variables.

STR$ et CHR$ sont des instructions de conversion : STR$ convertit la valeur flottante d'une expression en chaîne de caractères . CHR$ crée une chaîne de longeur unitaire, l'argument numérique étant considéré comme le code ASCII du caractère.

LEFT$, RIGHT$ et MID$ sont des fonctions d'extraction. Ces trois fonctions ont un argument chaîne(A$) suivi d'un argument numérique (X).(2 pour MID$: Y et X). La valeur de la fonction est une sous-chaîne de A$ dont la longueur est X. L'argument supplémentaire de MID$ indique à quelle position commence la sous-chaîne. Il peut être omis: dans ce cas, BASIC prendra la fin de la chaîne.

+,-,*,/,^,- monadique sont les opérateurs algébriques classiques. Tous sauf le dernier possèdent deux opérandes situés respectivement à gauche et à droite du signe opératoire. A noter la différence entre le - de soustraction qui possède deux opérandes (on soustrait la seconde de la première) du - monadique de négation qui ne fait que changer le signe de l'opérande.

AND,OR et NOT :ces opérateurs logiques ont un double but : les manipulations logiques de valeurs entières et l'utilisation dans les instructions conditionnelles ou dans les expressions conditionnelles.

Cependant à cause de la valeur des opérandes conditionnels - 0=FAUX ; <>0 = VRAI - les opérations à effectuer dans les deux cas sont identiques. Ces opérations portent toujours sur des entiers signés sur deux octets, c'est-à-dire que les opérandes sont compris entre +32767 et -32768, et sont codés sur seize bits dont le bit 15 sert à indiquer le signe (1 : négatif, 0 : positif). Les instructions AND et OR sont très souvent utilisées pour modifier des bits à l'intérieur d'un octet donné.

Illustrons ceci par un exemple : soit A une valeur inconnue codée sur 8 bits; (ce peut être le contenu d'un registre du circuit VIC-II ou d'un CIA par exemple), dont on désire positionner le bit 3 à 1. Cette opération consiste à prendre la valeur des 7 bits à ne pas modifier et à superposer à cette

valeur un octet dont seul le bit 3 vaut 1 (valeur = 8). Dans l'exemple ci-dessous, A vaut 230. Le masque, c'est-à-dire l'octet où tous les bits que l'on doit conserver valent 1, vaut donc 255-8=247.

No du bit	7	6	5	4	3	2	1	0
Valeur du bit	128	64	32	16	8	4	2	1

A	=	1	1	1	0	0	1	1	0	=230
MASQUE	=	1	1	1	1	0	1	1	1	=247
A AND MASQUE	=	1	1	1	0	0	1	1	0	=230
VALEUR DU BIT	=	0	0	0	0	1	0	0	0	= 8
APRES OR	=	1	1	1	0	1	1	1	0	=238

En BASIC : A = A AND 247 OR 8

Pour mettre le bit 3 à 0, on aurait simplement écrit :

A = A AND 247 OR 0 ou A = A AND 247

On voit donc que pour positionner des bits à des valeurs quelconques dans un octet, il suffit d'une ligne BASIC avec AND suivi de OR.

Forme générale : A = B AND C OR D

Ces opérations doivent être bien comprises car elles servent à tout moment dans l'établissement des images graphiques dans le CBM 64.

Les opérateurs de comparaison < = > sont très utiles car très employés (regardez entre IF et THEN !). (A<B) est une valeur : on dit une valeur logique car (A<B) ne peut prendre que deux valeurs: -1 (= vrai) ou 0(= faux). Les opérateurs de comparaison peuvent être groupés par deux comme >= pour signifier plus grand ou égal, ou <> pour signifier différent.

On emploie les opérateurs de comparaisons dans les instructions conditionnelles comme IF...THEN ou dans les expressions conditionnelles. Celles-ci sont très utiles bien que peu employées. L'exemple ci-après montre à l'évidence leur intérêt.

Exemple :

On désire obtenir dans A une des valeurs 4, 7, 17 ou 41 et ce, suivant que la variable B est égale à 37, 36, 128 ou 0.

La méthode habituelle donnera :

```
10  IF B=37  THEN A=4  : GOTO 50
20  IF B=36  THEN A=7  : GOTO 50
30  IF B=128 THEN A=17 : GOTO 50
40  IF B=0   THEN A=41
50  .....
```

ou quelque chose d'approchant. Il est bien plus élégant et plus efficace d'écrire :

```
A=-(4*(B=37)+7*(B=36)+17*(B=128)+41*(B=0))
```

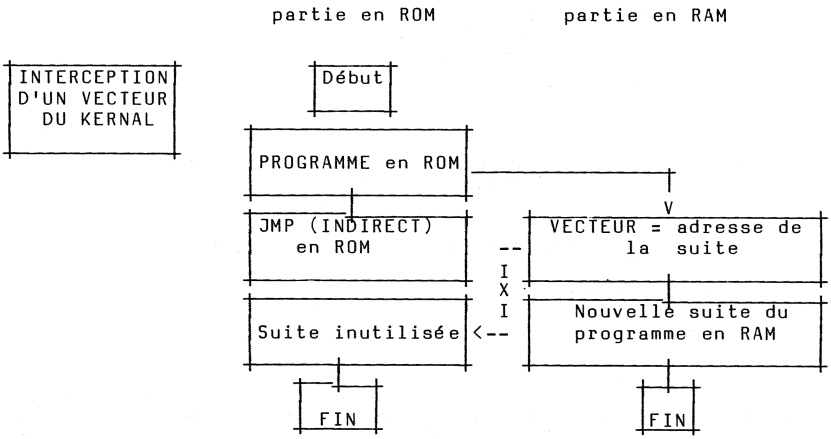
Faire plus avec BASIC

Les périphériques les plus spectaculaires du CBM 64 ne sont pas accessibles en BASIC. C'est pourquoi de nombreux programmes écrits en langage machine agrandissent le vocabulaire BASIC : GRAPHIC PAK, SIMON'S BASIC, par exemple. (Voir extensions logicielles).

Pour ajouter vous-même des fonctions au CBM 64, ou simplement pour mieux comprendre son fonctionnement, il est utile de connaître les routines du BASIC. On en trouvera une description ci-dessous. Il est certain que l'usage d'un programme moniteur comme celui décrit dans cet ouvrage est un avantage majeur pour l'expérimentation dans ce domaine.

L E K E R N A L

Le KERNAL est un programme ou plutôt un ensemble de programmes, qui contrôle les diverses actions du CBM64, c'est l'"**OPERATING SYSTEM**". Le terme KERNAL vient d'un mot signifiant 'NOYAU' en anglais. Le noyau de sous-programmes occupe 7K en mémoire morte ROM aux adresses $E500 à $FFFF et est contenu dans une des trois mémoires ROM du CBM64.



Les programmes internes du 64, comme ceux du VIC, comprennent quasiment toutes les routines déjà présentes dans le PET, mais organisées de manière bien plus rationnelle. L'interpréteur BASIC est entièrement séparé des routines système, ce qui permet une adaptation bien plus aisée à d'autres langages. De plus, toutes les routines importantes pour l'utilisateur passent par un vecteur en mémoire RAM où l'on peut les intercepter.

Les vecteurs sont écrits en RAM par le KERNAL lui-même à l'allumage du 64 mais ils sont modifiables très aisément par le programmeur. DE plus, la ROM contenant le KERNAL peut être à volonté retirée avec la ROM BASIC de l'espace d'adressage.

Un parti que l'on peut tirer de cet état de fait est la possibilité de recopier le KERNAL en RAM et de le modifier en-

suite suivant les besoins du moment. Par exemple, on peut voir en annexe le texte du programme "AZERTY" qui modifie la partie du KERNAL qui décode le clavier.

Si l'interpréteur BASIC a besoin des routines KERNAL, en revanche le contraire n'est pas vrai. Toutes les sous-routines d'entrée sortie vers l'écran, le clavier, la cassette, etc... sont situés dans le KERNAL. Les plus importantes de ces routines sont accessibles aux adresses de la table de JMP située en $FF81 à $FFF5 : il s'agit de 39 sauts (JMP ou JMP indirect) vers les routines principales du KERNAL. La plupart de ces routines portent un nom attribué par COMMODORE ou par certains programmeurs (J. BUTTERFIELD, N. HAMPSHIRE, etc).

Nous reprenons ici les noms les plus courants. Une des routines du KERNAL est cependant particulière : la routine CHRGOT prend un caractère dans le texte stocké en mémoire (normalement un programme BASIC) à l'endroit du pointeur de caractère, puis elle incrémente ce pointeur. Cette routine est d'usage extrêmement fréquent, et pour être très rapide doit être située en page 0 en RAM, ce qui la rend ainsi auto-modifiable -voir plus bas- et également aisément modifiable par un programme extérieur au KERNAL. Les programmes comme l' interpréteur de langage FORTH (un langage rapide fort différent du BASIC) , le **DOS SUPPORT** ou des programmes ajoutant des commandes supplémentaires au BASIC comme SIMON'S BASIC utisent tous des versions propres de la routine CHRGOT.

AUTO-START

Au démarrage, le KERNAL exécute une série d'opérations d'initialisation. L'adresse de la routine de RESET est toujours, avec le 6510, située en $FFFC et $FFFD.Pour le CBM64,le contenu de cette adresse est $FCE2.

A ce moment, le KERNAL initialise le pointeur de pile à $01FF, supprime le mode décimal et interdit les interruptions. Tout de suite après, le KERNAL teste la présence en $8000 d'une ROM spéciale dite à démarrage automatique (AUTO-START). Cette ROM, si elle est présente, doit contenir pour que le KERNAL lui passe le contrôle du 64 la suite de caractères $C3, $C2, $CD, $38, $30 (CBM80 en ASCII -avec les lettres C,B et M ayant le bit 7 à 1) à l'adresse $8004.

Si ce n'est pas le cas, le KERNAL initialise tous les vecteurs et pointeurs en RAM, réautorise les interruptions, puis effectue un JMP indirect en $A000, pour initialiser l'interpréteur BASIC, ce que l'on peut toujours faire à la main en exécutant le code JSR $FCFF ou SYS(64767). Pour redémarrer un

programme AUTO-START dont l'effet a été ainsi annulé, effecuez un JSR $FCEC ou SYS(64748) qui contient le code JMP($8000).

On peut annuler l'effet d'une ROM à démarrage automatique en rétablissant les pointeurs comme lors d'un démarage normal (Voir le programme SIBOFF qui met hors service la ROM SIMON'S BASIC). Ceci n'est bien entendu possible que si l'on a encore la possibilité de programmer le CBM 64. Par exemple, avec les jeux en cartouche,ce n'est pas possible et il faut bien alors éteindre l'ordinateur et retirer la cartouche.

Les ROM AUTO-START contiennent en $8000 et $8001 l'adresse du début de programme (COLD START). Les adresses $8002 et $8003 contiennent une adresse de redémarrage du même programme (WARM START) utilisée quand on emploie les touches STOP-RESTORE.

Pour démarrer le CBM avec une ROM auto-start sans qu'elle ne s'exécute, il faur couper momentanément le fil 11 du connecteur d'extension à l'intérieur de la cartouche. Ensuite, allumer le 64, puis rétablir (avec un interrupteur) cette connection - c'est le fil de sélection de la ROM $A000-. A ce moment, il n'y a plus de problèmes, car le KERNAL n'écrit ni ne lit plus rien aux adresses $8000 à $9FFF dès la fin de l'initialisation du BASIC. Ce genre de manipulation n'est à exécuter que par des personnes possédant de bonnes notions d'électronique digitale.

Désassembler et comprendre le fonctionnement de programmes AUTO-START est très instructif : on voit ainsi comment modifier le BASIC pour lui ajouter des mots-clés, comment faire des jeux animés, colorés, etc...

Les adresses RAM utilisées par le KERNAL sont décrites dans la carte-mémoire.

DESCRIPTION DES PRINCIPALES ROUTINES DU KERNAL.

Notons que pour essayer l'une ou l'autre de ces routines à partir du BASIC, le 64 procure au programmeur d'importantes facilités. En effet, les arguments qui doivent être présents dans les registres du 6510 à l'appel d'un programme KERNAL par l'instruction BASIC SYS, peuvent être préparés par chargement aux adresses 780 et suivantes (voir SYS).

Passons maintenant en revue les principales routines du BASIC et du KERNAL.

Les routines décrites ci-dessous sont précédées de leur adresse donnée en HEXADECIMAL, car elles ne sont la plupart du temps utilisables qu'en assembleur. La conversion de ces adresses en décimal pourra s'effectuer à l'aide du programme 'HEXA' en annexe.

PAGE ZERO - ROUTINE CHRGOT

JSR $0073 ou JSR CHRGOT : Appel de la routine de lecture du caractère courant dans le texte BASIC. Cette routine incrémente le pointeur de caractére en $007A et $007B dans le corps de la routine elle-même

JSR $0079 ou JSR CHR1 : Appel de la même routine sans incrémentation du pointeur de caractères.

```
$0073 CHRGOT    INC POINTEUR     ;METS A JOUR LE POINTEUR DE
$0075           BNE CHR1         ;CARACTERES.
$0077           INC POINTEUR+1
$0079           LDA xxxx         ;PRENDS LE CARACTERE.
      POINTEUR  =   $007A
$007C           CMP £$3A         ;TESTE SI CARACTERE > '9'.
$007E           BCS FIN
$0080           CMP £$20         ;IGNORE TOUS LES BLANCS.
$0082           BEQ CHRGOT
$0084           SEC              ;TESTE SI CARACTERE < '0'.
$0085           SBC £$30
$0087           SEC
$0088           SBS £$D0         ;RESTAURE LE CARACTERE LU
$008A FIN       RTS              ;A SA VALEUR EXACTE DANS A.
```

(Noter l'astuce: VALEUR-$20-$30-$D0 = VALEUR)

Un appel par JSR $0073 incrémente le pointeur de l'interpréteur BASIC et mets le caractère dans l'accumulateur A du 6510. En sortie le flag CARRY (flag C) vaut 1 si le caractère est un chiffre, le flag ZERO (flag Z) vaut 1 si le caractére est un $00. Le pointeur de caractère est en $007A et $007B dans le corps même de la routine. Un appel par JSR $0079 fait la même chose sans avancer le pointeur, ce qui est très pratique pour que l'accumulateur et les flags puissent refléter l'état du caractère courant avant d'appeller une des routines de l'interpréteur BASIC.

Il est possible d'installer un 'WEDGE', c'est-à-dire de modifier CHRGOT, pour tester la présence de caractères spéciaux, par exemple le '>' du programme DOS SUPPORT.

Dans ce cas,on déposera un JMP$xxxx vers la routine supplémentaire de test en $007C. La routine de test insérée dans CHRGOT, doit bien entendu, supléer aux octets que le JMP a remplacé. Le JMP peut aussi remplacer SBC et RTS en $0088. Le code de la routine CHRGOT est rangé initialement en ROM en $E3A2 à $E3B9. Il est recopié en RAM en page zéro à l'allumage de la machine.

ROM $A000 - INTERPRETEUR BASIC

$A00C — Table des 76 adresses des mots-clé BASIC. Les adresses - 1 sont en fait données car l'appel des programmes correspondants se fait par l'instruction RTS.

$A09E — Table des 76 mots-clé du BASIC. La fin de chaque mot-clé est repérée par le bit7 du caractère qui vaut 1. Noter que si on modifie cette table après l'avoir recopiée en RAM, elle ne peut dépasser une longueur de 256 octets car elle est utilisée par des instructions indexées.

$A19E — Table des messages d'erreur du BASIC. Le bit 7 du dernier caractère d'un message est à 1. On trouvera un exemple de modification de cette table dans le programme BASIC-FR en annexe.

$A328 — Table des 30 adresses des messages d'erreurs du BASIC. Voir la table en $A19E.

JSR $A3BF — Déplace un bloc en mémoire.
Pointeur de début de source en $5F, $60 (partie basse, partie haute).
Pointeur de fin de source en $5A, $5B.
Pointeur de fin de destination en $58, $59.
Le sommet de bloc est le premier déplacé. Cette routine est à utiliser pour transférer un bloc de mémoire si l'adresse de la destination est plus haute que l'adresse de la source. elle est aussi utilisable si source et destination n'ont aucune adresse commune.
En sortie, les registres X et Y sont remis à 0.

JSR $A3FB — Teste s'il reste au moins 62 octets libres dans la pile et si non,imprime le message 'OUT OF MEMORY ERROR' et retourne ensuite à BASIC READY.

JSR $A435 — Affiche 'OUT OF MEMORY ERROR' et retourne au BASIC READY.

LDX,errnum
JSR $A437 — Imprime le message d'erreur numéro errnum (voir la table en $A328 qui contient les adresses successives des messages) et retour au BASIC READY.

JMP $A483 — Entrée normale vers BASIC READY aussi appellée 'démarrage à chaud du BASIC' ou WARM BOOT en anglais.

JSR $A644 — Exécute le "NEW" du BASIC.

JSR $A660 — Execute le "CLR" du BASIC.

JSR $A67A — Remet le pointeur de pile à sa valeur de départ, et revient malgré cela au programme appellant. Routine trés utile en début d'un programme contenant de nombreuses sous-routines et ne devant revenir finalement qu'à BASIC READY.

JSR $A6C9 — Effectue le "LIST" du BASIC. Avant l'appel du programme, il faut ranger le numéro de la première ligne BASIC à lister en $5F et $6F (partie basse,partie haute) et le numéro de la dernière ligne BASIC à lister en $14 et $15. Rend toujours le contrôle à la routine 'BASIC READY'.

JSR $A81D — Execute le "RESTORE" du BASIC.

JSR $A831 — Exécute le "END" du BASIC.

JSR $A871 — Exécute le "RUN" du BASIC. Ne revient jamais au programme appellant,mais bien à 'BASIC READY'.

JSR $A96B — Acquiert une valeur décimale dans le corps du programme BASIC. Doit faire partie de la sous-routine SYS xxx qui précède le nombre. Le résultat est sur deux octets en $14 et $15. Lors de l'appel, l'accumulateur et les flags doivent refléter l'état du premier caractère du nombre comme en sortie de n'importe quelle routine de l'interpréteur BASIC. C'est-à-dire que l'accumulateur doit contenir le dernier caractère que la routine CHRGOT a saisi dans le corps du texte BASIC.
De même, les flags CARRY et ZERO doivent indiquer, comme en sortie de CHRGOT, l'état numérique ou nul du caractère . En sortie, l'accumulateur et les flags reflètent l'état du premier caractère non-numérique qui suit la valeur acquise dans le

texte. De plus, si la valeur lue est supérieure à 63999, alors le retour s'effectue à 'BASIC READY' avec le message d'erreur "SYNTAX ERROR".

JSR $A9C4

Convertit le contenu de l' accumulateur flottant principal en entier sur deux octets et le résultat est mis sous la forme (partie basse, partie haute) à l'adresse indiquée par le pointeur en page zéro en $49 et $4A.

JSR $AAAO

Exécute le "PRINT" du BASIC . Comme toute instruction BASIC, l'accumulateur et les flags doivent refléter l'état du premier caractère de la liste d'arguments. A employer dans un SYS xxx "COUCOU", par exemple. Respcte tous les codes de déplacement de curseur, de changement de couleurs, etc...
Peut être employé pour imprimer n'importe quelle liste d'arguments tels que chaînes de caractères, variables BASIC.. à l'intérieur d'un programme appellé par SYS ou USR. La liste des paramètres doit suivre immédiatement l'instruction SYS ou USR dans le corps du texte BASIC.

JSR $AB1E

Ecrit une chaîne de caractères dont le pointeur est dans les registres X et Y (partie basse dans X, partie haute dans Y).La chaîne de caractères doit se terminer par un octet 00.

```
Exemple : LDX £<MESSAG
          LDY £>MESSAG
          JSR $AB1E
          RTS
MESSAG    .BYTE 'COUCOU',$00
```

LDA £LEN
JSR £AB21

Ecrit une chaîne de caractères sur le périphérique courant. L'adresse de la chaîne doit être rangée en $22 et $23 et la longueur de la chaîne doit être dans l'accumulateur A lors de l'appel de la routine. La chaîne ne peut dépasser une longueur de 255 caractères.

$ACFC

Message "EXTRA IGNORED" suivi de $00.

$ADOC

Message "REDO FROM START" suivi d'un $00.

JSR $AD8A

Routine à utiliser pour prendre dans le corps du texte BASIC une valeur numérique comprise entre 0 et 65535. En sortie, le résultat se retrouve en $14 et $15 sous la forme d'un nombre entier non signé sur deux octets. Dans ce cas, l'appel de la routine se fera par:

```
JSR $AD8A
JSR $B7F7
RTS
```

JSR $AD9E — Prends une valeur numérique dans le texte BASIC, là où est le pointeur courant de CHRGOT. A utiliser à partir de SYS xxx suivi de paramètres. Le résultat est en sortie dans l'accumulateur flottant principal. Cette routine accepte toute expression BASIC, même très complexe utilisant les parenthèses,+,-,*,/,^,des variables,etc...

JSR $AE86 — Cette routine acquiert dans le texte BASIC une valeur numérique à l'endroit du pointeur courant de CHRGOT.La valeur à acquérir ne peut être qu'une variable simple ou le caractère 'PI'. En sortie, le résultat est située dans l'accumulateur flottant principal.(l'accu FLP.)

$AEA8 — Valeur du nombre 'PI' en flottant sur 5 octets.

JSR $AEF7 — Cette routine teste si le caractère courant est une parenthèse fermée et , si oui, incrémente le pointeur de caractère (dans CHRGOT). Si non, elle affiche le message "SYNTAX ERROR" et retourne au 'BASIC READY'. En entrée, l'accumulateur A doit contenir le caractère courant. Les flags doivent également refléter son état comme en sortie de CHRGOT.

JSR $AEFA — Même routine que ci-dessus mais pour vérifier la présence dans le texte BASIC de la parenthèse ouverte.

JSR $AEFD — Même routine que ci-dessus mais pour vérifier la présence dans le texte BASIC de la virgule. Il est indispensable d'utiliser cette routine si on désire acquérir dans le texte BASIC plusieurs paramètres dans une routine du genre: SYS xxx,12,34,56 par exemple. On utilisera alors alternativement les routines $AD9E et $AEFD pour lire les valeurs et les virgules de séparation, indispensables pour discerner l'un de l'autre les paramètres.

JSR $AF08 — Ecrit à l'écran le message "SYNTAX ERROR".

JSR $AF28 — Prends le nom d'une variable BASIC à l'endroit du pointeur de caractère courant (dans CHRGOT) et range la valeur de cette variable numérique dans l'accumulateur flottant.

JSR $AF84 — Cette routine met la valeur des trois octets de l'horloge interne TI dans l'accumulateur flottant principal en $63, $64 et $65.

JSR $B113 Cette routine teste si l'octet présent dans l'accumulateur A du 6510 est bien alphabétique. Si oui, elle met à 1 le flag C (CARRY). Si non, elle le met à 0.

JSR $B1AA Conversion d'un nombre rangé en notation flottante dans l'accumulateur flottant principal en nombre entier sur deux octets dans les registres A et Y (partie basse, partie haute).

JSR $B1BF Cette routine convertit un nombre flottant dans dans l'accumulateur flottant principal en entier signé sur deux octets (de valeur comprise entre -32768 et +32767)

JSR $B37D Cette routine exécute la commande BASIC FRE(0). Si la valeur rangée en $0D est à zéro, la routine ne désalloue pas l'espace mémoire réservé aux chaînes de caractères. En sortie, l'accumulateur flottant principal contient le nombre d'octets disponibles pour le BASIC. L'octet d'adresse $0D est de plus remis d'office à 0.

JSR $B3A2 Cette routine convertit la valeur entière contenue dans le registre Y du 6510 en nombre flottant dans l'accumulateur flottant principal.

JSR $B3A6 Cette routine teste si l'interpréteur BASIC est en mode DIRECT ou en mode PROGRAMME et si il est en mode direct, elle envoie à l'écran le message "ILLEGAL DIRECT ERROR" et retourne à BASIC READY.

JSR $B3AE Cette routine écrit "UNDEF'D FUNCTION ERROR" et retourne à BASIC READY.

JSR $B79E Cette routine acquiert dans le texte BASIC par appel à CHRGOT une valeur numérique comprise entre 0 et 255 et range le résultat dans le registre X du 6510. Si le nombre acquis est hors normes, le message "ILLEGAL QUANTITY ERROR" sera imprimé et le contrôle rendu à BASIC READY.

JSR $B7EB On acquiert dans le texte BASIC une valeur numérique comprise entre 0 et 65535 (inclus) et la range en page zéro aux adresses $14 et $15. ensuite, le test du caractère suivant sera effectué. Ce doit obligatoirement être une virgule. Ensuite, on acquiert une seconde valeur numérique comprise entre 0 et 255 inclus. Cette dernière valeur se retrouve dans le registre X du 6510. Cette routine est entre autres utilisée par la commande POKE.

JSR $B7F7 Cette routine convertit un nombre flottant dans l'accumulateur flottant principal en nombre entier non signé sur deux octets. Le résultat se retrouve en $14 et $15 en page zéro. Très importante routine pour acquérir la valeur du paramètre transmis du BASIC à un sous-programmee USR(xxx) écrit en langage machine.

JSR $B80D Evalue la fonction PEEK du BASIC. Il faut, en entrée, avoir dans l'accumulateur flottant principal la valeur de l'argument de la fonction. Le résultat est, en sortie, rangé en flottant dans l'accumulateur flottant principal.

JSR $B824 Exécute la commande POKE avec en entrée l'adresse en $14 et $15 (partie basse, partie haute) et la valeur à transférer en mémoire dans le registre X.

JSR $B82D Execute la commande WAIT avec l'adresse en $14 et $15 (partie basse, partie haute) et la valeur du second paramètre dans le registre X du 6510.Prends ensuite le caractère courant dans le texte BASIC pour le considérer comme troisième paramètre du WAIT, la valeur de masque.

JSR $B849 Cette routine ajoute la valeur 0.5 en flottant au contenu de l'accumulateur flottant principal.

JSR $B850 Cette routine soustrait d'une valeur flottante rangée en mémoire et dont le pointeur est dans les registres A et Y (partie basse, partie haute) la valeur contenue dans l'accumulateur flottant principal. Le résultat de la soustraction se retrouve en sortie dans l'accumulateur flottant principal.

JSR $B867 Cette routine ajoute au contenu de l'accumulateur flottant principal la variable dont le pointeur est constitué par les registres A et Y (partie basse, partie haute). Le résultat de l'opération se retrouve dans l'accumulateur flottant secondaire. Les deux accumulateurs flottants sont utilisés.

$B9BC Valeur 1 en flottant sur 5 octets.

$B9D6 Valeur 0.5 * SQR (2) en flottant sur 5 octets.

$B9DB Valeur SQR (2) en flottant sur 5 octets.

$B9E0 Valeur - 0.5 en flottant sur 5 octets.

$B9E5 Valeur LOG (2) en flottant sur 5 octets.

JSR $BA8C — Cette routine copie le nombre flottant pointé par les registres A et Y (partie basse, partie haute) dans l'accumulateur flottant secondaire en $6A à $6F.

JSR $BAE2 — Effectue la multiplication par 10 de la valeur rangée dans l'accumulateur flottant principal. En sortie, le résultat est rangé dans l'accumulateur flottant principal.

$BAF9 — Valeur 10 en flottant sur 5 octets.

JSR $BAFE — Effectue la division de la valeur rangée dans l'accumulateur flottant principal par 10. En sortie le résultat est rangé dans l'accumulateur flottant principal.

JSR $BB0F — Cette routine divise le nombre flottant pointé par les registres A et Y (partie basse, partie haute) par la valeur du contenu de l'accumulateur flottant principal. En sortie, le résultat est rangé dans l'accumulateur flottant principal.

JSR $BBA2 — Réalise la recopie du nombre flottant pointé par les registres A et Y (partie basse, partie haute) dans l'accumulateur flottant principal.

JSR $BBC7 — Réalise la recopie de la valeur contenue sur 5 octets dans l'accumulateur flottant principal dans les 5 octets de mémoire dont le pointeur est rangé en page zéro en $5C et $5D (partie basse, partie haute).

JSR $BBCA — Routine identique à la précédente mais avec le pointeur de destination rangé en $57 et $58.

JSR $BBD4 — Routine identique à la précédente mais avec le pointeur de destination rangé dans les registres X et Y (partie basse, partie haute).

JSR $BBFC — Cette routine recopie le contenu de l'accumulateur flottant secondaire dans l'accumulateur flottant principal.

JSR $BC2B — Cette routine réalise le test de signe de l'accumulateur flottant principal. En sortie, si la valeur était nulle , l'accumulateur A du 6510 vaudra $00. Si elle est positive, il vaudra $01 et si elle est négative, il vaudra $FF (soit -1 en notation complément à 2).

JSR $BC3C — Cette routine prends la valeur contenue dans l'accumulateur A du 6510 et la convertit en entier signé dans l'accumulateur flottant principal.

JSR $BC58 Impose le signe positif à la valeur de l'accumulateur flottant principal.

JSR $BC5B Comparaison de l'accumulateur flottant principal avec le nombre flottant dont le pointeur est rangé dans les registres A et Y. (partie basse, partie haute). Si les valeurs comparées sont identiques, l'accumulateur A du 6510 contiendra 0 en sortie. Si l'accumulateur flottant est plus petit, A contiendra $FF et s'il est plus grand, A contiendra $01.

JSR $BCE9 Cette routine remet le contenu de l'accumulateur A sous forme d'un entier sur 4 octets dans la mantisse de l'accumulateur flottant principal en $62 à $65.

JSR $BD7E Cette routine réalise l'addition de l'accumulateur A du 6510 considéré comme un nombre entier signé de valeur -128 à +127 avec le contenu de l'accumulateur flottant principal. Le résultat est en sortie dans l'accumulateur flottant principal.

$BDB3 Valeur en flottant de 999999,9

$BDB8 Valeur en flottant de 99999999

$BDBD Valeur en flottant de 1000000000

JSR $BDCD Cette routine écrit sur le périphérique de sortie courant (l'écran si l'affectation du périphérique de sortie n'a pas été modifiée par l'instruction CMD) la valeur entiére sur deux octets contenue dans les registres A et X du 6510 (partie basse, partie haute) après l'avoir converti en une chaîne de caractères. La chaîne comprend un blanc ou le signe - en première position.

JSR $BDDD Conversion de la valeur contenue dans l'accumulateur A du 6510 considérée comme un entier non signé compris entre 0 et 255 en chaîne de caractères. En sortie, le résultat se retrouve aux adresses $0100 et suivantes. La chaîne générée se termine par un $00. De plus, en sortie, les registres A et Y pointent vers le début de la chaîne de caractères (partie basse, partie haute) soit $00 dans A et $01 dans Y.

$BF11 Valeur en flottant de 0,5

$BF16 Table des valeurs des puissances de 10 en flottant sur 5 octets:

```
- 100000000
+10000000
 -1000000
  +100000
    +1000
     -100
      +10
       -1
```

$BF52 à $BF70 Cette zone de la mémoire N'est utilisée ni par BASIC ni par le KERNAL. Si vous recopiez le BASIC en RAM, il possible d'utiliser cette zone pour de petites corrections. Longueur de cette zone : 30 octets.

JSR $BF71 Cette routine calcule la racine carrée de la valeur de l'accumulateur flottant principal. En sortie, le résultat est dans l'accumulateur flottant principal.

$BFE1 Valeur en flottant de 1

ROM $E000 - SUITE DE L'INTERPRETEUR BASIC

JSR $E059 Cette routine calcule une fonction considérée comme un développement en série polynomiale et est utilisée pour les fonctions BASIC SIN, COS, TAN, EXP, etc...
En entrée ,les registres A et Y pointent vers la table des coefficients Pn du polynôme et l'accumulateur flottant principal contient la valeur x de la variable en flottant.En sortie, le résultat Y qui est la valeur calculée de F(x) se retrouve dans l'accumulateur flottant principal.
Le calcul de Y = F(x) est réalisé comme suit :

Y = P0+P1*X+P2*X^2+P3*X^3+...+Pn*X^n

La table des coefficients contient les valeurs de n et des Pn de l'équation ci-dessus. Sa structure est la suivante:

ler octet : Nombre de valeurs aussi appelé degré du polynôme.Cette valeur doit toujours être impaire pour des raisons mathématiques.

Octets 2 à 6 : Coéfficient n (le coéfficient multiplicateur de x^n) codé en flottant sur 5 octets.

Octets 7 à 11 : Coéfficient n-1 (celui de x^(n-1)) codé en flottant sur 5 octets.

Octets 12 à ....

Et ainsi de suite jusqu'au (5*n+1)ème octet qui sera la fin du coéfficient PO qui est le terme indépendant de l'équation.

JSR $E097 Cette routine effectue la fonction RND du BASIC. Le paramètre d'entrée et la valeur calculée en sortie sont tous deux dans l'accumulateur flottant principal.

JSR $E12A Cette routine réalise la fonction SYS du BASIC. L'adresse de retour est celle de l'interpréteur BASIC et elle est poussée dans la pile avant d'effectuer le saut vers la routine dont l'adresse est communiquée dans le paramètre. Cette adresse de retour est $E146. Cette fonction BASIC est la seule à accepter un argument numérique non limité par des parenthèses. Avant de passer à la routine en langage machine de l'utilisateur, SYS effectue les transferts suivants:

Ranger dans le registre A la valeur en $030C.
Ranger dans le registre X la valeur en $030D.
Ranger dans le registre Y la valeur en $030E.
Ranger dans le registre d'état (les flags) la valeur en $030F.

Le transfert à la routine de l'utilisateur s'effectue par un saut indirect à l'adresse qui a été convertie en provenance du texte BASIC et rangée en $14 et $15 (partie basse, partie haute). Donc, une routine appellée par SYS peut relire l'adresse de son point d'entrée en $14 et $15. Une sortie de SYS par une instruction RTS ramène normalement à l'adresse $E147 qui mets à jour les valeurs en $030C à $030F avec les contenus des registres en fin de la routine de l'utilisateur. Si le BASIC a été recopié en RAM, on peut intercepter la routine en la modifiant en $E130 où le paramètre du SYS est converti en entier. On peut aussi l'intercepter en $E147 pour récupérer la sortie de la routine SYS.

JSR $E156 Cette routine exécute le SAVE du BASIC en prenant les paramètres dans le corps du texte BASIC à

l'endroit du pointeur de CHRGOT.

JSR $E165 Exécution de la commande VERIFY du BASIC.

JSR $E168 Exécute la fonction LOAD du BASIC et affiche un message éventuel si nécessaire.

JSR $E200 Cette routine teste la présence d'une virgule dans le corps du texte BASIC puis acquiert une valeur numérique entière sur un octet. En sortie, le résultat est dans le registre X du 6510.

JSR $E206 Cette importante routine sert à acquérir un caractère dans le corps du texte BASIC par appel à CHRGOT. L'accumulateur A et les flags sont affectés de façon correcte pour les routines de l'interpréteur BASIC. Elle est nécessaire avant l'appel d'un grand nombre de routines du BASIC. Si le caractère lu dans le texte est un $00 de fin de ligne BASIC, la routine exécute deux instructions "PLA" pour enlever l'adresse de retour de la pile et retourner directement à l'interpréteur BASIC.

JSR $E20E Cette routine prends le caractère suivant dans le texte BASIC après avoir testé la présence d'une virgule. Si la virgule n'est pas présente, le message 'SYNTAX ERROR' sera affichée et le retour se fera à l'interpréteur BASIC.

JSR $E264 Cette routine exécute la fonction COS en ajoutant 0,5*PI à la valeur présente dans l'accumulateur flottant principal puis en effectuant la fonction SIN.

JSR $E26B Cette routine calcule la fonction SIN de la valeur contenue dans l'accumulateur flottant principal. Elle utilise la méthode du développement en série polynomiale décrite en $E059.

JSR $E2B4 Cette routine calcule la fonction TAN avec la valeur contenue dans l' accumulateur flottant principal en appellant successivement les routines SIN, puis COS, et enfin division. Elle utilise les deux accumulateurs flottants.

$E2EF Table des coéfficients pour le développement en série du SIN. C'est une fonctions d'ordre 5.

JSR $E30E Cette routine ($E30E à $E33D) exécute ATN. Cette fonction étant très peu utilisée, il est possible de la remplacer par une autre si le BASIC et le KERNAL ont été recopiés en RAM. ATN étant peu utilisé, voici un endroit tout trouvé pour dissimuler une routine de 47 octets. On pourra en profiter pour modifier également le nom de la

fonction ATN.Ce nom est présent sous la forme des valeurs $41, $54 et $CE (le bit 7 du dernier caractère du nom est à un) en $A174 à $A176 dans la table des mots-clé du BASIC.
Le polynôme d'évaluation de ATN est le plus complexe à calculer (c'est un polynôme d'ordre 11). C'est pourquoi on peut employer ATN(x) comme sous-programme de délai court. En effet, ATN est le plus lent des mots-clé du BASIC.

$E33E — Table des coéfficients pour le développement en série de ATN (ordre du polynôme = 11). Cette table occupe les adresses $E33E à $E376.

JMP $E37B — Cette routine simule l'enfoncement simultané des touches STOP et RESTORE et revient toujours à BASIC READY.

JMP $E38B — Routine de retour à BASIC READY avec impression d'un message d'erreur du BASIC. Le numéro du message doit être dans le registre X lors de l'appel de la routine.Si le contenu de X a le bit 7 à 0 ,on imprimera le message d'erreur numéro x. Si le bit 7 vaut 0, seul le message 'READY' sera imprimé.

JMP $E394 — Routine de RESET ou ré-initialisation du CBM 64. A le même effet que l'enfoncement du bouton RESET. Elle peut être appellée en BASIC par SYS 58260.
Il y tout de même une différence avec le vrai RESET: La couleur du curseur n'est pas remise à sa valeur par défaut (bleu clair).Il faut donc rétablir celle-ci en envoyant à l'écran le code de contrôle approprié.

JMP $E3BF — Initialisation de l'interpréteur BASIC. On peut utiliser cette routine en sortie d'un programme qui a modifié ou détruit des valeurs en RAM utiles au BASIC, telles que pointeurs de début et de fin de programme,vecteurs d'USR,etc... Recopie la routine CHRGOT en page zéro en $0073. Cette routine remplace NEW tout en étant plus sûre car l'interpréteur est vraiment ici remis à neuf.

JSR $E422 — Cette routine écrit le message de début "COMMODORE 64 BASIC", puis effectue NEW et retourne à BASIC READY. Si le BASIC a été recopié en RAM, on peut remplacer le JMP $A644 qui est en $E444 par un RTS ($60) pour revenir au programme appellant en langage machine.

$E447 — Table des vecteurs du BASIC qui est recopiée en RAM lors de l'initialisation en $0300 à $030B (messages d'erreur, démarrage à chaud, compactage des mots-clé, impression des mots-clé, exécution

d'une commande, acquisition d'un paramètre.

JSR $E457 Cette routine ré-initialise la table des vecteurs du BASIC à partir de la table ci-dessus.

$E460 Message 'BASIC BYTES FREE' suivi d'un $00.

$E479 Message '**** COMMODORE 64 BASIC V2 **** 64 K RAM SYSTEM' suivi de $00

$E4B7 à $E4D9 Zone de mémoire morte inutilisée qui peut être employée si le BASIC est recopié en RAM. 35 octets sont libres.

ROM $E000 - SYSTEME D'EXPLOITATION KERNAL

JSR $E4DA Cette routine efface un octet de la RAM couleur. $F3 et $F4 doivent contenir en entrée le pointeur vers la ligne-écran de la RAM couleur. Le registre Y doit contenir la position du caractère dans la ligne. l'octet en question est alors remis à la couleur de fond 0.

$E4EC Table des valeurs sur 2 octets nécessaires pour programmer la minuterie RS-232. 10 valeurs existent pour les vitesses de 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800 et 2400 bauds. Ces valeurs sont celles convenant aux CBM 64 avec contrôleur VIC-II au standard PAL, le modèle européen.

JSR $E500 Cette routine charge dans les registres X et Y (partie basse, partie haute) l'adresse du CIA1, soit $DC00.

JSR $E505 Cette routine met dans le registre X le nombre de colonnes et dans le registre Y le nombre de lignes de l'écran normal (soit 40 et 25)

JSR $E50A Cette routine lit ou positionne le curseur en X,Y dans l'écran: Si le flag C est à 1 en entrée, on lira la position du curseur (verticale dans X de 0 à 24, horizontale dans Y de 0 à 39). S'il vaut 0, on impose la position du curseur d'après les valeurs rangées dans le même ordre dands X et Y.

JSR $E518 Cette routine initialise clavier et écran et restaure les pointeurs de périphériques par défaut. Le curseur est repositionné en haut,

l'écran est bleu et le curseur clignotant. La table des adreses de lignes-écran est également reconstruite en $D9 et suivantes.

JSR $E56C — Cette routine établit le pointeur vers la ligne-écran où est le curseur en $D1 et $D2 et range ensuite en $D5 la longueur de la ligne-écran: 40 caractères ou 80 si c'est une ligne-suite. Ceci pour les lignes BASIC qui occupent deux lignes d'écran.

JSR $E59A — Cette routine effectue le redémarrage de l'écran et positionne le curseur en haut et à gauche, mais sans rendre le curseur visible ni toucher au clavier comme en $E518.

JSR $E5A0 — Cette routine effectue le redémarrage de l'écran mais en ré-écrivant dans les 47 registres du VIC-II les valeurs de départ lues dans la table en $ECB9.

JSR $E5B4 — Lecture d'un caractère dans le tampon de clavier. Attention, cette routine autorise les interruptions IRQ. Ne pas l'utiliser dans un programme où les interruptions sont inhibées.

JSR $E5CA — EDITEUR D'ECRAN: Cette routine accepte des caractères au clavier et les affiche à l'écran. La prise en compte de tous les codes de contrôle est correcte. En entrée, elle commence par initialiser les paramètres nécessaires pour la prise en compte de la ligne dans le tampon d'entrée BASIC en $0200.

JSR $E632 — Routine d'acquisition d'un caractère sur le périphérique d'entrée courant.Le caractére lu est dans le registre A en sortie.

JSR $E716 — Routine d'écriture d'un caractère à l'écran. En entrée, le code ASCII du caractére doit être dans l'accumulateur A.Tout les codes de contrôle sont pris en compte.

JSR $E8CB — Cette routine teste si le contenu de l'accumulateur A est un code de changement de couleur et ,si oui, range cette valeur en $0286 qui est l'adresse de la couleur de tracé en cours.

JSR $E8EA — Cette routine décale l'écran d'une ligne vers le haut (SCROLL). Le décalage de l'écran est ralenti si on maintient la touche CTRL enfoncée.l'appel en BASIC par SYS 59626 est autorisé.

JSR $E965 — Cette routine insère une ligne vierge dans l'écran sous la ligne où est le curseur. Elle ne fonction-

ne correctement qu'une fois comme lors de l'insertion d'une deuxième ligne de 40 caractères dans une ligne BASIC. Si on l'appelle de nombreuses fois à la même ligne, la gestion du curseur devient incorrecte.

JSR $E9D2 — Cette routine déplace une ligne-écran vers une autre (lignes de 40 caractères). Elle déplace la RAM écran et la RAM couleur. En entrée, les adresses $AC et $AD contiennent (partie basse, partie haute) l'adresse de la ligne-écran source;
$D1 et $D2 l'adresse de la ligne destination.
$AE et $AF l'adresse de la ligne-couleur source.
$F3 et $F4 l'adresse de la ligne-couleur destination.

JSR $E9F0 — Cette routine calcule l'adresse d'une ligne-écran. En entrée, le registre X contient le numéro de la ligne-écran et le pointeur en $0288 contient le numéro de la page-écran (adresse/256). En sortie, le résultat est en $D1 et $D2 (partie basse,partie haute) et ainsi prêt pour la routine ci-dessus.

JSR $E9FF — Cette routine efface la ligne-écran dont le numéro est dans le registre X.L'appel en BASIC peut s'effectuer par:

```
POKE 781,ligne : SYS 59903
```

Ceci est très pratique dans les programmes qui réalisent des saisies de donées à l'écran.

JSR $EA1C — Cette routine écrit un caractére à l'endroit du curseur sur l'écran. En entrée, le code du caractère doit être dans l'accumulateur A et la couleur de tracé dans le registre X. Elle ne déplace pas le curseur mais transfère directement la valeur de l'accumulateur dans l'écran comme le POKE du BASIC et non comme le PRINT.
L'accès en BASIC se fait par:

```
POKE 780,car : POKE 781,couleur : SYS 59932
```

JSR $EA24 — Cette routine établit l'adresse de la ligne de RAM couleur correspondant à la ligne-écran courante. En entrée, l'adresse de la ligne-écran doit être en $D1 et $D2 (partie basse, partie haute). En sortie, l'adresse de la ligne-couleur est en $F3 et $F4. Il convient d'utiliser cette routine avant celle en $E9D2.

JMP $EA31 — ROUTINE D'INTERRUPTION IRQ NORMALE

Elle s'occupe du balayage de clavier, de la gestion de l'horloge TI, du clignotement de

curseur et de la mise en marche ou de l'arrêt du moteur de cassette en fonction de l'enfoncement des interrupteurs du lecteur de cassettes.

JSR $EA87 — Routine de balayage du clavier. Si une touche est valablement enfoncée,le code ASCII de la touche est rangé dans le tampon de clavier. Le nombre de caractères présents dans le tampon est mis à jour en $C6. La taille maximale du tampon est rangée en $0289 et si elle est dépassée, le tampon est vidé et son remplissage recommence. Si le KERNAL a été recopié en RAM, le pointeur vers la table de décodage peut être modifié (voir la table ci-dessous en $EB79). Il est également possible de modifier soit les valeurs dans cette table, soit le contenu des tables directement. On trouvera en annexe dans le programme AZERTY un exemple de modification du décodage clavier qui adapte celui-ci aux accents du français.

$EB79 — Table des adresses des quatre tables de décodage correspondant aux 4 couches du clavier: normal, avec SHIFT, avec LOGO et avec CTRL enfoncés.

En $EB79 (valeur = $EB81): Clavier normal
En $EB7B (valeur = $EBC2): Clavier avec SHIFT
En $EB7D (valeur = $EC03): Clavier avec LOGO (C=)
En $EB7F (valeur = $EC78): Clavier avec CTRL

$EB81 — Table de décodage du clavier normal : 64 octets + 1 octet de valeur $FF signifiant fin de table. Chaque octet représente le code ASCII de la touche correspondante de la matrice d'interrupteurs dont est formé le clavier. $FF dans la table signifie que la touche correspondante ne doit générer aucun code ASCII.

$EBC2 — Idem pour le clavier avec SHIFT.

$EC03 — Idem pour le clavier avec LOGO (C=).

$EC78 — Idem pour le clavier avec CTRL.

JMP $E6A8 — Portion de la routine IRQ qui gère les codes ASCII spéciaux tels que 8, 9, 14 et 142.

$ECB9 — Table des 47 valeurs par défaut utilisées lors de la programmation initiale du VIC-II. La table est en fait trop courte d'un octet et c'est le 'L' du mot 'LOAD' - voir ci-dessous - qui sert de pointeur de couleur pour le lutin numéro 7, ce qui le programme en gris clair!! Erreur bénigne, mais erreur tout de même de la part de COMMODORE.

$ECE7 Message 'LOAD'+<RETURN>+'RUN'+<RETURN>. Ce message se mettra d'office dans le tampon de clavier à l'enfoncement de la touche STOP-RUN simultanément avec SHIFT. Une bonne façon de modifier ceci est d'écrire un autre message plus court que le tampon de clavier n'importe où en mémoire RAM et - le KERNAL aura été recopié en RAM- de modifier le pointeur vers ce message. On utilisera, par exemple, le message : 'LOAD"*",8'+<RETURN> que l'on rangera en $C000. Il faut modifier la longueur de la chaîne de caractères et son pointeur comme suit:

En $E5EF : longueur de la chaîne
En $E5F4 : partie basse du pointeur de chaîne -1
En $E5F5 : partie haute du pointeur de chaîne -1

(valeurs en rom de ces 3 octets: $09, $E6, $EC)

$ECF0 Table des poids faibles (parties basses) des adresses de début de chacune des 25 lignes de l'écran. Les parties hautes de ces adresses sont rangées en mémoire RAM en $D9 à $F2 car elles sont susceptibles d'être modifiées lors du déplacement en mémoire de l'écran (voir en annexe le programme ECRAN)

JSR $ED09 Cette routine envoie sur le bus série IEEE la commande TALK (parlez)

JSR $ED0C Cette routine envoie sur le bus série IEEE la commande LISTEN (écoutez)

JSR $ED40 Cette routine envoie un octet sur le bus série IEEE. L'octet doit être présent en entrée en $95. Ne pas utiliser cette routine, mais bien celle en $EDDD.

JSR $EDB9 Cette routine envoie une adresse secondaire sur le bus série IEEE puis remet le fil ATN au niveau bas pour signaler la fin d'une commande. En entrée, l'adresse secondaire doit être présente en $95 (après un LISTEN).

JSR $EDC7 Idem , mais après une commande TALK. La routine attend ensuite la réponse, c'est-à-dire une impulsion positive sur le fil d'horloge (CLOCK) du bus IEEE.

JSR $EDDD Cette routine envoie un octet de donnée sur le bus IEEE série. Elle maintient en mémoire-tampon un caractère d'avance jusqu'à ce qu'il n'y ait plus de données à expédier. Le dernier caractère est alors envoyé avec le fil ATN au niveau haut pour signaler la fin de la transmission. Elle utilise la routine en $ED40.

JSR $EDEF — Cette routine envoie sur le bus IEEE série la commande UNTALK (cessez de parler).

JSR $EDFE — Cette routine envoie sur le bus IEEE série la commande UNLISTEN (cessez d'écouter).

JSR $EE13 — Cette routine reçoit un caractère de donnée en provenance du bus IEEE série.

JSR $EE85 — Cette routine met le fil d'horloge IEEE série au niveau bas.

JSR $EE8E — Cette routine met le fil d'horloge IEEE série au niveau haut.

JSR $EE97 — Cette routine met le fil de donnée IEEE série au niveau bas.

JSR $EEA0 — Cette routine met le fil de donnée IEEE série au niveau haut.

JSR $EEA9 — Cette routine effectue la lecture du port parallèle PA2, attend une lecture stable et en sortie, revient avec le bit de donnée lu sur l'IEEE série dans le flag C et le bit d'horloge dans le flag N.

JSR $EEB3 — Cette routine génère un petit délai de 1 milliseconde. Elle utilise le registre X du 6510.

$EEBB — Routines de gestion de l'RS-232.

$F0BD — MESSAGES DU KERNAL

Chaque message du KERNAL se termine par un caractère dont le bit 7 vaut 1.

N.	MESSAGE D'ERREUR DU KERNAL
0	I/O ERROR
1	SEARCHING
2	FOR
3	PRESS PLAY ON TAPE
4	PRESS RECORD AND PLAY ON TAPE
5	LOADING
6	SAVING
7	VERIFYING
8	FOUND
9	OK

JSR $E12B — Cette routine écrit un message du KERNAL si l'on est en mode direct. En entrée, le numéro de

message doit être dans le registre Y du 6510.

JSR $F12F — Cette routine écrit un message du KERNAL en mode direct ou en mode programme. En entrée, le numéro de message doit être dans le registre Y du 6510.

JSR $F13E — Cette routine prends un caractère dans le périphérique courant en entrée (prise au vol comme avec un GET en BASIC).

JSR $F157 — Cette routine prends un caractère en entrée avec la gestion du RETURN effectuée comme dans un INPUT en BASIC.

JSR $F199 — Prise d'un caractère dans le tampon de cassette.

JSR $F1B5 — Prise d'un caractère sur le bus IEEE série.

JSR $F1B8 — Prise d'un caractère sur l'RS-232.

JSR $F1CA — Envoi d'un caractère sur un périphérique de sortie.

JSR $F20E — Cette routine établit le numéro du périphérique d'entrée courant. En entrée, le registre X contient le numéro du fichier logique. Si ce fichier n'a pas été correctement ouvert, la routine envoie le message 'FILE NOT OPEN ERROR'.

JSR $F250 — Cette routine établit le numéro du périphérique de sortie courant. Comme ci-dessus, le registre X contient le numéro du fichier logique. De même, la routine renvoie le message d'erreur si le fichier n'a pas été correctement ouvert.

JSR $F291 — Fermeture d'un fichier logique. En entrée, l'accumulateur A contient le numéro d'un fichier logique. Si celui-ci n'a pas été préalablement ouvert, il n'y a pas d'action effectuée mais la routine ne produit aucun message d'erreur dans ce cas.

JSR $F30F — Cette routine vérifie que la valeur contenue dans le registre X existe bien dans la table des fichiers logiques ouverts aux adresses $0259 et suivantes.

JSR $F31F — Acquisition des paramètres d'un fichier logique. En entrée, le registre X indique la position du fichier considéré dans la table des fichiers ouverts en $0259. Elle dépose les paramètres lus dans cette table dans les descripteurs de périphériques courants:

En $B8 : le numéro de fichier logique
En $BA : le numéro de périphérique physique
En $B9 : l'adresse secondaire de ce périphérique.

JSR $F32F — Fermeture de tous les fichiers logiques. Cette fonction n'existe pas en BASIC où l'on doit toujours préciser le numéro du fichier à refermer par CLOSE xx .On peut y accéder par : SYS 62255 En anglais, cette routine s'appelle CLALL ou CLOSE ALL.

JSR $F333 — Cette routine restaure les périphériques d'entrée et de sortie par défaut: le clavier et l'écran.

JSR $F34A — Routine OPEN : ouverture d'un fichier logique.

JSR $F483 — Initialisation du CIA-2.

JSR $F49E — Cette routine effectue la commande LOAD à partir du périphérique d'entrée courant. L'adresse de début de chargement doit être en entrée en $C5 et $C6 (partie basse, partie haute).

JSR $F5DD — Routine SAVE sur le périphérique courant.

JSR $F657 — Cette routine effectue la commande SAVE sur cassette. En entrée, les registres X et Y pointent vers le tampon cassette (partie basse dans X, partie haute dans Y).

JSR $F693 — Cette routine envoie à l'écran le message "SAVING"+nom de programme.

JSR $F69B — Cette routine incrémente l'horloge TI d'un soixantième de seconde. On peut utiliser cette routine pour tenir à jour l'horloge pendant les instants où les interruptions normales sont interdites ou bien à l'intérieur d'une routine d'interruption qui remplace totalement celle d'origine.

JSR $F6DD — Cette routine lit l'heure dans le compteur TI: la partie haute est en sortie dans le registre Y, la partie moyenne dans le registre X et la partie basse dans le registre A.

JSR $F6E4 — Cette routine écrit l'heure en mémoire dans le compteur TI en $A0, $A1 et $A2. En entrée, les trois registres contiennent la valeur à écrire sur trois octets et exprimée en soixantièmes de secondes.L'ordre des octets est identique à celui de la routine ci-dessus.

JSR $F6ED — Cette routine teste la touche STOP et se termine par un RTS normal si elle n'est pas enfoncée. Si STOP a été enfoncé, elle referme tous les fichiers ouverts et remet à 0 le compteur de caractères du tampon de clavier.

JSR $F72C — Cette routine lit un bloc 'HEADER' ou en-tête sur la cassette. Dès qu'elle a trouvé le bloc recherché, elle affiche à l'écran le message "FOUND"+nom de programme.

JSR $F82E — Cette routine teste les boutons du lecteur de cassette. En sortie, le flag Z du 6510 vaut 0 si aucun bouton du lecteur n'est enfoncé.

JSR $F841 — Cette routine lit un bloc de données sur la cassette. Ce peut être un bloc d'en-tête, un bloc de données ou un programme BASIC, par exemple.

JSR $F864 — Cette routine écrit un bloc sur cassette.

JSR $FB8E — Cette routine prépare les paramètres pour LOAD ou SAVE.

JSR $FB97 — Suite de la routine ci-dessus qui est , en fait, en deux partie à appeller successivement.

JSR $FCCA — Cette routine arrête le moteur de la cassette.

JSR $FCD1 — Cette routine compare deux pointeurs en page 0 : Celui en $AC et $AD avec celui en $AE et $AF. Les flags sont affectés comme lors d'un CMP normal.

JSR $FCDB — Cette routine incrémente le pointeur sur deux octets en $AC et $AD (partie basse, partie haute).

JMP $FCE2 — routine de RESET. Elle ne revient jamais au programme appellant. Elle test et tient compte d'une éventuelle cartouche de ROM en $A000.

JSR $FD15 — Cette routine restaure la table des vecteurs d'entrée/sortie du KERNAL à ses valeurs par défaut.
ATTENTION!!! Cette routine réécrit la table qui est, par défaut, en $FD30 à $FD4E. C'est la seule routine du CBM 64 qui écrit dans la RAM derrière le KERNAL. Ne jamais utiliser dès lors les adresses $FD30 à $FD4E pour des programmes vitaux. Ils seraient 'écrasés' aux adresses 64816 à 64846 lors de la frappe simultanée de STOP et RESTORE ou à l'exécution d'une instruction BRK.

$FD30 — Table des 16 vecteurs recopiée en RAM lors du RESET en $0314 à $0333.

JSR $FD50 Cette routine initialise les pointeurs de mémoire et les pages 0, 2 et 3 (variables de service du KERNAL).

JSR $FDA3 Initialisation des périphériques.

JSR $FDF9 Cette routine initialise la longueur du nom et le nom d'un fichier. En entrée, le registre A doit contenir la longueur. Les registres X et Y pointent (partie basse, partie haute) vers la chaîne de caractères du nom du fichier.

JSR $FE00 Cette routine initialise le numéro de fichier logique (présent dans l'accumulateur A en entrée), l'adresse du périphérique (dans le registre X en entrée) et l'adresse secondaire (dans le registre Y en entrée).

JSR $FE25 Routine à double usage: Elle lit ou établit le pointeur de sommet de mémoire BASIC. Si le flag C (Carry) vaut 1 en entrée, il y a lecture. Sinon, il y a écriture. La valeur lue ou à écrire est présentée dans les registres X et Y (partie basse, partie haute).

JSR $FE34 Routine à double usage: Elle lit ou établit le pointeur de bas de mémoire BASIC. Usage semblable à la routine ci-dessus.

JSR $FE66 ROUTINE de BREAK. Elle restaure tous les périphériques et revient à BASIC READY. L'appel par SYS en BASIC est toujours possible: Il suffit d'exécuter un SYS suivi de l'adresse d'un octet de valeur zéro. l'instruction BRK de l'assembleur ($00 en langage machine) est en fait une simulation programmée de routine d'interruption. Voici quelques adresses habituellement à zéro:

```
SYS 2
SYS 8
SYS 2048
```

Si on a préalablement activé le programme MONITEUR, les SYS ci-dessus font 'tomber' le KERNAL dans le MONITEUR, qui prend alors le contrôle du CBM 64.

$FEC2 Tables des vitesses de transmission pour la transmission RS-232 pour les CBM 64 américains au standard télévision NTSC.

$FF81 à $FFF5 Table des 39 JMP officiels du KERNAL. Ces 39 points d'entrée sont reconnus et commentés par COMMODORE. Donc, les programmeurs qui travaillent en assembleur et désirent que leurs programmes

restent le plus possible indépendant d'un type de machine particulier ont tout intérêt à n'utiliser que ces 39 routines du KERNAL. Il semble que l'on soit ainsi à l'abri de certains changement de version des programmes internes. En effet, depuis la création de la société COMMODORE, toutes les machines conservent cette table. Elle s'allonge malgré tout régulièrement mais la compatibilité des futures machines de la gamme avec le CBM 64 semble assurée.

Les noms des routines reprises ci-dessous sont les noms officiels que COMMODORE donne à ces routines.

$FF81 -CINT Routine d'initialisation de l'éditeur d'écran et du VIC-II. Elle modifie les registres A, X et Y et ne nécessite pas de paramètres en entrée. A utiliser par exemple en fin d'un programme qui modifie fortement la configuration de l'écran ou la page 0.

Appel par : SYS CINT

$FF84 -IOINIT Routine d'initialisation des périphériques. Elle modifie les registres A, X et Y et ne nécessite pas de paramètres en entrée. A utiliser en sortie d'un programme qui modifie fortement la page 0 ou les registres des 6526 (les 2 CIA).

Appel par : JSR IOINIT

$FF87 -RAMTAS Cette routine initialise la mémoire RAM, alloue le tampon de clavier et les adresses d'écran.

Appel par : JSR RAMTAS

Elle modifie les registres A, X et Y et remet à zéro les octets $0000 à$0101 et $0200 à $03FF. Elle initialise les pointeurs en page 0 et 3 pour l'écran et la cassette. Elle test ensuite la présence de mémoire RAM sans en effacer le contenu.

$FF8A -RESTOR Routine de restauration des vecteurs du KERNAL. Les 16 vecteurs des routines principales du KERNAL sont des pointeurs indirects vers les routines correspondantes. Ces vecteurs sont en mémoire RAM en $0314 et suivants. Après un programme qui a modifié certains pointeurs, on peut les rétablir à leurs valeurs initiales par un appel à cette routine.

Appel par : JSR RESTOR

$FF8D -VECTOR Cette routine établit ou lit les vecteurs du KERNAL. Si le flag C (CARRY) vaut 1 en entrée, les vecteurs (les 32 octets situés en $0314 et suivants) sont lus en $0314 et recopiés en mémoire à l'adresse qui était contenue dans les registres X et Y en entrée (partie basse dans X, partie haute dans Y). Si, en entrée, le flag C (CARRY) vaut 0, la table de 32 octets dont l'adresse est fournie dans les registres X et Y (partie basse, partie haute) est recopiée en $0314. A utiliser pour modifier un ou plusieurs vecteurs simultanément: Il faut tout d'abord lire la table, la copier dans une zone provisoire, la modifier puis la réécrire à sa place à l'aide de cette routine. Ceci pour éviter des erreurs dues aux routines d'interruptions qui se servent de ces vecteurs.

Mettre les vecteurs en $2000:

```
LDX £$00    ;PARTIE BASSE
LDY £$20    ;PARTIE HAUTE
SEC
JSR VECTOR
```

Etablir les vecteurs à partir de la table en $3000

```
LDX £$00
LDY £$30
CLC
JSR VECTOR
```

$FF90 -SETMSG Cette routine contrôle l'impression des messages du KERNAL. Elle établit le type de messages qui doivent être envoyé à l'écran en fonction des circonstances (mode direct ou mode RUN).
Elle utilise l'accumulateur A en entrée. Le bit 6 à 1 signifie que l'on doit utiliser les messages de la table en $F0BD. Si le bit 7 vaut 1, on doit utiliser les messages de la table en $A328.
Si les bits 7 et 6 sont tous deux à 0, on n'imprime jamais aucun message. Cette routine modifie le registre A.

$FF93 -SECOND Cette routine envoie une adresse secondaire après un LISTEN sur le bus IEEE série. A ne pas employer aprés TALK.

Exemple: Envoi de l'adresse secondaire 15 au lecteur de disquette 1541 pour envoyer une commande sur le canal d'erreur (il faudra relire le canal d'erreur par READST en $FFB7).

Appel par:

```
SECADD = $0F      ;15=ADRESSESECONDAIRE
ADDR   = $08      ; 8=N.DE.PERIPHERIQUE
LDA £ADDR
JSR LISTEN        ;ENVOIE LISTEN
LDA £SECADD
JSR SECOND        ;ENVOIE ADD.SECONDAIRE
```

$FF96 -TKSA Cette routine envoie une adresse secondaire aprés TALK. A ne pas employer après LISTEN. Pour lire le canal d'erreur du lecteur de disquettes, on prépare la transaction par l'exemple ci-dessous:

Appel par:

```
SECADD = $0F      ;15=ADD.SECONDAIRE
ADDR   = $08      ; 8=N.DE PERIPHERIQUE
LDA £ADDR
JSR TALK          ;ENVOIE L'ADRESSE
LDA £SECADD
JSR TKSA          ;ENVOIE ADD.SECONDAIRE
```

$FF99 -MEMTOP Cette routine est à double usage: Elle lit ou établit le sommet de mémoire. Si le flag CARRY (C) vaut 1 en entrée, le pointeur de sommet de mémoire en $0283 et $0284 est chargé dans les registres X et Y. Si le flag CARRY vaut 0 en entrée, le pointeur stocké dans les registres X et Y (partie basse, partie haute) est transféré en $0283 et $0284. Pour redescendre le sommet de mémoire de 512 octets (2 pages de 256 octets), il suffit de faire:

```
SEC               ;flag CARRY à 1 pour
JSR MEMTOP        ;lire le pointeur
DEY               ;décrémente la partie
DEY               ;haute 2 fois.
CLC               ;flag CARRY = 0
JSR MEMTOP        ;réécrit le pointeur
                  ;diminué.
```

Note: Le registre A n'est pas modifié.

$FF9C -MEMBOT Routine à double usage: Elle lit ou établit le pointeur de bas de zone BASIC. Ce début de zone est normalement en $0800 (2048). Le mode d'emploi est identique à celui de la routine MEMTOP. Le pointeur qui est modifié est en $0281 et $0282.

Ces pointeurs en $0281 à $0284 sont à l'usage du KERNAL. Ne pas les confondre avec les pointeurs de mémoire en page 0 qui sont utilisés uniquement par l'interpréteur BASIC.

$FF9F -SCNKEY Routine de balayage du clavier. Elle ne peut être utilisée qu'aprés un appel à IOINIT. Elle modifie les registres A, X et Y. On peut l'utiliser indépendamment des routines d'interuptions. Par exemple, dans un programme où les interruptions sont interdites, ce qui permet d'avoir accès au clavier malgré tout. En sortie, le caractère éventuellement frappé au clavier est rangé dans le tampon de clavier. Ne pas utiliser cette routine si les interruptions normales sont en service. Pour lire un caractére au clavier, il faut appeler GETIN aprés avoir balayé le clavier par SCNKEY, ou GETIN seul si les interruptions normales sont en service.

Appel par : JSR SCNKEY

Exemple : lecture d'une touche au vol :

```
JSR SCNKEY ;Balayage clavier
JSR GETIN  ;lit le caractère
JSR CHROUT ;l'écrit a l'écran
...
```

Exemple: attente de la frappe d'un RETURN au clavier:

```
ATTEND  JSR SCNKEY
        JSR GETIN
        CMP £$OD  ;est-ce un RETURN ?
        BEQ ATTEND;si non,boucler
```

$FFA2 -SETTMO Cette routine établit le délai d'attente pour bus IEEE parallèle. Sur les CBM 64 non équipés d'une carte d'extension IEEE-488 parallèle, cette routine n'est d'aucune utilité. Le délai d'attente maxi est de 64 millisecondes sur les périphériques parallèles. Le délai est infini si en entrée l'accumulateur A contient $FF.

$FFA5 -ACPTR Acquisition d'un octet sur le bus IEEE série. Pour lire un octet d'un fichier su disque,par exemple. En sortie, l'accumulateur A contient l'octet lu. Bien entendu, pour le lecteur de disques, il faut avoir envoyé au préalable la commande TALK. Si nécessaire, cette routine doit être suivie de JSR

TKSA pour communiquer au lecteur le numéro de tampon à utiliser (l'adresse secondaire). Cette routine modifie les registres A et X.

Appel par : JSR ACPTR

$FFA8 -CIOUT Cette routine envoie un octet sur le bus IEEE série. Elle doit être précédée de LISTEN et éventuellement de SECOND. En entrée, l'accumulateur A doit contenir le caractère à envoyer. Pour lire une éventuelle erreur, il faut ensuite utiliser la routine READST. Cette routine maintient en permanence un des octets à envoyer dans son tampon interne: Elle n'envoie le premier caractère qu'elle reçoit qu'au deuxième appel . Le dernier caractère du message doit être envoyé par UNLSN qui doit suivre tout envoi de données sur le bus IEEE série.

$FFAB -UNTLK Envoie UNTALK sur le bus IEEE série. Ceci sert à signaler à un périphérique qu'il doit cesser d'envoyer des données. Cette routine modifie le contenu de l'acccumulateur A.

Appel par : JSR UNTLK

$FFAE -UNLSTN Cette routine envoie UNLISTEN sur le bus IEEE série . Ceci signale à tous les périphériques qu'ils doivent arrêter de recevoir des données. A utiliser par exemple après avoir envoyé un message au disque ou à l'imprimante. Elle ne modifie que l'accumulateur A. Une éventuelle erreur peut être détectée avc un JSR READST.

Appel par : JSR UNLSTN

$FFB1 -LISTEN Cette routine envoie à un périphérique IEEE série la commande LISTEN lui signalant qu'il va reçevoir des données. En entrée, l'accumulateur A contient le numéro (donc l'adresse IEEE) du périphérique concerné: Valeur 4 à 31. Elle ne modifie que l'accumulateur A. Par exemple, avant d'envoyer une commande au lecteur de disquette 1541, il faut faire :

```
LDA £$08    ;08 = disque
JSR LISTEN ;commande LISTEN
```

$FFB4 -TALK Cette routine envoie à un périphérique IEEE série la commande TALK lui signalant qu'il peut envoyer des données. En entrée, l'accumulateur A contient l'adresse du périphérique. Elle ne modifie que l'accumulateur A.

```
Appel par :    LDA £$08    ;08 = disque
               JSR TALK    ;commande TALK
```

$FFB7 -READST Cette routine lit le registre ST-STATUS qui indique les erreurs des périphériques IEEE série et CASSETTE. Elle modifie l'accumulateur A qui contiendra en sortie la valeur de l'octet ST. Il faut utiliser cette routine après tout accès à un périphérique pour vérifier la validité des transactions effectuées. Par exemple, après un SAVE sur disquette pour savoir si le disque n'était pas protégé en écriture, etc...
Après l'appel, l'octet ST est dans l'accumulateur A et en $90 en page 0. Cet octet s'interprète bit par bit.

```
Appel par :    JSR READST      ;en assembleur

   ou par :    ? PEEK(144)     :REM en BASIC

   ou      :    ? ST
```

SIGNIFICATION DES BITS DE L'OCTET ST-STATUS

BIT	VALEUR	SIGNIFICATION
0	1	Délai dépassé écriture IEEE par.
1	2	Délai dépassé lecture IEEE par.
2	4	Bloc trop court sur cassette
3	8	Bloc trop long sur cassette
4	16	Erreur fatale en lecture
5	32	Erreur vérification cassette
6	64	Fin de fichier rencontrée
7	128	Fin de bande sur cassette ou périphérique IEEE absent.

ATTENTION: Il peut y avoir plusieurs erreurs simultanément. Par exemple, si PRINT ST donne un résultat de 48 après un LOAD sur cassette, c'est que la cassette est à ce point mauvaise que les données lues sont

irrécupérables.(noter que 48=32+16 soit les 2 erreurs 16 et 32 à la fois)

$FFBA -SETLFS Cette routine établit les adresses primaires et secondaires et le numéro logique d'un fichier. Pour le mode d'emploi de cette routine, voir la routine SAVE. Le nom de SETLFS vient de SET (mettre), L (logical filename), F (first address), S (secondary address). Les numéros de fichiers logiques sont compris, en ASSEMBLEUR comme en BASIC, entre 0 et 255. On ne peut avoir que 10 fichiers ouverts à la fois. Les numéros de périphériques les plus courants se retrouvent au tableau ci-desous:

N.Périph.	Désignation
0	CLAVIER (lecture seulement)
1	CASSETTE(Interdit sur le SX-64)
2	RS-232 (lecture et écriture)
3	ECRAN (lecture et écriture)
4	IMPRIMANTE (écriture seulement)
5	2ème IMPRIMANTE (idem)
8	LECTEUR DE DISQUES (lect. et écrit.)
9	2ème LECTEUR DE DISQUES
11	MODEM TELEPHONIQUE (lect. et écrit.)
.	
31	N. de périphérique maximum

Le numéro de périphérique IEEE est toujours compris entre 4 et 31, car les numéros 0 à 3 sont réservés aux accés internes au CBM 64.
Pour LOAD et SAVE, l'adresse secondaire 1 indique que le mouvement doit se faire avec les adresses réelles (format ABSOLU). Dans les autres cas, le chargement se produira toujours en début de zone BASIC à l'adresse $0800 ou 2048 (format RELOCATABLE).

Pour l'imprimante, l'adresse secondaire permet l'accès aux fonctions spéciales.
En entrée, l'accumulateur A contient le numéro de fichier logique, le registre X le numéro de périphérique et le registre Y l'adresse secondaire (ou $FF si pas d'adresse secondaire).

Appel par : JSR SETLFS

$FFBD -SETNAM Cette routine établit le nom d'un fichier avant un OPEN, SAVE ou LOAD. En entrée, l'accumulateur A contient le nombre de caractères du nom, le registre X la partie basse de l'adresse du nom et le registre Y la partie haute.
Le lecteur de disquettes refuse les noms de plus de 16 caractères. Le lecteur de cassette accepte les noms de programmes jusqu'à 255 caractères,ce qui peut être fort utile. Cela permet en effet de sauver de petits programmes écrits en langage machine dans un bloc d'en-tête sur cassette sans charger deprogrammeproprementdit.

Appel par : JSR SETNAM

$FFCO Cette routine ouvre un fichier logique et doit être précédée de SETNAM et SETLFS pour faire en langage machine la même chose qu'OPEN en BASIC L'erreur éventuelle peut être lue ensuite par JSR READST.

Par exemple : OPEN 1,8,2,"PROG,S,R" peut être remplacé par :

```
OUVRE  LDY £>NOMFIC     ;ADRESSE HAUTE DU NOM
       LDX £<NOMFIC     ;ADRESSE BASSE DU NOM
       LDA £$08         ;LONGUEUR DU NOM
       JSR $FFBD        ;SETNAM
       LDA £$01         ;NUMERO DE FICHIER
       LDY £$02         ;ADRESSE SEECONDAIRE
       LDX £$08         ;NUMERO DE PERIPHERIQUE
       JSR $FFBA        ;SETLFS
       JSR $FFCO        ;OPEN
       RTS

NOMFIC .BYTE 'PROG,S,R' ;NOM DU FICHIER
```

$FFC3 -CLOSE Cette routine referme un fichier logique et peut être suivie d'un appel à READST pour lire l'erreur éventuelle. Elle modifie les registres A,X et Y.

Par exemple, pour fermer le fichier logique 1 :

```
       LDA £$01         ;FICHIER N.1
       JSR $FFC3        ;CLOSE
```

$FFC6 -CHKIN Dérivation du canal d'entrée. Pour dialoguer avec

un périphérique extérieur, il faut d'abord ouvrir un fichier logique vers ce périphérique avec OPEN (voir ci-dessus). Après qu'un certain nombre de fichiers logiques d'entrée aient été ouverts, il faut aiguiller la routine d'entrée (CHRIN- voir plus loin), qui est unique, vers l'un ou l'autre des fichiers logiques ouverts à ce moment. C'est le rôle de CHKIN de manipuler cet aiguillage devant la routine d'entrée.
Quand le périphérique physique défini par un fichier logique est un périphérique IEEE, la routine CHKIN envoie sur le bus la commande TALK. Tous les appels consécutifs aux routines CHRIN ou GETIN prendront des caractères sur ce périphérique jusqu'à ce que l'aiguillage soit à nouveau redéplacé par CHKIN ou par CLRCHN (Voir plus loin). Cette routine modifie les registres A et X. Elle doit être précédée d'au moins un OPEN.

Appel par :

```
LDX £$01  ;N. DU FICHIER D'ENTREE
JSR $FFC6 ;AIGUILLE VERS LE PERIPHE-
          ;RIQUE DU FICHIER 1.
```

$FFC9 -CHKOUT Dérivation du canal de sortie. Cette routine redéfinit le numéro du fichier logique à utiliser pour toutes les sorties consécutives de caractères par CHROUT. Le canal de sortie restera affecté à ce périphérique jusqu'à l'appel de CHKOUT ou de CLRCHN. Toutes les remarques faites ci-dessus pour la routine CHKIN restent valables. Elle modifie A et X. Les erreurs peuvent être lues par READST.

Appel par:

```
LDX £$01  ;FICHIER 1
JSR $FFC9 ;A UTILISER EN SORTIE
```

$FFCl -CLRCHN Cette routine rétablit les canaux d'entrée/sortie vers le clavier et l'écran. Il faut appeler cette routine pour rétablir les aiguillages modifiés par CHKIN et CHKOUT vers les périphériques 0 et 3. Entre autres, en refermant un canal ouvert vers un périphérique IEEE, la routine CLRCHN envoie UNTALK ou UNLISTEN sur le bus. Ne pas appeler cette routine après une communication peut conduire à des erreurs sur le bus IEEE. Par exemple, laisser un canal de sortie ouvert à l'imprimante et un canal d'entrée ouvert sur le disque peut donner une copie sur papier de tous les octets envoyés par le disque . Cette routine est appelée d'office par CLALL.

Appel par : JSR $FFCC

$FFCF -CHRIN Cette routine lit un caractère sur le canal d'entrée, normalement le clavier sauf après appel de OPEN et CHKIN. Elle ne modifie pas les aiguillages des routines d'entrée.
Si le canal d'entrée est le clavier, il se passe diverses choses liées à l'éditeur d'écran. Le premier appel à CHRIN fait apparaître le curseur clignotant. Le curseur reste ensuite visible jusqu'à la frappe au clavier d'un caractère 13 (RETURN)

A ce moment, tous les caractères de la ligne où se trouve le curseur (40 ou 80 caractères si le curseur est passé une fois à la ligne) sont transférés dans le tampon d'entrée du BASIC en $200. Cette routine est à peu de choses près, l'INPUT du BASIC quand l'aiguillage est sur le canal du clavier.

Les appels successifs à CHRIN retournent au programme appellant les caractères successifs rangés à ce moment dans le tampon d'entrée jusque et y compris le caractère RETURN. Un appel de plus à CHRIN et le processu recommence! Elle ne modifie que A et X. On peut donc utiliser Y comme index dans une chaîne de caractères à entrer, par exemple.

Appel par : JSR $FFCF

$FFD2 -CHROUT Ecriture d'un caractère sur le canal de sortie. Normalement, c'est l'écran, mais ce peut être un autre périphérique si l'on a utilisé précédemment OPEN et CHKOUT. Attention à ne pas avoir laissé de canaux ouverts en sortie sur le bus IEEE car si plusieurs sont ouverts simultanément et que l'appel à CLRCHN n'a pas été effectué, tous les périphériques concernés reçevront les données émises par CHROUT. Elle ne modifie que l'accumulateur A.

Appel par : LDA £CARAC
JSR $FFD2

$FFD5 -LOAD Cette routine charge la mémoire à partir d'un périphérique. Elle ne peut utiliser les périphériques 0, 2 et 3: Clavier, RS232 et écran. Elle peut effectuer au choix un chargement ou une vérification (LOAD ou VERIFY).

En entrée, si l'accumulateur A contient 0, on effectuera 'LOAD'; s'il contient 1, on effectuera 'VERIFY'.

Si, en entrée, l'adresse secondaire établie par SETLFS est 0, les informations d'adresse de chargement contenues au début du fichier sur le disque ou la cassette seront ignorés et les registres X et Y doivent contenir l'adresse de début de chargement (partie basse dans X, partie haute dans Y). Le LOAD normal du BASIC emploie cette routine avec $0801 dans X et Y. Pour un LOAD absolu, c'est-à-dire dont l'adresse de début de chargement est lue dans le fichier lui-même, il faut utiliser l'adresse secondaire 1. En sortie, les registres X et Y contiennent (partie basse, partie haute) l'adresse du dernier octet chargé +1. Elle modifie les registres A, X et Y.

Appel par :

```
LOADPR          LDA £$08     ;LECTEUR DE DISQUES
                LDX £$05     ;FICHIER LOGIQUE 5
                LDY £$01     ;ABSOLU
                JSR SETLFS   ;LOAD"...",8,1
                LDA £$05     ;NOM DE LONGUEUR 5
                LDX £<NOM
                LDY £>NOM    ;ADRESSE DU NOM
                JSR SETNAM   ;LOAD"ESSAI",8,1
                LDX £$00     ;ADRESSE DE CHARGEMENT
                LDY £$00     ;SANS IMPORTANCE
                LDA £$00     ;LOAD,PAS VERIFY
                JSR $FFD5    ;EFFECTUE LOAD
                STX FINAD    ;RANGE L'ADRESSE DE FIN
                STY FINAD+1  ;EN FINAD
                RTS

NOM             .BYTE 'ESSAI'     ;NOM
FINAD           .WORD              ;ICI SE METTRA
                                   ;L'ADRESSE DE FIN
```

NOTE: Si le 08 est remplacé par 01 pour accéder à la cassette, il faut ajouter au début les lignes ci-dessous pour obtenir à l'écran les messages du KERNAL comme 'PRESS PLAY...',etc...

```
                LDA £$80
                STA $9D    ;AUTORISE MESSAGES
```

$FFD8 -SAVE Cette routine sauve sur un périphérique une zone de la mémoire. Les périphériques 0, 2 et 3 ne peuvent pas être utilisés. Le nom de fichier est obligatoire, sauf pour la cassette (périphérique 1). Elle doit être précédée de SETLFS et SETNAM. Ele modifie les registres A, X et Y. L'erreur éventuelle peut être lue ensuite par READST. L'adresse de début de zone à sauver doit être rangée dans deux adresses consécutives (partie basse, partie haute) en page 0, et l'accumulateur A doit contenir en entrée le pointeur vers cette adresse en page 0. En entrée, les registres X et Y contiennent l'adresse (partie basse, partie haute) du dernier octet à sauver +1.

Appel par :

```
;SOIT LE POINTEUR DE DEPART EN $64
;DANS L'ACCUMULATEUR FLOTTANT. ON
;SAUVERA LES MEMOIRES DE $4000 A
;$42B0
;SUR DISQUETTE COMME PAR UN
;SAVE"ESSAI",8
;
DEBUT = $4000
FIN   = $42B1
LDA £$08      ;DISQUETTE
JSR SETLFS
LDA £$05      ;LONGUEUR DU NOM
LDX £<NOM
LDY £>NOM     ;ADRESSE DU NOM
JSR SETNAM
LDA £<DEBUT   ;PARTIE BASSE
STA $64
LDA £>DEBUT   ;PARTIE HAUTE
STA $65
LDA £$64      ;POINTEUR DE DEBUT
LDX £<FIN     ;PARTIE BASSE
LDY £>FIN     ;PARTIE HAUTE
JSR $FFD8     ;SAVE
```

$FFDB -SETTIM Remise à jour de l'horloge 'TI'. Cette routine utilise et modifie A, X et Y. La valeur de l'horloge TI est à décrire en entrée sur 3 octets: Partie haute dans A, partie moyenne dans X et partie basse dans Y. La valeur s'exprime en soixantièmes de secondes.

Appel par :

```
LDA £$00      ;PARTIE BASSE
LDX £$00      ;PARTIE MOYENNE
```

```
                    LDY £$00    ;PARTIE HAUTE
                    JSR $FFDB
```

$FFDE -RDTIM Cette routine lit l'horloge. Elle utilise les registres A, X et Y dans le même ordre que ci-dessus.

```
Appel par  :  JSR $FFDE
              STY HEURE
              STX HEURE+1
              STY HEURE+2
```

$FFE1 -STOP Cette routine teste la touche STOP. Elle nécessite l'appel préalable de UDTIM pour lire le clavier si les interruptions ont été interdites. En effet, dans le cas où la routine d'interruption normale n'est pas en fonction, c'est la seule façon d'arrêter un programme coincé dans une boucle infinie. En sortie, le flag Z est remis à 1 et les canaux d'entrée/sortie sont ré-aiguillés vers le clavier et l'écran. En sortie, l'accumulateur A contient le résultat du balayage de la ligne clavierqui contient la touche STOP. On peut ainsi tester l'enfoncement du SHIFT.

```
Appel par  :  JSR UDTIM
              JSR $FFE1
              BNE SUITE   ;SI PAS STOP
              JMP BASIC   ;SI STOP
     SUITE    .....
```

$FFE4 -GETIN Cette routine lit un caractère au clavier au sur l'RS-232. Si le canal d'entrée est aiguillé vers le clavier, ce qui est la situation la plus fréquente, un caractère est pris (et donc enlevé) du tampon de clavier. Si aucun caractére n'est présent, la valeur lue est 0. On peut à ce moment faire une boucle d'attente jusqu'à la frappe d'un caractère. C'est donc le GET A$ du BASIC. Si le canal est aiguillé sur l'RS-232, un caractère est lu dans le tampon de reception de l'RS-232.

Appel par :

```
     BOUCLE   JSR $FFE4
              CMP £$00    ;SI RIEN,ON BOUCLE
              BEQ BOUCLE
              STA CARACT  ;OCTET LU
```

$FFE7 -CLALL Fermeture de tous les fichiers ouverts. Cette routine appelle CLRCHN pour rétablir les périphériques clavier et écran. Il est de bonne pratique d'appeller CLALL au début d'un programme pour éviter tous problèmes.

Appel par : JSR $FFE7

$FFEA -UDTIM Cette routine incrémente l'horloge d'un soixan-

tième de seconde. Si un programme nécessite une routine d'interruption spéciale (un balayage de clavier ou de joystick, par exemple), on peut l'appeller 60 fois par seconde pour maintenir l'horloge à l'heure. Elle modifie A et X. Il est toutefois à conseiller d'utiliser la vraie horloge temps réel de l'un ou l'autre des 6526 pour maintenir un temps à jour. Sa précision est bien meilleure et l'incrémentation est automatique, ce qui simplifie considérablement la programmation.

Appel par : JSR $FFEA

$FFED -SCREEN Cette routine donne au programme appellant les dimensions de l'écran en nombre de caractères. Dans le cas du CBM 64, le résultat est bien sûr toujours identique: 25 et 40. Elle ne tient pas compte de la modification du mode 38/40 du VIC-II. Elle existe sur toutes les machines COMMODORE et permet à un programme conçu pour plusieurs machines de reconnaître le type de format d'écran à utiliser. Le nombre de colonnes de l'écran se retrouvera en sortie dans le registre X et le nombre de lignes dans le registre Y.

Appel par : JSR $FFED

$FFOO -PLOT Cette routine lit ou établit la position du curseur dans l'écran. En l'appellant avec le flag C mis à 1, les coordonnées horizontale et verticale sont lues dans les registres (numéro de colonne dans Y et numéro de ligne dans X). X peut donc être compris entre 0 et 24, Y entre 0 et 79. En effet, si le curseur est dans une ligne-suite, sa position se compte dans la ligne logique qui est équivalente à deux lignes physiques de 40 caractères. Ceci arrive lors de l'utilisation de l'éditeur d'écran quand le curseur dépasse une fin de ligne. A ce moment, l'éditeur d'écran maintient en mémoire le fait que les deux lignes sont à considérer comme une seule (en mettant à 1 le bit 7 de l'octet de poids fort de l'adresse de ligne dans latable en RAM- ce qui explique que le BASIC ne supporte pas d'écran à des adresses supérieures à 32767). Elle modifie les registres A, X et Y.

```
Appel par : SEC
            jsr $FFFO
```

Si en entrée, le flag C est mis à 0, le contenu de X et Y est à considérer comme une nouvelle position à imposer au curseur. En entrée, X contient le numéro de ligne et Y le numéro de colonne.

Appel par : CLC
JSR $FFF0

Note concernant le CBM SX-64.

Le Commodore SX-64 portable est compatible à 99% avec le CBM-64. Toutes les routines du système sont donc utilisables de la même façon. Le pourcent de différence vient du fait que l'interface cassette n'existe pas sur le SX-64. Listons ici les différences entre les routines ROM du CBM 64 et du SX-64.

- LOAD et SAVE rendent "ILLEGAL DEVICE ERROR" si on fait appel au périphérique 1 (la cassette).

- Le message "LOAD" <RETURN> "RUN"<RETURN> généré par la touche RUN du clavier est remplacé par :

LOAD":*",8 <RETURN> RUN <RETURN>

(aux adresses $F0D8 à $F0E6)

- Une erreur de l'RS-232 est corrigée : les adresses $EF94, $EF95, $EF96 qui contenaient $85, $A9, $60 contiennent maintenant : $4C, $D3, $E4 : soit : JMP $E4D3.

- En $E4D3, zone précédemment inutilisée (voir ci-dessus à $E4B7), on trouve maintenant :

```
$E4D3     $85  $A9  STA $A9
```

(cequi était en $EF96 dans le CBM 64)

```
$E4D5     $A9  $01  LDA  $01

          $85  $AB  STA  $AB
          $60       RTS
```

- On peut essayer cette modification sur le CBM-64 après avoir recopié les ROM en RAM.

- Les octets $ECD9 et $ECDA qui contiennent les couleurs de bord et de fond à l'allumage de la machine ($0E, bleu clair, et $06, bleu) contiennent dans le SX 64 les valeurs :
$03 (cyan) et $01 (blanc).

- La couleur d'écriture au départ se trouve en $E536. La valeur ($0E, bleu clair) devient ($06, bleu) dans le SX-64.

- Le message qui apparaît à l'allumage de la machine est rangé en $E479 à $E4AC :

"**** COMMODORE 64 BASIC..."

est remplacé par

" ***** SX-64 BASIC V2-0 *****",$0D.

Ces différences sont fort mineures mais, attention, les programmes qui utilisent la cassette ne peuvent fonctionner.

D'autre part, les programmes qui utilisaient une couleur d'écriture 6-bleu pour écrire des mots non visibles à l'écran auront leur texte bien visible en bleu sur fond blanc. C'est le cas des programmes chargeurs de routines en langage machine qui "écrivent" des caractères dans le tampon clavier.

CHAPITRE 3

LA GESTION DE L'ECRAN

Le générateur d'image du CBM 64, le circuit 6567 est fort complexe et autorise une grande variété d'effets visuels très spectaculaires. Ce circuit est usuellement appelé VIC-II, ce qui signifie "Video Interface Chip" numéro 2, le VIC-I étant le contrôleur d'écran du VIC-20.

Le fonctionnement interne du VIC-II est, bien sûr, terriblement complexe. Le VIC-II contient environ deux fois plus d'éléments internes que le microprocesseur 6510 lui-même. Il existe en deux versions : le 6567 pour l'Europe avec le codage couleur PAL et le 6566 pour les Etats-Unis avec le codage couleur NTSC. De plus, le brochage de ces deux modèles est différent.

FONCTIONNEMENT DU VIC-II

Du point de vue de l'électronicien, le VIC-II est connecté au 6510 de deux côtés différents :

Tout d'abord, 47 registres sont considérés par le 6510 comme des cases mémoires ordinaires. Ce sont les valeurs de ces 47 registres qui contrôlent entièrement le fonctionnement du VIC-II. Ces 47 adresses se trouvent en 53248 à 53294 ($D000 à $D02E) dans la couche C.

De l'autre côté, le VIC-II est connecté au bus d'adresses et de données du CBM 64. Le VIC-II posséde 14 fils d'adresses et 12 fils de données. C'est donc un processeur "12 bits". Comme on le sait, le 6510, lui, possède des bus d'adresses à 16 et de données à 8 bits, ce qui rend les deux circuits difficilement compatibles.

Comment se résoud le problème ?

Les 14 bits d'adresses du VIC-II lui permettent d'accéder à 16 K de mémoire et non 64 K. Donc seulement un quart de la

mémoire est accessible au VIC-II. Parmi les 4 configurations possibles, laquelle utilisera-t-on ? Le choix en est heureusement laissé au programmeur. Les concepteurs du CBM-64 ont en effet ajouté 2 bits programmables dans le décodage d'adresse du VIC-II. Ces deux bits qui sont programmés dans le circuit CIA-2, permettent de choisir le bloc de 16 K qui sera utilisé par le VIC-II.

numéro du banc mémoire VIC-II	adresse pour 6510	adresse pour VIC-II	caractères en ROM	lecture de mémoire par
BN = 0	0-16383	0-16383	OUI	PEEK
BN = 1	-32767	0-16383	NON	PEEK
BN = 2	-49158	0-16383	OUI	PEEK
BN = 3	-65535	0-16383	NON	USR *

*: voir USR-PEEK

Par défaut, à l'allumage du CBM-64, le banc 0 est sélectionné. L'écran est donc nécessairement à ce moment dans les 16 premiers K-octets de mémoire.

Si on désire changer de banc, voici la procédure :

```
POKE 56578,  PEEK (56578)OR 3 : REM REGISTRE 6526 EN SORTIE
POKE 56576, (PEEK (56576)AND 252)OR(3-BN)
```

BN est bien entendu, le numéro de Banc-Mémoire repris au tableau ci-dessus.

Si on désire travailler dans le banc 0, il est préférable de relever le pointeur de bas de BASIC pour le mettre en $4000 (16384). Ceci rend le premier bloc de 16 K intouchable par BASIC et permet donc d'éviter nombre de fâcheuses interférences entre BASIC et VIC-II.

Le programme ci-dessous est à utiliser dans ce cas avant tout usage du BASIC. On préservera ainsi les images graphiques au maximum. La zone réservée au texte BASIC perd ainsi 14 K, mais avec 24 K, on peut déjà créer un énorme programme. Voir aussi le programme INIMEM en annexe.

Pour déplacer le bas de BASIC en 16384 (soit 64*256):

```
10 POKE 16384,0 : POKE 44,64
20 NEW
```

Il existe une seconde particularité intéressante du décodage d'adresse du VIC-II : le dessin des caractères standard du CBM-64 est contenu en mémoire ROM dans le banc 3 en $D000 à $DFFF. Mais

ceci n'est valable que pour le 6510. En effet, pour le VIC-II, cette ROM générateur de caractères est présente dans les bancs mémoire 0 et 2 aux adresses $1000 à $1FFF. Il est donc compréhensible que, même en utilisant le banc 0,on dispose ainsi des caractères normaux dès l'allumage de la machine.

L'écran standard est programmé en Mode 1 (voir plus loin), c'est-à-dire que l'écran est subdivisé en 1000 octets - 25 lignes de 40 caractères - qui sont présents dans le banc 0 aux adresses 1024 à 2023 ($0400 à $07E7). Chacune de ces cases mémoire peut contenir 256 valeurs différentes (0-255) qui donnent le numéro du caractère à afficher à cet endroit.

Mais, on s'en rappelle, le VIC-II adresse non pas 8 mais 12 bits de données à la fois. Les 4 bits restants proviennent, par une dernière astuce du décodeur d'adresse du VIC-II, des adresses 55296 à 56295 ($D800 à $DCE7) qui contiennent chacune 4 bits, c'est-à-dire 16 valeurs possibles qui donnent la couleur d'affichage de ce caractère. Cette mémoire est d'adresse immuable, quel que soit le banc mémoire choisi pour le VIC-II. Nous l'appellerons "RAM Couleur".

Quelle configuration faut-il utiliser de préférence ?

Nous verrons ci-après les différents modes graphiques en détail. Ils peuvent se subdiviser en deux groupes : les modes caractères et les modes BITMAP (haute résolution). Le premier groupe nécessitant souvent l'accès au générateur de caractères ROM, on utilisera donc le banc 0 ou le banc 2. Ce dernier sera préféré si le CBM-64 n'est pas équipé d'extensions ROM et si on veut programmer de nombreux caractères et lutins, employer plusieurs écrans, etc. En effet le banc 0 n'offre que 15 K et non 16 K utiles : les 1000 premiers octets étant occupés par le CBM-64 pour son usage personnel. De plus, le déplacement obligatoire du pointeur de début de BASIC complique le chargement des programmes.

Il est bien entendu possible d'utiliser des extensions logicielles graphiques comme SIMON'S BASIC, etc... A ce moment, le choix de la configuration n'est plus possible, mais imposé par les programmes graphiques.

Avec les modes BITMAP, nous utiliserons souvent le banc 3, qui est entièrement caché, comme on l'a vu, en-dessous des ROM du BASIC et du KERNAL. Pour écrire dans cette zone de la mémoire, l'instruction POKE fonctionne sans problème. Par contre, pour relire un point de l'écran, PEEK ne rend que le contenu de la ROM et non de notre mémoire graphique. Ceci se contourne grâce à l'instruction USR du BASIC. Cette instruction est programmable. Nous programmerons donc USR pour qu'il effectue le " PEEK GRAPHIQUE" ou "PEEK RAM". Les programmes graphiques dans le banc 3 utiliseront donc simplement ?USR (62000) au lieu de ?PEEK (62000) pour lire le contenu de la mémoire RAM. (voir en annexe le programme USR-PEEK)

LES MODES GRAPHIQUES

On peut utiliser dans le CBM-64, sept modes graphiques différents. Nous les repérerons par leur numéro, bien que l'on puisse y faire référence par famille également.

Le tableau ci-dessous résume les définitions des 7 modes et l'appartenance de ceux-ci aux diverses familles graphiques.

N.Mode	Désignation	Famille de modes graphiques				
		Caractères	couleur étendue	multi-colore	BITMAP H.Res	Caract. en ROM
1	Caractères standard	X				X
2	Caractères programmables	X				
3	Caractères programmables multicolores	X		X		
4	Caractères standard (étendu)	X	X			X
5	Caractères programmables (étendu)	X	X			
6	Bitmap monochrome				X	
7	Bitmap multicolore			X	X	

Avant de détailler les différents modes graphiques, regardons encore le processus de création d'images par le VIC-II. Deux méthodes différentes existent, une pour les 5 modes caractères et une autre pour les 2 modes haute résolution.

MODES CARACTERES (1-5)

Dans les modes 1 à 5, 3 zones de la mémoire sont actives : la zone "POINTEURS", la zone "GENERATEUR" et la RAM couleur.

La zone "POINTEURS" possède toujours 1000 octets exactement correspondants aux positions de chacun des 40 caractères de chacune des 25 lignes de l'écran. Chaque octet (ou pointeur) pointe vers - c'est-à-dire donne l'adresse - du morceau de la zone "GENERATEUR" qui contient le dessin du caractère. On appelle usuellement cette zone "L'ECRAN".

La zone "GENERATEUR" contient le dessin de chaque caractère qu'il est possible d'afficher. Un caractère se représente comme une série de points éteints ou allumés de différentes couleurs. Un caractère comporte 8 lignes de points et une ligne est toujours définie par un seul octet (8 bits). Il y a donc 8 octets utilisés par caractère. On voit donc que pour trouver l'adresse du dessin d'un caractère, il faut :

1) Prendre la valeur du pointeur dans la zone "POINTEURS"
2) La multiplier par 8
3) Ajouter l'adresse de début du générateur de caractères dans la page de 16 K du VIC-II
4) Ajouter 16384*(numéro du banc-mémoire VIC-II)

Parallèlement, la RAM Couleur, d'adresse immuable (55296) possède 1000 quartets de longueur. Un quartet est un demi-octet ou groupe de 4 bits. Les valeurs d'un quartet vont donc de 0 à (2 exposant 4)-1 soit 15. Chaque quartet contient donc 4 informations précisant quelles couleurs sont utilisées pour représenter le caractère dont le pointeur a une position similaire dans la zone "POINTEURS".

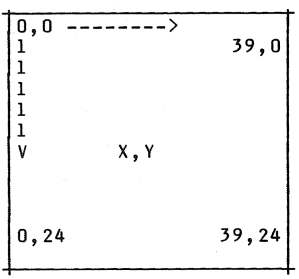
Pour faciliter la compréhension des exemples de ce livre, définissons d'abord comment calculer la position en mémoire du pointeur d'un caractère dont on connaît les coordonnées dans l'écran : si l'adresse du début de la zone "POINTEURS" s'appelle BASE, l'adresse d'un pointeur se calcule ainsi :

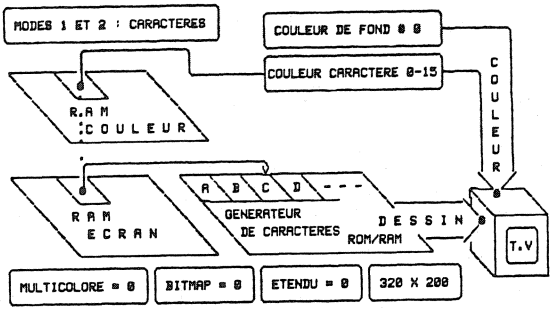
POINTEUR = BASE + X + 40*Y

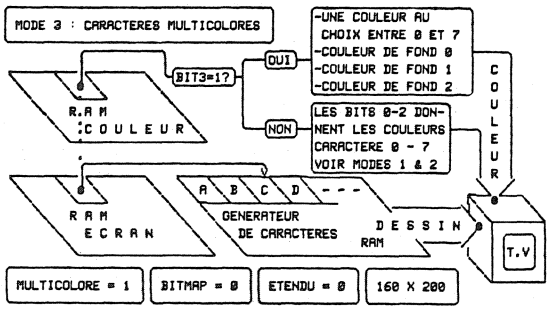
avec X compris entre 0 et 39 inclus
et Y compris entre 0 et 24 inclus.

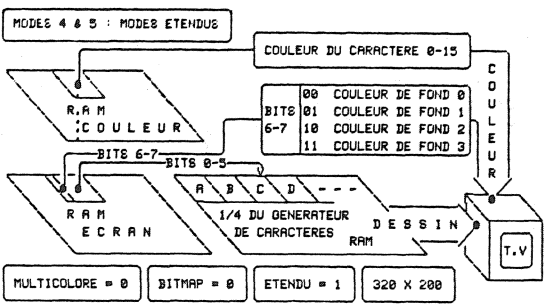
De même,

COULEUR = 55296 + X + 40*Y.




MODES 1 ET 2 : CARACTERES
COULEUR DE FOND 0 0
COULEUR CARACTERE 0-15
COULEUR
R.A M
:C O U L E U R
R A M
E C R A N
A B C D - - -
GENERATEUR
DE CARACTERES
D E S S I N
ROM/RAM
T.V
MULTICOLORE = 0
BITMAP = 0
ETENDU = 0
320 X 200

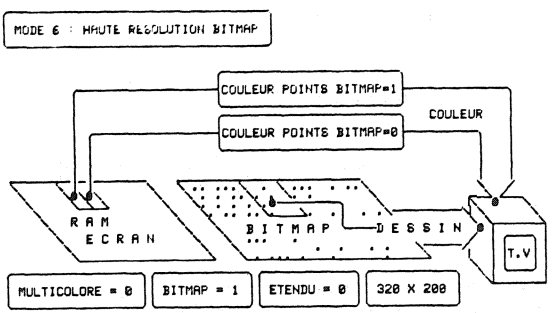

MODE 3 : CARACTERES MULTICOLORES
-UNE COULEUR AU
CHOIX ENTRE 0 ET 7
-COULEUR DE FOND 0
-COULEUR DE FOND 1
-COULEUR DE FOND 2
OUI
BIT3=1?
NON
LES BITS 0-2 DON-
NENT LES COULEURS
CARACTERE 0 - 7
VOIR MODES 1 & 2
COULEUR
R.A M
:C O U L E U R
R A M
E C R A N
A B C D - - -
GENERATEUR
DE CARACTERES
D E S S I N
RAM
T.V
MULTICOLORE = 1
BITMAP = 0
ETENDU = 0
160 X 200

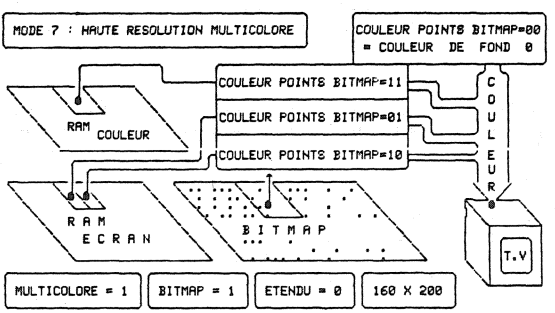

MODES 4 & 5 : MODES ETENDUS
COULEUR DU CARACTERE 0-15
COULEUR
R.A M
:C O U L E U R
BITS
6-7
00 COULEUR DE FOND 0
01 COULEUR DE FOND 1
10 COULEUR DE FOND 2
11 COULEUR DE FOND 3
BITS 6-7
BITS 0-5
R A M
E C R A N
A B C D - - -
1/4 DU GENERATEUR
DE CARACTERES
D E S S I N
RAM
T.V
MULTICOLORE = 0
BITMAP = 0
ETENDU = 1
320 X 200

MODES HAUTE RESOLUTION (6 - 7)

L'écran du CBM-64 peut être considéré comme une carte (MAP en anglais) de 320 points horizontaux sur 200 points verticaux, soit 64.000 points. A raison d'un bit par point ou PIXEL (de PICture ELement ou élémment de dessin) et de 8 bits par octet, on comprend la nécessité d'occuper 8.000 octets pour représenter un écran. Il faut donc écrire huit octets dans l'écran pour obtenir un dessin de 8 x 8 Pixels, ce qui est bien plus lent que l'emploi d'un seul caractère conmme dans les modes 1 à 5.

Mais l'avantage de cette solution est qu'il n'y a plus de comtraintes sur la position des points allumés ou éteints. Dans les modes caractères, il est impossible d'avoir plus de 256 caractères différents. En mode 6, on peut en créer 65536 !!





Le choix de la couleur de chaque point est cependant limité. En effet, les informations de couleur sont contenues dans la mémoire "RAM Couleur" et dans la mémoire "POINTEURS", deux zones longues de 1000 octets. On sera donc limité à choisir les couleurs globalement pour une zone de 8 x 8 Pixels (la zone couverte par un caractère en mode normal).

MODES CARACTERES ETENDUS

En sacrificiant le nombre de caractères possibles à l'écran, qui passe de 256 à 64, on récupère 2 bits par octet de la RAM "POINTEURS", que l'on peut affecter au choix de la couleur de fond de ce caractère.

C'est de loin le mode le plus simple à utiliser pour obtenir des résultats spectaculaires sans sacrifier la vitesse d'exécution du programme à sa qualité, comme en mode BITMAP.
(Voir le programme "SVP")

MODES MULTICOLORES

En sacrifiant encore davantage la résolution qui est alors réduite de moitié en horizontal, on dispose d'une palette de couleurs deux fois plus étendue.

COMMENT PROGRAMMER L'ADRESSE DE L'ECRAN ET DU GENERATEUR DE CARACTERES ?

Le choix des adresses possibles pour ces deux zones de mémoire est fort vaste. Cependant, peu sont vraiment utilisables en pratique.

Nous avons vu plus haut comment choisir un banc mémoire pour le VIC-II, il nous faut aussi choisir une adresse pour la zone des pointeurs et du générateur.

Si A = Pointeur écran
B = Pointeur générateur de caractères dans le banc de 16 K du VIC-II
BN = Numéro du banc - mémoire du VIC-II

On programme écran et générateur par :

```
POKE 53272, A+B
```

Adresse réelle écran :

```
EC = (3-(PEEK(56578)AND3))*16384+A*64
```

Adresse réelle générateur :

```
GEN = (3-(PEEK(56578)AND3))*16384+B*2048
```

sauf si BN = 0 ou 2 et B = 4 ou 6, auquel cas :

```
GEN = 53248 (ROM dans le banc B)   (B=4)
GEN = 55296 (ROM dans le banc B)   (B=6)
```

où A et B proviennent du tableau ci-dessous.

ADRESSE ECRAN		VALEUR
DECIMAL	HEXA	A
0	$0000	0
1024	$0400	16
2048	$0800	32
3072	$0C00	48
4096	$1000	* 64
5120	$1400	* 80
6144	$1800	* 96
7168	$1C00	* 112
8192	$2000	128
9216	$2400	144
10240	$2800	160
11264	$2C00	176
12288	$3000	192
13312	$3400	208
14336	$3800	224
15360	$3C00	240

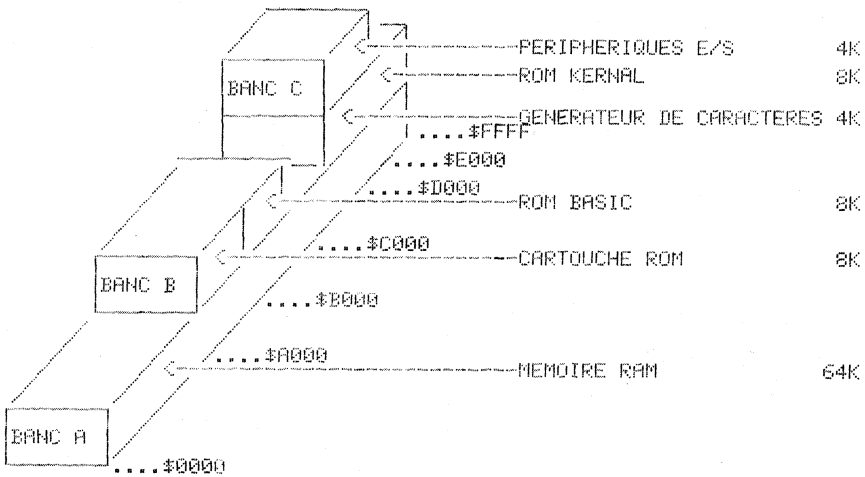
ADR. GENERATEUR		VALEUR
DECIMAL	HEXA	B
0	$0000	0
2048	$0800	2
4096	$1000	+ 4
6144	$1800	+ 6
8192	$2000	8
10240	$2800	10
12288	$3000	12
14336	$3800	14

\+ B= 4,6 : ROM dans les bancs 0 et 2

* peuvent être inaccessibles au VIC-II si le générateur de caractères ROM est à la même adresse dans les bancs 0 et 2.

LA MEMOIRE ROM GENERATEUR DE CARACTERES

Le générateur de caractères est contenu dans une ROM de 4 K située en $D000 dans le banc B. Elle est toujours accessible au VIC-II si le banc mémoire 0 ou le banc mémoire 2 est actif.



On peut alors sélectionner B = 4 ou B = 6, ce qui donne accès à deux zones différentes de la ROM. B = 4 rend active la zone $1000 à $17FF qui contient les 256 caractères disponibles à l'allumage de la machine : majuscules + graphiques.

B = 6 donne accès à la zone $1800 à $1FFF qui contient les caractères accessibles après la frappe simultanée de C= et de SHIFT au clavier : majuscules + minuscules.

Le générateur peut être décomposé en huit zones de 64 caractères. On en verra la raison plus bas : le mode caractère étendu n'accède en effet qu'à 64 caractères différents à la fois.

NUMERO DE ZONE DE 64 CARACT.	ADRESSE POUR VIC-II	ADRESSE POUR 6510	CONTENU
0	$1000	53248	Majuscules
1	$1200	53760	Graphiques
2	$1400	54272	Majuscules inverses
3	$1600	54784	Graphiques inverses
4	$1800	55296	Minuscules
5	$1A00	55808	Majuscules+graphiques
6	$1C00	56320	Minuscules inverses
7	$1E00	56832	Maj.+graph. inverses

L'adresse du générateur de caractères en ROM est la même que celle des circuits d'Entrée/Sortie. Ceci s'explique bien entendu par le phénomène de la commutation des bancs-mémoire.

Normalement la zone $D000 ne contient donc pas le générateur de caractères. Il est cependant très utile de pouvoir lire cette ROM pour la recopier en RAM, dans le but d'obtenir une base de départ pour un générateur de caractères programmable. On doit donc mettre les E/S hors circuit, brancher la ROM, la lire et remettre ensuite les E/S en service.

Attention!! Les E/S étant inaccessibles, il ne faut faire aucun accès à des périphériques pendant le transfert.

Deux méthodes sont possibles : BASIC ou langage Machine.

Voici la méthode BASIC directe :

```
0 RAM = ....  : ROM = ....  : MAX = ....
1 POKE 56334,PEEK(56334)AND 254
2 POKE 1,PEEK(1)AND 251
3 FORI=0 TO MAX : POKE RAM+I,PEEK(ROM+I) : NEXT I
4 POKE 1,PEEK(1)OR 4
5 POKE 56334,PEEK(56334)OR1
```

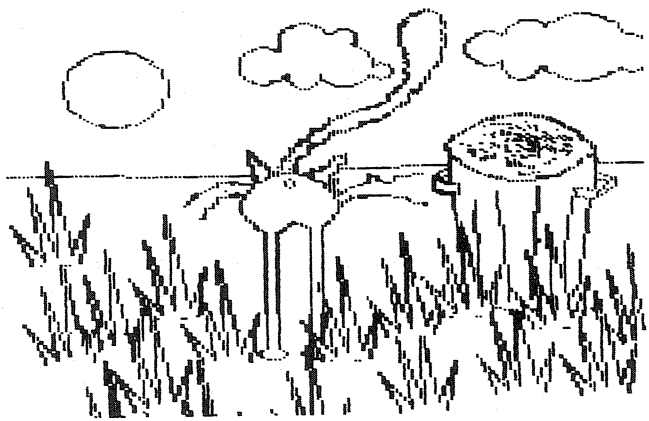
La ligne 1 supprime les interruptions normales : si on emploie d'autres interruptions, comme en modes graphiques mixtes, effectuer aussi un STOP+RESTORE avant le programme ci-dessus.

La ligne 2 met les E/S hors service.

La ligne 3 transfère MAX octets de la ROM à partir de l'adresse ROM vers la mémoire normale à partir de l'adresse RAM.

La ligne 4 remet les E/S en service.

La ligne 5 rétablit les interruptions du clavier, etc.



LA COULEUR DE L'ECRAN

La couleur du fond se programme en 53281 :
la couleur du bord se programme en 53280 :

le choix des couleurs se fait parmi les valeurs 0 à 15.

0	NOIR
1	BLANC
2	ROUGE
3	CYAN
4	MAUVE
5	VERT
6	BLEU
7	JAUNE

8	ORANGE
9	BRUN
10	ROSE
11	GRIS FONCE
12	GRIS MOYEN
13	VERT CLAIR
14	BLEU CLAIR
15	GRIS PALE

La couleur du tracé, réalisé avec PRINT peut se contrôler avec les codes suivants :

CHR$(xxx)	COULEUR TRACE	N. COULEUR
144	NOIR	0
5	BLANC	1
28	ROUGE	2
159	CYAN	3
156	MAUVE	4
30	VERT	5
31	BLEU	6
158	JAUNE	7
129	ORANGE	8
149	BRUN	9
150	ROSE	10
151	GRIS MOYEN	11
152	GRIS FONCE	12
153	VERT CLAIR	13
154	BLEU CLAIR	14
155	GRIS PALE	15

On peut bien entendu inverser couleur de fond et couleur de caractère par :

```
PRINT CHR$(18)     Inversion
PRINT CHR$(146)    Fin d'inversion
```

LE MODE GRAPHIQUE NUMERO 1

Le mode 1, appellé aussi mode "caractères" est disponible dès l'allumage de la machine : le dessin des caractères est pré-programmé dans la ROM générateur de caractères.

Toutefois, il existe deux jeux de caractères possibles, les caractères Majuscules / Graphiques ou les caractères Majuscules / Minuscules.

Pour accéder à ce dernier jeu, il y a deux méthodes : en mode direct, il suffit d'appuyer simultanément sur SHIFT et sur C=. Par programme, il faut programmer le registre 53272 du VIC-II. En effet, le dessin de chaque caractère occupant 8 octets, chaque jeu de 256 caractères occupe donc 2048 octets : ils occupent les adresses 53248 ou 55296 dans le banc 8.

Pour passer en MAJ/GRAPH : POKE 53272,21 ou PRINT CHR$(142)
MAJ/MIN : POKE 53272,23 ou PRINT CHR$(14)

C'est le KERNAL qui intercepte certains caractères dans les ordres PRINT pour effectuer cette programmation. Deux autres fonctions annexes du KERNAL sont liées à celle-ci :

supprime l'effet de SHIFT + C= PRINT CHR$(8)
autorise l'effet de SHIFT + C= PRINT CHR$(9)

On trouvera en annexe les jeux de caractères ainsi que leurs codes.

La méthode la plus simple pour écrire dans le mode 1 est le PRINT. Les caractères de contrôle CHIFFRE + CTRL et CHIFFRE + C= permettent le choix des couleurs et de l'inversion couleur caractère/couleur de fond.

On peut bien entendu accéder directement à l'écran pour y inscrire un caractère avec l'instruction POKE. Si on ne travaille qu'en une seule couleur, il est très simple de "noircir" d'abord toute la RAM couleur par :

```
CO = 55296 : NOIR = 0
FOR I = CO TO CO+999 : POKE I,NOIR : NEXT I
```

L'écriture d'un caractère en position (x,y) de l'écran s'effectue alors par :

```
BASE = 1024 : REM ECRAN NORMAL
CAR = 42 : REM ASTERISQUE
POKE (BASE + X + 40*Y),CAR
```

L'adresse d'écriture varie toujours entre BASE et BASE + 999. Si l'écran n'a pas été expressément disposé ailleurs, BASE = 1024.

Si on désire dessiner en plusieurs couleurs, il faut écrire le code couleur dans la RAM COULEUR en même temps que dans l'écran :

```
CAR = 42 : VERT = 5 : REM ASTERISQUE VERTE
BASE = 1024 : CO = 55296
POKE (BASE + X + 40*Y),CAR
POKE (CO + X + 40*Y),VERT
```

Nous avons vu plus haut comment modifier le pointeur de début de BASIC pour le reporter en 16384 pour protéger le banc mémoire 0 du VIC-II. Il est aussi possible de travailler en mode 1 dans le banc-mémoire 2 du VIC-II. Pour cela, notre programme doit commencer par :

```
POKE 55,0 : POKE 56,128 : CLR
```

ceci pour protéger la zone mémoire 2 du BASIC. A ce moment, plusieurs adresses sont possibles pour l'écran : voir le paragraphe "comment programmer l'adresse de l'écran". Par exemple, pour avoir l'écran au début du banc 2 en 32768 :

Adresse écran = 32768 ----> A = 0

Adresse écran dans le banc 2 du VIC-II = 0

Adresse générateur de caractères (normal) pour le VIC-II = 4096 ----> B = 4

POKE 53272, A + B

Les adresses utilisables en BASIC dans le banc 2 du VIC-II vont de 32768 à 40960 soit 8 K. Pour le VIC-II, 2 K sont occupés par le générateur de caractères qu'il voit aux adresses 4096 à 6143 du bloc 2. Restent donc disponibles comme écrans utilisables les quatre premiers et les deux derniers blocs de 1 K du banc 2. (A = 0, 16, 32, 48, 96 et 112 dans la formule ci-dessus).

Attention, la RAM COULEUR étant commune à tous ces écrans, on ne peut passer sans problème de l'un à l'autre que si la RAM COULEUR est au préalable "noircie" - voir ci-dessus -.

Attention toutefois, le BASIC n'apprécie pas ce déplacement d'écran brutal car il utilise des pointeurs d'écran que nous n'avons pas modifiés lors de la manipulation ci-dessus. De plus, l'éditeur d'écran et donc les instructions PRINT ne fonctionnent pas si l'écran est à une adresse égale ou supérieure à 32768.

Conservons donc l'écran normal en 1024 pour le travail avec PRINT, tandis que les zones dans le banc 2 peuvent servir de pages graphiques accédées par POKE seulement.

Une seule instruction BASIC POKE est nécessaire pour changer l'adresse de l'écran. Avec 6 écrans possibles que l'on peut commuter rapidement, la porte est ouverte à la réalisation d'images animées.

Autre usage possible : Après avoir affiché un écran qui occupe l'attention de l'utilisateur pendant quelques dizaines de secondes, le programme crée dans une autre zone-mémoire le dessin de l'image suivante. Le temps apparent d'exécution de l'image devient ainsi virtuellement nul.

En annexe, on trouvera le programme "DESSIN" qui n'emploie que deux caractères ordinaires pour tracer des figures à l'écran.

De même, le programme "DESSIN MOYRES" n'utilise que 16 caractères différents. Ces programmes sont assez commentés pour être compris aisément.

Passons maintenant en revue les caractères graphiques les plus utiles:

Les caractères à barrettes:

horizontales			verticales		
	POKE	CHR$		POKE	CHR$
a fond en haut 1	99	195	à fond à gauche 1	101	197
2	69	197	2	84	212
3	68	196	3	71	199
4	67	195	4	66	194
5	64	192	5	93	221
6	70	198	6	72	200
7	82	210	7	89	217
à fond en bas 8	100	196	à fond à droite 8	103	199

Les caractères en 8*n points :

Il y a quatre sous-groupes : les caractères noircis sur toute la hauteur mais seulement sur n huitièmes de la largeur cadrés à gauche ou à droite et ceux noircis sur toute la largeur mais seulement sur n huitièmes de la hauteur cadrés à partir du haut ou du bas.

noircis à partir de la gauche			noircis à partir de la droite		
taille	POKE	CHR$	taille	POKE	CHR$
1/8	101	165	1/8	103	167
2/8	116	180	2/8	106	170
3/8	117	181	3/8	118	182
4/8	97	161	4/8	225	161 *
5/8	246	182 *	5/8	245	181 *
6/8	234	170 *	6/8	244	180 *
7/8	231	167 *	7/8	229	165 *
8/8	160	32 *	8/8	160	32 *

noircis à partir du bas			noircis à partir du haut		
taille	POKE	CHR$	taille	POKE	CHR$
1/8	100	164	1/8	99	163
2/8	111	175	2/8	119	183
3/8	121	185	3/8	120	184
4/8	98	162	4/8	226	162 *
5/8	248	184 *	5/8	249	185 *
6/8	247	183 *	6/8	239	175 *
7/8	227	163 *	7/8	228	164 *
8/8	160	32 *	8/8	160	32 *

* : sur fond inversé : précédé de 'REV' et suivi de 'OFF'.

Les caractères blocs 4*4 points.

Le CBM 64 contient tous les caractères possibles pour obtenir une résolution par 1/4 de caractères dans les deux axes. Des effets très spéciaux peuvent être obtenus avec ces caractères en mode multicolore. Pour écrire un bloc de 4*4 points dans l'écran, il convient de lire le caractère qui se trouve sous le bloc à écrire et voir si le bloc en question est déjà noirci dans le caractère. S'il ne l'est pas encore, changer en conséquence le code du caractère. Cette procédure peut paraître lourde, mais, en pratique, on constate qu'elle est environ dix fois plus rapide (en BASIC) que l'emploi du mode haute-résolution, ce qui donne des vitesses de tracé encore acceptables. On peut bien entendu employer ces caractères pour construire un graphisme à partir de PRINT et de chaînes de caractères.

Considérons un caractère bloc comme une juxtaposition de 4 carrés numérotés 1, 2, 4 et 8.

1	2
4	8

Un caractère pouvant contenir entre 0 et 4 blocs, il suffit, pour trouver le code du caractère correspondant à une configuration, d'additionner les numéros des blocs noircis dans le caractère.

Exemple :

1	2
4	8

sera le caractère numéroté
1+2+8 = 11

1	2
4	8

No du caractère	X=carré noirci				code du caractère	
	1	2	4	8	POKE	CHR$
0					32	32
1	X				126	190
2		X			124	188
3	X	X			226	162 *
4			X		123	187
5	X		X		97	161
6		X	X		255	191 *
7	X	X	X		236	172 *
8				X	108	172
9	X			X	127	191
10		X		X	225	161 *
11	X	X		X	251	187 *
12			X	X	98	162
13	X		X	X	252	188 *
14		X	X	X	254	190 *
15	X	X	X	X	160	32 *

* = caractère inversé précédé de 'REV' et suivi de 'OFF'.

On peut positionner directement le curseur en X, Y : voir ci-dessous en mode graphique 2.

LE MODE GRAPHIQUE NUMERO 2

Le mode 2 est le mode "caractères programmables". Le fonctionnement est identique au mode 1 à ceci près que le dessin des caractères est programmable. Le dessin est donc en mémoire RAM. Il faut, comme en mode 1, réserver une zone de 2 K à cet

usage. Plusieurs emplacements sont bien entendu possibles mais la meilleure des zones est le banc 3 du VIC II, c'est-à-dire les 16 K supérieurs de la machine. Nous avons vu que POKE permet d'écrire dans cette zone "derrière" la mémoire ROM du KERNAL.

L'instruction PEEK ne permettant pas de relire la RAM à ces adresses, on utilisera la fonction USR dans ce but. Il faut pour cela changer en mémoire le programme USR-PEEK. Autre possibilité: comme décrit plus haut, on peut relever le bas de BASIC et placer le générateur en $2000.(8192).

Dernière solution : placer le générateur de caractères en $3000 (12288) et n'employer que de courts programmes BASIC n'utilisant pas plus de 10200 octets pour le texte et les variables. N'utiliser cette méthode que pour des essais car elle est peu sûre.

Si on ne désire pas reprogrammer le dessin de tous les caractères, il est très simple de recopier le générateur d'origine en tout ou en partie à l'adresse voulue (voir COPIECARA en annexe).

Comme dans le mode 1, il ne faut pas oublier de "noircir" la totalité de la RAM COULEUR si on désire accéder à l'écran par POKE. Cependant, en mode 1 et 2, il est souvent plus aisé d'utiliser PRINT.

En employant PRINT, on peut utiliser le truc suivant pour positionner le curseur à un endroit précis de l'écran :

```
POKE 211,X      (0<=X<=39)
POKE 214,Y      (0<=Y<=24)
SYS 58732
```

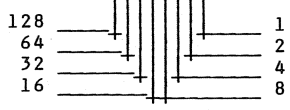
Comme on l'a vu plus haut, le dessin d'un caractère occupe 8 octets. L'adresse du premier octet du dessin d'un caractère se calcule donc comme suit :

adresse = (adresse gén.car.)+8*(code du caractère)

Le code du caractère "A commercial" est le code 0. "A" et "B" ont le code 1 et 2, etc.

Donc, soit le générateur recopié en 8192. Nous pouvons lire le dessin du "A" en 8192+8 et les 7 octets qui suivent.

ADRESSE	DESSIN	VALEUR		
		décimale	binaire	hexa
8200	oo	24	00011000	18
8201	oooo	60	00111100	3C
8202	oo oo	102	01100110	66
8203	oooooo	126	01111110	7E
8204	oo oo	102	01100110	66
8205	oo oo	102	01100110	66
8206	oo oo	102	01100110	66
8207		0	00000000	00



On peut voir qu'à chaque octet correspond une ligne de 8 points et à chaque PIXEL (ou point) d'une ligne correspond un bit.

Dans un octet, le bit le plus à droite vaut 1 si le point est allumé et 0 sinon. Le bit suivant vaut 2 ou 0, le suivant 4 ou 0,etc...

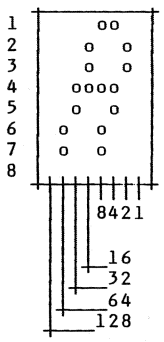
On peut aussi dire qu'un point vaut 2 élevé à la puissance N où N est la position du PIXEL dans l'octet à partir de la gauche. En effet : 2 puissance 0 = 1 et 2 puissance 7 = 128..

Reprogrammons donc un nouveau caractère: En utilisant le programme COPIECARA, on dispose d'un jeu de caractères normaux mais modifiables aux adresses 8192 à 10239.

Modifions le caractère "A". Il est donc en 8192+1*8 = 8200.

```
10 FOR I = 0 TO 7 :READ A:POKE I, A:NEXT I
20 DATA 12,18,18,60,36,72,72,0
```

PRINT "A" nous donne maintenant un "A" en italique.




```
8  + 4              = 12
16 + 2              = 18
16 + 2              = 18
32 + 16 + 8 + 4     = 60
32 + 4              = 36
64 + 8              = 72
64 + 8              = 72
 0                  =  0
```

Tous les nombres définissant un dessin de caractère sont des octets. Ils sont donc compris entre 0 et 255 inclus.

Le programme "italiques" en annexe nous donne un exemple de programmation des caractères. La ligne 170 qui modifie le dessin des caractères d'origine décale vers la gauche la moitié inférieure de chaque caractère en multipliant sa valeur par 2, tout simplement !

Si, comme presque tout le monde, vous utilisez un téléviseur couleur ordinaire, il convient d'employer pour les lignes verticales des barres de 2 PIXELS de largeur, faute de quoi l'affichage des couleurs est rarement correct.

LE MODE GRAPHIQUE NUMERO 3

Le mode 3 est encore une évolution du précédent. La différence réside dans le choix de la couleur du caractère programmable qui est plus complexe et n'autorise plus que des caractères définis en 4 * 8 PIXELS. L'avantage du mode 3 est qu'il permet l'emploi des couleurs avec plus de souplesse.

En mode 1 ou 2, un caractère ne peut posséder que 2 teintes à la fois : une couleur de tracé à choisir parmi 16 et la couleur du fond. En mode 3, chaque point peut prendre 4 teintes différentes : 3 couleurs communes, appelées couleurs de fond 0, 1 et 2 et la couleur propre au caractère comme dans les modes 1 et 2.

Couleur de bord 0 : POKE 53280, COULEUR
Couleur de bord 1 : POKE 53282, COULEUR
Couleur de bord 2 : POKE 53283, COULEUR

avec 0 <= COULEUR <= 15

La couleur du caractère est rangée dans la RAM couleur correspondant à ce caractère. Mais ici les choses se compliquent. En effet, nous n'avons pas encore "prévenu" le VIC II qu'il doit interprêter l'écran comme étant multicolore. Pour cela il y a deux conditions :

1) - Programmer le VIC-II par:

POKE 53270, PEEK (52270) OR 16

(suppression du mode multicolore par POKE 53270, PEEK(53270) AND 239).

2) - Utiliser les couleurs 8 à 15 pour le caractère considéré. C'est important car cela rend possible l'usage de caractères programmés en mode 2 et 3 à la fois sur le même écran. En effet, pour les couleurs 0 à 7, le VIC-II continuera à utiliser le mode 2 et nous pouvons donc continuer à utiliser (en 8 couleurs maximum) les caractères définis en 8*8 points.

On voit que la couleur du caractère programmé ne peut être choisie que dans une palette de 8 couleurs. Ces dernières sont d'ailleurs les couleurs numérotées 0 à 7 normalement.

Le choix des 3 couleurs de fond s'opérant dans des registres auxiliaires, on peut donc modifier à la fois par un seul POKE tous les PIXELS de l'écran qui ont cette couleur. Par exemple, on programme les trois couleurs de fond à NOIR, on trace le dessin désiré, puis en deux coups de POKE, apparaîssent les deux autres couleurs. L'effet peut être saisissant. Un exemple de programme en mode 3 se trouve repris en annexe (DESSIN MULTI-MOY).

On y retrouve toutes les commutations nécessaires: banc mémoire du VIC-II, adresse écran, adresse générateur de caractères, protection du sommet de la mémoire, copie en RAM du générateur de caractères.

Si on désire employer PRINT, ce qui est bien naturel, il est indispensable d'effectuer un effacement d'écran par PRINT CHR$(147) après avoir changé l'adresse de l'écran pour que le KERNAL recalcule les valeurs correctes des pointeurs (adresses) de chaque ligne de l'écran. Le numéro de page de l'écran (adresse écran/256) doit aussi être communiqué au KERNAL. Cette valeur sera écrite en 648.

LE MODE GRAPHIQUE NUMERO 4

Le mode 4 est aussi appelé "mode caractère étendu". C'est à nouveau une variante du mode 1 où le nombre de caractères accessibles est réduit en contre partie d'une plus grande souplesse d'utilisation des couleurs. En effet, la couleur de fond est programmable par caractère et non plus commune à tout l'écran. Par exemple, on peut écrire un mot bleu sur fond blanc dans un écran jaune. L'inconvénient est que le nombre de caractères utilisables est réduit de 256 à 64.

Les bits 6 et 7 du code caractère (code CHR$) ne font en effet plus partie du pointeur de caractère mais sont considérés comme un pointeur à 4 valeurs possibles vers un des quatre registres de couleur de fond.

Passage en mode étendu (modes 4 et 5)

```
POKE53265, PEEK(53265)OR64
```

Retour en mode 1 ou 2 :

```
POKE53265, PEEK(53265)AND191
```

En mode 4, on dispose donc toujours des caractères standard et de 16 couleurs de caractères. Les 4 couleurs de fond se programment dans les quatre registres 'couleur de fond' 0 à 3, en 53281 à 53284. Les caractères de code 0 à 63 auront le même dessin qu'en mode 1 et la couleur de fond du registre 53281. Les caractères de code 64 à 127 auront le même dessin que les codes 0 à 63 en mode 1 et la couleur de fond du registre 53282, etc...

code caractère	adresse du registre couleur de fond	adresse hexa.
0 - 63	53281	$D021
64 - 127	53282	$D022
128 - 191	53283	$D023
192 - 255	53284	$D024

Nous savons que le VIC II est un processeur "12 bits". En modes 1 et 2, 8 bits servent à définir l'adresse du dessin du caractère et 4 bits la couleur du caractère. Un seul registre commun à tout l'écran indique la couleur de fond. En modes 4 et 5, la répartition des bits entre les pointeurs est différente :

6 bits définissent l'adresse du dessin du caractère. On n'a donc plus que 64 possibilités: 64 = 2^6. Les six bits restants définissent les couleurs d'affichage : 4 bits pour la couleur des points "1" du générateur et 2 bits pour la couleur des points "0" du générateur.

LE MODE GRAPHIQUE NUMERO 5

Le mode graphique 5 est au mode 2 ce que le 4 est au 1. C'est-à-dire que les mêmes manipulations des couleurs de fond sont possibles. Seulement le générateur de caractères n'étant plus composé que de 64 * 8 = 512 octets au lieu de 2K, il ne sert à rien de faire une recopie de tous les caractères de la ROM en RAM.

Par contre, comme on l'a vu ci-dessus, en mode 4, on n'a accès qu'aux 64 premiers caractères définis en ROM. Le mode 5 permet par simple recopie d'un ou de plusieurs morceeaux de la ROM générateur de caractères, de reconstituer en RAM, un jeu de 64 caractères provenant de divers endroits de la ROM (minuscules, graphiques). De plus, la programmation des caractères conserve toute la souplesse du mode 2.

LE MODE GRAPHIQUE NUMERO 6

Le mode 6 est appelé aussi "haute résolution" ou "BITMAP". En anglais, BITMAP signifie carte des bits, c'est-à-dire que l'image est constituée de points que l'on peut allumer ou éteindre indépendamment l'un de l'autre.

La notion de générateur de caractères disparaît au profit du 'BITMAP'. Un des inconvénients de ce mode est sa grande voracité en espace-mémoire. En effet, une image BITMAP occupe 8K RAM + 1K RAM écran. Cette dernière ne servant plus de zone pointeur vers un générateur de caractères contient maintenant des informations de couleur.

Un autre inconvénient des modes BITMAP est lié à ses avantages : on a accès à chaque PIXEL de l'écran mais il n'est plus possible d'écrire un caractère en une seule opération : il faut envoyer 8 octets à l'écran pour tracer un caractère. Les

programmes sont donc au minimum 8 fois plus lents et le langage machine devient indispensable. (ou le SIMON'S BASIC, et mieux encore, le FORTH).

Les avantages compensent cependant largement les désavantages : le CBM 64 atteint avec le mode 6 la qualité d'une image télévisée, 'presque une photo'. De fantastiques exemples de graphiques sont fournis avec les programmes de dessin comme le 'KOALA PAINTER'.

Le mode 6 donne accès à 64.000 points (320 en horizontal, 200 en vertical). Les couleurs accessibles dans ce mode sont restreintes à un choix de 2 couleurs par bloc de 8*8 points.

Pour entrer en mode 6, il faut modifier un bit du registre 53265 :

mode 6 :POKE 53265, PEEK (53265) OR 32

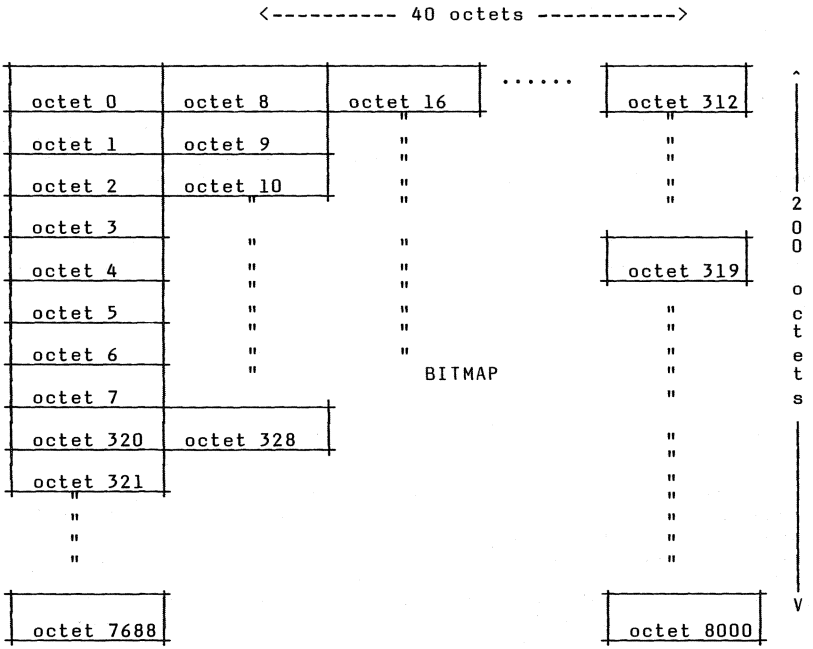
Retour mode 1 :POKE 53265, PEEK (53265) AND 223

Mais cette instruction seule ne suffit pas car il faut aussi préparer et réserver l'emplacement mémoire que le BITMAP va occuper. Nous préférons toujours, contrairement à beaucoup d'autres utilisateurs du CBM 64, positionner notre écran en haut de mémoire RAM, derrière la ROM du KERNAL, ceci dans le but de ne jamais empiéter sur la zone de mémoire BASIC de 38K (ou 30K avec les cartouches SIMON'S BASIC, ARROW, etc...). L'inconvénient de cette adresse est que le PEEK du BASIC ne peut pas relire ce que l'on a écrit à ces adresses. Il faut donc utiliser le USR-PEEK (voir en annexe).

Dans un but didactique, nous avons laissé le programme "DESSIN BITMAP" en 8192 pour bien montrer le principe. Mais dans un vrai programme utile, employez toujours USR-PEEK et les adresses hautes pour l'écran (Banc-Mémoire 3 du VIC-II).

LA COULEUR EN BITMAP

Les informations de couleur ne sont pas présentes en nombre suffisant pour autoriser le choix de 16 couleurs par points. En effet, seulement 1K de mémoire (la RAM-ECRAN) contient ces informations. Comme en modes 1 à 5, chaque octet de la RAM-écran est associé à un bloc de 8*8 points de l'écran. Il contient la couleur de tous les points allumés de ce bloc (bit =1) dans les 4 bits supérieurs. La couleur de tous les points éteints (bit=0) d'un même bloc est contenue dans les 4 bits inférieurs du même octet de la RAM-ECRAN.



Pour démarrer le mode haute-résolution, il nous faut quelques éléments de base pour programmer convenablement dans ce mode : il nous faut pouvoir allumer et éteindre un point du BITMAP, effacer tout le BITMAP, remplir la mémoire écran avec des couleurs de tracé déterminées, etc.

Considérons l'exemple du programme "DESSIN BITMAP".

1. Choisir le banc mémoire du VIC II :
   A l'allumage, nous avons le banc 0, donc on ne fait rien de spécial. Pour travailler dans un autre banc (le 3 par exemple) on utilise :
   POKE 56576, PEEK(56576) AND 252 OR (3-BN) où BN est le numéro du banc.

2. Protéger éventuellement la zone BITMAP du BASIC (Voir début de ce chapître).

3. Définir l'adresse de la RAM-ECRAN.

   ligne 50 : CO = 1024
   C'est là que l'on range les informations de

couleur.

Ligne 65 : POKE 53665, PEEK (53665) OR 32
On passe en mode BITMAP.

Ligne 67 : HO = 8192
C'est l'adresse du BITMAP.

POKE 53272, PEEK (53272) OR 8
donne l'adresse du BITMAP au VIC II

Ligne 70 : FOR C =0 TO 7999 : POKE HO+C,0 : NEXT C
efface l'écran (remet les 64.000 bits à 0).

Ligne 80 : FOR C = 0 TO 999 : POKE CO + C,1 : NEXT C
couleur de tracé = 0 (noir)
couleur de fond = 1 (blanc)pour tout l'écran.

Calcul des éléments nécessaires au tracé.

Ligne 250 : R = INT(Y/8):C = INT(X/8)

R et C sont les coordonnées verticales et horizontales du bloc de 8*8 points dans lequel se trouve le point (X,Y)

L = Y AND 7 : B = 7 - (X AND 7)

L est le numéro de la ligne (0 à 7) où se trouve le point (X,Y) dans le bloc (R,C). B est le numéro du bit dans l'octet numéro L du bloc (R,C).

BY = HO + R*320 + C*8 + L

BY est l'adresse absolue de l'octet où se trouve le point (X,Y).
Nous avons donc transformé les coordonnées absolues (X,Y) en adresse physique : octet BY, bit B.

On avait bien sûr :

$$0 < X < 319$$
$$0 < Y < 199$$
$$HO < BY < HO + 7999$$
$$0 < B < 7$$

Pour allumer (mettre à 1) un point :
Ligne 8020 : POKE BY, PEEK (BY) OR 2^B

Pour éteindre (mettre à 0) un point :
Ligne 8020 : POKE BY, PEEK(BY) AND(255-2^B)

LE MODE GRAPHIQUE NUMERO 7

Aussi appelé "mode haute résolution multicolore", le mode 7 sacrifie la résolution de l'image à la variété des couleurs. La grille BITMAP est constituée de 160 * 200 pixels, chaque pixel étant deux fois plus large qu'en mode 6. Mais pour chaque bloc de 4 sur 8 pixels, on dispose non plus de 2 mais de 4 couleurs. Une de ces couleurs est la couleur de fond 0, commune à tout l'écran. Les 3 autres ne dépendent que du bloc et sont rangées dans la RAM couleur, la partie haute et la partie basse de l'octet correspondant de la mémoire-écran.

A chaque pixel de l'écran correspondent maintenant 2 bits. Les 4 combinaisons possibles donnent accès aux 4 couleurs.

bits	Valeur	La couleur du pixel vient de :
00	0	couleur de fond 0
01	1	4 bits supérieurs mémoire écran.
10	2	4 bits inférieurs mémoire écran.
11	3	RAM couleur.

(VOIR LE TABLEAU AU DEBUT DU CHAPITRE)

On accède au mode 7 par :

```
POKE 53265, PEEK (53265) OR 32 (multicolore)
POKE 53270, PEEK (53270) OR 16 (bitmap)
```

On sort du mode 7 par :

```
POKE 53265, PEEK (53265) AND 223
POKE 53270, PEEK (53270) AND 239
```

LE DEFILEMENT

le CBM 64 possède un atout vis-à-vis de nombreux autres ordinateurs de sa catégorie : le défilement en douceur (SMOOTH SCROLLING en anglais). Le défilement de l'écran lors d'un LIST par exemple, ou dans le programme 'FIGURES' en annexe, se produit caractère par caractère, c'est-à-dire 8 pixels à la fois.

Le VIC-II est programmable d'une façon différente : l'écran change de taille et passe de 25*40 à 24*38 caractères et la bordure s'accroît d'autant. Mais l'image est toujours présente en mémoire sous la forme 25*40 caractères dont une partie seulement sont visibles.

Pour donner une impression de défilement doux d'un texte, le VIC-II déplace légèrement l'image-écran d'un pixel à la fois,

derrière la fenêtre que représente le bord de l'écran. Ce contrôle du défilement est réalisé par les registres 53270 et 53265 du VIC-II.

Procédure à suivre :

1. réduire la taille de l'écran, soit en horizontal, soit en vertical, soit les deux.

   mode 38 colonnes : POKE 53270, PEEK(53270) AND 247
   mode 24 lignes : POKE 53265, PEEK(53265) AND 247

2. positionner le registre de défilement à fond (à gauche avant un défilement vers la droite).

   Position de défilement en X (0<X<7) :
   POKE 53270, PEEK(53270) AND 248) OR X

   Position de défilement en Y (0<Y<7) :
   POKE 53265, PEEK(53265) AND 248) OR Y

3. écrire (par POKE ou PRINT) le texte voulu dans la colonne ou la ligne cachée par le bord.

4. effectuer une boucle FOR Y = 0 TO 7 - ou FOR Y = 7 TO 0 STEP-1, pour un décalage dans l'autre sens - en modifiant chaque fois la valeur Y donnée au point 2. Bien entendu, il faut remplacer Y par X pour le décalage horizontal.

5. Quand l'image-écran s'est déplacée d'un caractère entier dans une direction, il faut simultanément remettre dans les registres les valeurs de départ et recopier l'écran un caractère plus loin dans la direction du décalage. On se retrouve subitement au point 2 et ainsi de suite. Notons bien le mot 'simultanément' dans la phrase ci-dessus : la seule solution possible est de décaler l'image par un programme en langage machine, seul langage assez rapide pour donner une illusion de simultanéité.

L'ARRET D'AFFICHAGE

Il est possible d'interrompre le fonctionnement du VIC-II par programme :

pas d'affichage : POKE 53265, PEEK(53265) AND 239

affichage : POKE 53265, PEEK(53265) OR 16

Quand l'affichage est arrêté, tout l'écran apparaît de la couleur du bord. L'intérêt de ce mode est qu'il arrête tout accès à la mémoire par le VIC-II. Or, comme on l'a vu, ce dernier vole des temps de cycle au microprocesseur 6510.

On peut donc ainsi accélérer le microprocesseur. Bien plus important encore, le 6510 ne peut réaliser des délais de durée très précise que pendant l'arrêt du contrôleur VIC-II. C'est particulièrement vital lors des calculs nécessaires à l'écriture et à la lecture sur la cassette, ce qui explique que le KERNAL interrompt l'image pendant les accès cassette.

LES INTERRUPTIONS DU VIC-II

La notion d'interruption n'est guère utilisable en BASIC qui est un langage trop lent pour pouvoir utiliser ce concept efficacement. C'est par contre une des clés du sucès de nombreux programmes écrits en assembleur.

Une interruption, comme son nom l'indique, interrompt la tâche en cours dans le microprocesseur, pour lui faire exécuter une autre tâche urgente. A la fin de celle-ci, il reprend son ouvrage là où il l'avait laissé. Dans le CBM-64, il y a deux interruptions, la NMI et la IRQ. Seule cette dernière est utilisable en pratique. Nous en verrons un usage très surprenant dans le paragraphe suivant concernant les modes graphiques mixtes.

Plusieurs causes peuvent provoquer une interruption IRQ : le clavier, par exemple, interrompt le 6510 tous les 1/60 de seconde. Comment savoir quelle est la source de l'interruption ?

Le programme assembleur appelé routine d'interruption (voir l'exemple du programme MODEMIXTE-SRC en annexe) passe en revue les registres d'interruption des périphériques susceptibles d'être les générateurs de la demande de service. Cette technique s'appelle en anglais le "POLLING". Le VIC-II possède un tel registre à l'adresse 53273. Seuls 5 des 8 bits nous intéressent:

IRQ-VIC II	REGISTRE 53273 ($D019)
le bit numéro :	indique la cause :
0	balayage écran-valeur atteinte
1	collision écran-lutin
2	collision lutin-lutin
3	photostyle
7	IRQ VIC II

Si le bit 7 vaut 1, cela signifie que l'interruption provient du VIC-II. La cause peut alors être déterminée en

regardant les autres bits du même registre.
Le bit 0 indique que la ligne-écran en cours de balayage est celle dont la valeur a été indiquée au VIC-II (voir plus bas).

Le bit 1 indique qu'une collision a eu lieu entre lutins.

Le bit 2 indique qu'une collision a eu lieu entre un lutin et les données de l'écran.

Le bit 3 indique que le balayage de l'écran vient de passer devant le crayon lumineux.(photostyle).

Chaque bit du registre d'interruption ne peut être remis à zéro qu'en écrivant un '1' dans ce bit. Cette méthode peut paraître curieuse, mais elle évite de devoir garder en mémoire les bits que la routine d'interruption n'a pas encore eu le temps d'exploiter. Dès la fin d'une routine d'interruption, il faut évidemment remettre à zéro les bits qui ont causé celle-ci.

De structure semblable au registre ci-dessus, le registre d'autorisation d'interruption ($D01A-53274) ne modifie en rien le fonctionnement du registre $D019 mais permet de n'autoriser que les sources d'interruptions voulues.
Chaque bit à 1 dans le registre $D01A autorise l'événement correspondant à générer une interruption IRQ.

LE PHOTOSTYLE

Le photostyle est un crayon photo sensible qui, pointé vers l'écran envoie une impulsion au VIC-II dès que le point lumineux du balayage écran passe devant lui. A ce moment le VIC-II verrouille les coordonnées du point dans deux registres (et génère éventuellement une interruption).

Ces deux registres dont les valeurs peuvent varier de 0 à 255, se lisent comme suit :

```
X = PEEK(53267)

Y = PEEK(53268)
```

Attention, à l'achat d'un photostyle, exigez toujours une très bonne qualité : en effet, si le phototransistor est un élément à bas prix, son temps de réponse est insuffisant et la lecture de la coordonnée X sera imprécise : l'écran est balayé de gauche à droite en 64 microsecondes seulement. Il faut donc un

élément répondant à une fréquence de l'ordre de 10 mégahertz.

LES REGISTRES ECRAN DU VIC-II.

$D011 53265	reg. balay.	mode étendu	bit map	autor. image	25/23	déca- lage		
$D012 53266	registre de balayage							
$D013 53267	entrée photostyle X							
$D014 53268	entrée photostyle Y							
$D016 53270	rien		reset	multi colore	38/40	décalage horiz.		
	écran				générateur caract.			
$D018 53272	adl3	adl2	adll	adl0	adl3	adl2	adll	rien
$D019 53273	IRQ	detect.interr.---->			photo- style	collisions lutin lutin lutin	bord	balay.
$D01A 53274	autorisation	interrupt-->			photo- style	collisions lutin lutin lutin	bord	balay.
$D020 53280	rien				couleur de bord			
$D021 53281	rien				couleur de fond 0			
$D022 53282	rien				couleur de fond 1			
$D023 53283	rien				couleur de fond 2			
$D024 53284	rien				couleur de fond 3			
$D025 53285	rien				couleur lutins 0			
$D026 53286	rien				couleur lutins 1			

LES GRAPHIQUES MULTI-MODES

Les différents modes graphiques du CBM 64 affectent normalement tout l'écran. Il existe cependant une possibilité d'avoir plusieurs modes différents présents simultanément. La clé de ce secret s'appelle "interruption de balayage" (en anglais RASTER INTERRUPT).

Le contrôleur d'écran VIC-II ne peut générer des images que conformément à un seul mode à la fois. Or, l'affichage d'une image n'est pas instantané. Une ligne s'écrit en 64 microsecondes. Il suffit donc de changer la programmation du VIC-II à la fin d'une ligne pour obtenir deux modes différents simultanés : un au dessus et l'autre en dessous de la ligne où passe cette commutation. On obtient ainsi un écran partagé (en anglais, SPLIT-SCREEN). Le balayage de l'écran s'effectue 50 fois par seconde, et à chaque fois, le VIC-II génère 270 lignes horizontales superposées dont 200 forment la partie utile de l'écran. Le numéro de la ligne de balayage peut donc se coder sur 9 bits accessibles dans deux registres du VIC-II :

```
bits 0-7   : registre 53266 ($D012)
bit 8      : registre 53265 ($D011 bit 7)
```

On peùt les lire ainsi:

```
10   (PEEK(53265) AND 128), PEEK(53266)
20   GOTO 10
```

On constate que ces nombrent varient sans cesse mais le BASIC est bien trop lent pour suivre un tel phénomène qui se modifie plus de 10.000 fois par seconde ! Seul un programme en langage machine est assez rapide pour cela. Un programme,même très rapide, n'étant cependant pas instantané, on constate parfois des bizarreries sur la ligne où le changement de mode se produit.

Comment déterminer quand doit se produire la commutation ? Heureusement le VIC-II a tout prévu : il existe une possibilité de le programmer pour qu'il prévienne - cela s'appelle une interruption - le microprocesseur 6510 que la bonne ligne d'écran est en cours d'écriture. Cette possibilité est liée au registre d'autorisation d'interruption du VIC-II. Le registre de balayage a en fait deux fonctions : si on le lit, on peut voir à quelle ligne de balayage on est arrivé et si on y écrit une valeur, le VIC-II génèrera une interruption de balayage au moment où la ligne portant ce numéro sera atteinte. Bien entendu, il conviendra au préalable d'autoriser cette interruption grâce au registre d'autorisation d'interruption en 53274 ($D01A) et en écrivant un programme de gestion de cette interruption.

Nous nous proposons d'écrire ici un programme d'interruption que nous logerons à l'abri du BASIC en $C000. Nous avons besoin de deux zones de mémoire différentes pour loger nos deux images de modes différents : ici, ce sera le mode texte normal et le

mode haute résolution multicolore. Le paramètre variable de ce programme est le numéro de ligne où se passe la commutation. Ce paramètre peut être modifié par POKE à l'adresse appropriée dans le corps du programme d'interruption lui-même.

Décrivons la logique du programme. Celui-ci se sépare en deux : un morceau initialise la procédure d'interruption, l'autre est la routine d'interruption proprement dite.

MODES MIXTES

Première partie : le démarrage.

1. L'instruction SEI est employée pour interdire les interruptions pendant le cours de la routine.

2. Autoriser l'interruption de balayage en mettant à 1 le bit 0 du registre $D01A.

3. Modifier le vecteur d'interruption, c'est-à-dire indiquer au microprocesseur où se trouve la routine à exécuter lors d'une interruption. L'adresse de la routine ci-dessous doit donc être placée en $314-$315, soit 788 et 789 en décimal. Cette adresse est au format habituel c'est-à-dire partie basse en 788 et partie haute en 789.

4. interdire les interruptions clavier en écrivant 127 en 56333 ($DC0D).

5. Réautoriser les interruptions et c'est tout.

Seconde partie : l'interruption.

A l'allumage de la machine, il existe déjà une routine d'interruption qui avance l'horloge TI$ et qui lit le clavier. Cette interruption est provoquée par une minuterie. Nous avons interdit cette interruption au point 4 ci-dessus. Notre programme se terminera donc par un saut à la routine d'interruption clavier-horloge si la minuterie a terminé son décomptage.

Pour obtenir deux modes différents en simultané à l'écran, il faut bien deux modifications de la programmation du VIC-II à chaque balayage de l'écran : une au début de chacune des deux zones.

La routine d'interruption doit donc :

1. programmer l'écran pour l'un ou l'autre mode.

2. reprogrammer l'interruption de balayage pour repasser au premier mode à l'interruption suivante.

3. si nécessaire, effectuer la routine clavier.

Notre exemple sépare l'écran en deux zones : la supérieure en haute résolution, l'inférieure en mode texte. Les paramètres des deux modes sont définis dans la table en fin de programme. Un POKE à ces valeurs modifie instantanément les réglages de l'écran. Il est bien entendu possible de repasser au mode texte normal par STOP+RESTORE.

Pour sortir du mode mixte dans un programme, employez le petit programme suivant en langage machine :

```
EFFACE      SEI          ;peut se mettre n'importe où
            JSR $FD15    ;en mémoire
            JSR $FDA3
            JMP $E518
```

ou encore :

```
10 DATA 120,32,21,253,32,163,253,76,24,229
20 REM CHECKSUM = 1203
30 FOR I = 49458 TO 49467 : READA:POKE I,A:NEXT
40 SYS 49458
```

INHIBER LES ERREURS DE SYNCHRONISATION

Lorsque l'on modifie un registre du VIC-II comme couleur de fond ou couleur de bord, on observe souvent un décrochage indésirable de l'image. L'idéal consiste à attendre la fin du balayage de l'écran pour effectuer le POKE. L'effet est ainsi bien plus beau. De plus l'écran semble ainsi changer de couleur instantanément sans que l'on aperçoive un écran bicolore pendant une fraction de seconde. Cette synchronisation s'effectue en attendant le passage à 1 puis le passage à 0 du bit 7 de l'adresse $D011 (53265) par :

```
WAIT 53265, 128: WAIT 53265, 128, 128
```

Essayez le test suivant :

```
1    FOR I = 0 TO 15
2    WAIT 53265, 128: WAIT 53265, 128, 128
3    POKE 53281, I : NEXT: GOTO 1
```

Faites le test avec et sans la ligne 2. Il y a une nuance ! Le BASIC n'est cependant pas toujours assez rapide. Les 3 lignes de BASIC ci-dessus ont été traduites en langage machine : essayez le programme "GLITCH" en annexe.

CHAPITRE 4

LES LUTINS

Les lutins sont appelés "SPRITES" en anglais ou encore "MOB" ce qui est une contraction de "Mobile Object Block". Ce sont des formes graphiques, des dessins entièrement programmables en forme, en couleur, et en position. L'électronique interne du contrôleur d'écran VIC-II permet de positionner les lutins n'importe où dans l'écran. Chaque lutin possède une priorité de passage par rapport au dessin affiché à l'écran et il peut ainsi passer soit devant soit derrière les points 'allumés' de l'écran.

Il est reconnu que le concept de LUTIN est la fonction la plus intéressante que les nouveaux contrôleurs d'écran ont apportée à la micro-informatique. En effet, les LUTINS sont totalement indépendants de l'écran, d'où une extraordinaire facilité de programmation d'images animées. Indépendants du code graphique en cours, les LUTINS ne se soucient pas des contraintes sur les couleurs du fond, du bord, etc...

Les LUTINS se programment de façon fort semblable à l'écran du CBM 64. Ils peuvent être créés en mode monochrome ou multicolore et possèdent leur propre zone de définition de couleurs et de formes.

Le CBM 64 est capable d'afficher en plus de son écran graphique huit lutins simultanément sans intervention particulière. Mais, en utilisant une technique semblable au mode mixte du

chapître précédent, il est possible de définir 8 lutins par zone de l'écran. C'est à ce moment un programme d'interruption qui modifie cycliquement les contenus des registres 'LUTINS'. En utilisant cette possibilité assez impressionnante , le nombre de lutins présents à l'écran simultanément peut atteindre 32.

Les huit lutins sont numérotés de 0 à 7 et possèdent un ordre de priorité : le lutin 0 est le plus important car il passe toujours devant les 7 autres. Le lutin 1 passe toujours devant les lutins 2 à 7 mais est chaque fois masqué par le 0 s'il passe devant lui.

La priorité de passage d'un lutin devant ou derrière les points 'allumés' de l'écran est programmable. Un lutin est défini sur une grille de 24 points horizontaux sur 21 points verticaux (12 points horizontaux pour les lutins multicolores).

Chaque groupe de 8 points horizontaux se définit par un octet. Un lutin occupe donc 3x21=63 octets. Chaque lutin occupe donc en mémoire un bloc de 64 octets dont le 64ième est inutilisé.

N. de ligne L	octet L*3							octet L*3+1							octet l*3+2						
N. de bit	7	6	5	4	3	2	1	7	6	5	4	3	2	1	7	6	5	4	3	2	1
0																					
1																					
2																					
3																					
4																					
5																					
6																					
7																					
8																					
9																					
10																					
11																					
12																					
13																					
14																					
15																					
16																					
17																					
18																					
19																					

Les 63 octets sont arrangés en 21 rangées consécutives de trois octets :

```
octet 0      octet 1      octet 2
octet 3      ....
....
....
octet 60     octet 61     octet 62
```

La création du dessin d'un lutin peut se faire sur du papier quadrillé ou, plus aisément, avec l'aide d'un programme éditeur de lutins (SPRITE EDITOR) comme il existe beaucoup.

Les lutins multicolores ont, comme l'écran en mode 7, des points de double largeur définis sur 2 bits au lieu d'un, mais en quatre couleurs.

La position en mémoire d'un lutin est définissable sans problèmes: l'adresse de départ de la zone de définition étant toujours un multiple de 64, on appellera

NUMERO DE BLOC = ADRESSE/64

Le numéro de bloc est, bien entendu, compris entre 0 et 255 puisque le VIC-II n'accède qu'à 16K de mémoire à la fois et que 16K = 64*256. Il est donc bien évident que les mémoires 'LUTINS' doivent être dans le même banc de 16K que les autres pointeurs video (écran, gén. de caractères, etc...)

Avec l'écran normal à l'adresse 1024 et le BASIC débutant en 2048, seuls les blocs 13, 14 et 15 sont utilisables sans reprogrammer le VIC-II (adresses 832, 896, 960).

Ici encore, il y a un très net intérêt à déplacer l'écran dans un autre banc. Prendre le banc 2, par exemple, et le protéger du BASIC, libère les adresses 32768 à 40960 pour divers usages. On peut ainsi définir sans problème un maximum de 128 lutins (on ne peut utiliser les mêmes adresses que la ROM caractères dans le banc 2 aux adresses 36864 à 40960).

En donnant au VIC-II l'adresse de plusieurs lutins successivement, il est très aisé de créer des personages animés comme des oiseaux battant des ailes, bonshommes marchant en balançant les bras, etc...

Adresse d'un lutin = (banc mémoire VIC-II*16384)+(n.de bloc*64)

Comment donner au VIC-II l'adresse des 8 lutins qu'il doit utiliser ? Explorons à nouveau la carte-mémoire. Dans le banc de 16K alloué au VIC-II, une page de 1K (1024 octets) est attribuée à l'écran. Or, ce dernier occupe 25x40 soit 1000 octets.

Exemple: L'écran est, à l'allumage de la machine, en 1024 ($0400), la fin d'écran est en 2023 ($07E8). Le BASIC se situe en 2048 ($0800). Les 8 pointeurs de lutins sont rangés dans le reste la page écran, dans les 8 derniers octets de la page. Pour notre exemple, les octets 2040 à 2047 ($07F8 à $07FF) contiennent les numéros des blocs qui sont les images des 8 lutins actifs. Nous avons vu ci-dessus qu'un seul octet est suffisant pour définir un numéro de bloc.

Le VIC-II sait qu'un lutin est actif quand le bit correspondant vaut 1 dans le registre "AUTORISATION DE LUTINS" en 53269 ($D015). Donc, pour activer un lutin, il faut effectuer cinq opérations:

1)Dessiner et mettre en mémoire le lutin dans le bloc de 64 octets numéro "B" dans le banc de 16K actif du VIC-II. Pour l'exemple, considérons le pointeur de lutin numéro N avec 0<=N<=7.

2)Ecrire la valeur B dans la zone des pointeurs de lutins:

```
POKE (ECRAN+1016+N),B: REM 0<=N<=7
                     : REM 0<=B<=255
```

Car les pointeurs sont bien les octets 16 à 23 aprés la fin des 1000 caractéres de l'écran.

3)Prévenir le VIC-II qu'il doit afficher ce lutin:

```
POKE 53269,PEEK(53269)OR(2^N)
```

Cette manoeuvre un peu complexe, a pour but comme fréquemment dans cet ouvrage de modifier un bit d'un registre sans modifier les autres bits du même registre.

On interdit un lutin par :

```
POKE 53269,PEEK(53269)AND(255-2^N)
```

4)Prévenir le VIC-II de la position (X,Y) dans l'écran à laquelle doit se positionner le lutin.

L'écran est défini comme une matrice de 300*240 points. Un seul octet suffit donc à donner la coordonnée verticale d'un lutin mais 9 bits sont nécessaires pour la coordonnée horizontale.

Chaque lutin s'est donc vu attribuer deux registres pour X et Y, plus un neuvième bit. Les huit neuvièmes bits des huit lutins se retrouvent dans un seul registre du VIC II : le registre des "MSB X". MSB signifie en anglais MOST SIGNIFIANT BIT, ou bit de poids fort.

Soit X et Y les coordonnées du lutin numéro N. (0<=N<=7).On le positionne par :

```
POKE(53248+2*N),(X AND 255):REM coord.X 8 bits inférieurs
POKE(53249+2*N),Y           :REM coord.Y
POKE 53264,PEEK(53264)AND(255-2^N)OR(2^N)*-(X>255)
```

5)Choisir la taille du lutin. En effet, chaque lutin peut être agrandi d'un facteur 2 soit en horizontal, soit en vertical, soit les 2 à la fois.

Agrandir le lutin N horizontalement (0<=N<=7) :

```
POKE 53277, PEEK(53277) OR (2^N)
```

Taille normale en horizontal :

```
POKE 53277, PEEK(53277) AND (255-2^N)
```

De même, pour agrandir verticalement :

```
POKE 53271, PEEK(53271) OR (2^N)
```

Taille normale en vertical :

```
POKE 53271, PEEK(53271) AND (255-2^N)
```

Les positions minimales et maximales des lutins données dans de nombreux ouvrages américains sont inexactes en Europe où le contrôleur VIC-II existe en version PAL et non NTSC : les standards de télévision sont différents ainsi que les tailles et nombres de lignes des écrans. Ceci pose par ailleurs certains problèmes avec des programmes "made in USA".

Pour qu'un lutin soit entièrement visible, il faut qu'il ne dépasse pas les limites suivantes :

Largeur normale	24<X<320	pour un lutin totalement visible
	6<X<343	pour un lutin partiellement visible
Hauteur normale	50<Y<229	pour un lutin totalement visible
	30<Y<249	pour un lutin partiellement visible
Largeur double	23<X<296	pour un lutin totalement visible
	X<341	pour un lutin partiellement visible
Hauteur double	49<Y<208	pour un lutin totalement visible
	8<Y<255	pour un lutin partiellement visible

On voit qu'en largeur double il est impossible de faire disparaître le lutin à fond à gauche.

Le programme "ACTIVITE" en annexe est un exemple de l'utilisation des lutins. Quatre lutins sont utilisés pour visualiser une page de 256 octets de la mémoire du CBM-64.

On y constate que, une fois activés, les lutins restent présents et n'interfèrent pas avec le BASIC. On trouvera aussi un exemple de programmation de lutin dans les programmes "DESSIN" où trois dessins différents sont utilisés successivement pour le lutin numéro 0. Voir aussi le programme "LUTIN" de création d'un lutin à partir de DATA.

COULEURS DES LUTINS

En mode monochrome, les lutins sont aisés à programmer : chacun possède deux couleurs : sa couleur propre et la transparence : c'est à dire que tous les bits qui valent 0 dans un lutin rendent le point correspondant transparent : on voit à travers soit l'image-écran soit la couleur de fond.

La couleur propre de chaque lutin se programme comme ceci :

Le lutin numéro N (0<=N<=7) peut prendre les 16 couleurs possibles (0<=CL<=15) :

```
POKE(53287+N), CL    : rem couleur CL pour lutin N
```

Les lutins peuvent aussi être programmés en mode multicolore. Comme en mode 7, la résolution horizontale diminue de moitié et 4 couleurs sont possibles.

PAIRE DE BITS	COULEUR DU LUTIN NUMERO N
00	transparent
01	couleur lutin 0 -registre 53285
10	couleur du registre 53287+N (couleur propre)
11	couleur lutin 1 -registre 53286

Chaque paire de bits horizontaux sont considérés comme des pointeurs de couleurs, 3 étant communes à tous les lutins et une programmable par lutin, comme au paragraphe ci-dessus.
On définit un lutin multicolore par :

```
POKE 53276,PEEK(53276) OR (2^N) 0<=N<=7
```

On revient au mode monochrome par :

```
POKE 53276,PEEK(53276) AND (255-2^N)
```

LES PRIORITES DES LUTINS

Chaque lutin a priorité sur ceux de numéros plus élevés que lui : la priorité d'un lutin par rapport aux données affichées à l'écran peut se programmer. A nouveau, un seul registre contrôle

cela : chaque bit définit la priorité d'un lutin par rapport au fond. Un lutin passe devant le texte si sa priorité vaut 0, s'il passe derrière, sa priorité vaut 1.

Pour que le lutin N passe derrière le texte :

```
POKE 53275, PEEK(53275) OR (2^N)
```

Pour que le lutin passe devant le texte :

```
POKE 53275, PEEK(53275) AND (255 - 2^N)
```

LA DETECTION DES COLLISIONS

Une collision est par construction du CBM 64, un événement où un lutin rencontre un autre lutin ou le texte de fond de façon à ce que au moins un bit vale 1 pour chacun des éléments se trouvant exactement superposés à l'écran.

Une exception est toutefois faite pour la paire de bits '01' en mode multicolore qui ne déclenche pas les collisions. Ainsi pour un jeu où il convient d'éviter certains obstacles, les couleurs de l'écran 10 et 11 provoqueront des collisions, mais pas les couleurs 00 et 01.

On peut voir qu'une collision a eu lieu en testant les registres 53278 et 53279.

```
X1 = PEEK(53278) : X2 = PEEK(53279)
```

Si X1 est différent de 0, il y a eu collision entre deux lutins au moins. Si X2 est différent de 0, il y a eu collision entre un lutin au moins et les données de l'écran.

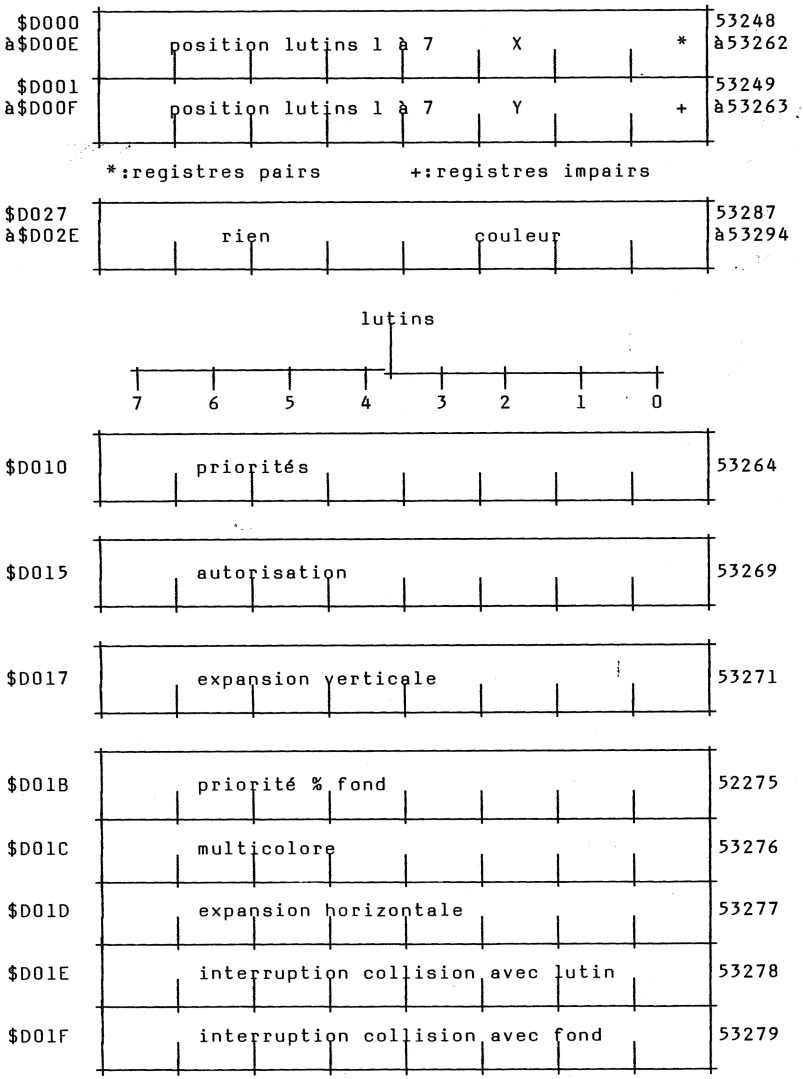
Attention !!! : le fait de lire les registres 53278 et 53279 les remet à 0 d'office. On n'a pas droit à un deuxième tour !

Pour savoir si le lutin N est impliqué dans une collision, il faut tester le bit correspondant dans X2 :

```
IF X1 AND (2^N) THEN PRINT "COLLISION"
```

détecte donc une collision entre le lutin N et un autre lutin. On voit bien qu'il faut d'abord mettre le résultat du PEEK dans une variable car dès cette lecture le registre est remis à 0 et ne peut plus être testé par IF PEEK(...) mais bien par IF X1...

LES REGISTRES DE CONTROLE DES LUTINS



CHAPITRE 5

LE SON DU 64 ET LE CIRCUIT S.I.D.

Le circuit SID est le synthétiseur musical intégré au CBM 64. Il possède trois voix, un générateur de bruit et de très nombreux registres de contrôle permettant le choix de quatre formes d'onde par voix, le choix de l'amplitude sur 48 dB. Les quatre paramètres principaux de l'enveloppe (la forme globale) de chaque voix sont également programmables.

Des fonctions plus complexes encore peuvent être obtenues par un contrôleur de modulation en anneau, un synchronisateur et trois filtres programmables. De plus, une entrée audio permet de mixer des signaux extérieurs avec la sortie du synthétiseur.

Le SID contient aussi deux entrées analogiques dénommées POT X et POT Y qui permettent de mesurer la position angulaire de deux potentiomètres extérieurs (en anglais "PADDLES"). Le SID est contrôlé par 29 registres à 8 bits:

Les 25 premiers sont des registres à écriture seulement (WRITE ONLY) et les 4 derniers sont à lecture seulement (READ ONLY). L'usage exact de chaque bit de ces registres est décrit dans le tableau récapitulatif de la page suivante.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

VOIX 1

$D400 54272	partie basse de la fréquence d'oscillation - bits 0 à 7							
$D401 54273	partie haute de la fréquence d'oscillation - bits 8 à15							
$D402 54274	largeur d'impulsion - bits 0 à 7							
$D403 54275	r i e n				largeur d'impulsion			
$D404 54276	bruit	rect.	scie.	triang	TEST	modul. anneau	sync.	DE/AR
$D405 54277	attaque (A de ADSR)				décroissance (D de ADSR)			
$D406 54278	maintien (S de ADSR)				relâchement (R de ADSR)			

VOIX 2 VOIX 3

	COMME LA VOIX 1 :	COMME LA VOIX 1 :
FREQ.BAS	$D407 - 54279	$D40E - 54286
FREQ.HTE	$D408 - 54280	$D40F - 54287
IMPULS.B	$D409 - 54281	$D410 - 54288
IMPULS.H	$D40A - 54282	$D411 - 54289
MODES	$D40B - 54283	$D412 - 54290
A D	$D40C - 54284	$D413 - 54291
S R	$D40D - 54285	$D414 - 54292

FILTRES

$D415 54293	R I E N					fréquence bits 0à2		
						bit 2		bit 0
$D416 54294	fréquence bits 3 à 10							
	bit 10							bit 3
$D417 54295	résonance bits 0 à 3				filtre actif sur :			
	bit 3			bit 0	exter	voix 3	voix 2	voix 1
$D418 54296	arrêt	passe	passe	passe	volume sonore bits 0 à 3			
	voix 3	haut	bande	bas	bit 3			bit 0

REGISTRES A LECTURE SEULE

$D419 54297	ENTREE POT X							
$D41A 54298	ENTREE POT Y							
$D41B 54299	VALEUR DE LA SORTIE DE L'OSCILLATEUR 3							
$D41C 54300	VALEUR DE LA SORTIE DE L'ENVELOPPE 3							

COMMENT PROGRAMMER UN SON ?

Un son est principalement caractérisé par:

1) son volume
2) sa fréquence
3) sa forme d'onde
4) son enveloppe

La programmation du son sur le CBM 64 nécessite d'abord d'"allumer" l'amplificateur sonore en tournant le bouton de volume. Pratiquement, on modifie les 4 bits inférieurs du registre 24 du SID :

```
                    F=0
volume à fond : POKE 54296,15
volume à 0    : POKE 54296,0
```

La fréquence de chacune des trois voix est programmable sur deux octets, soit de 0 à 65535. La notation habituelle (partie basse, partie haute) est utilisée ici.

Soit F la valeur de fréquence à programmer, FH l'octet de poids fort et FL l'octet de poids faible. Nous écrirons dans le registre partie haute la valeur :

```
FH = INT(F/256)
```

et dans le registre partie basse la valeur :

```
FL = F-FH*256
```

Le calcul de la valeur F en fonction de la fréquence désirée est donné ci-dessous. Il faut faire attention à ceci car les formules citées dans pratiquement tous les ouvrages américains reprennent des valeurs différentes des nôtres. en effet, les CBM 64 vendus aux USA ont une fréquence d'horloge différente de celle des machines européennes. Voici la bonne formule :

```
F = fréquence * 17,0284

ou   fréquence = F * 0,058725 Hz.
```

La valeur de F est de 3,8% supérieure à la valeur américaine car la fréquence d'horloge est de 1,022727 MHz aux USA contre 0,985248 MHz pour les CBM 64 européens. On trouvera en annexe le programme "fréquences" qui imprime à l'écran ou à l'imprimante les valeurs de F, FL et FH pour toutes les notes que le CBM 64 peut jouer. On y notera la façon de calculer les fréquences des notes : les notes de musique ont toutes la fréquence de la note précédente multipliée par 2 à la puissance 1/12. Les notes d'une gamme ont des fréquences doubles de celles de la gamme précédente.

Les notes sont: DO, DO dièse, RE, RE dièse, MI, FA, FA dièse, SOL, SOL dièse, LA, LA dièse, SI; et la fréquence du "LA" est de 440 Hz dans la quatrième gamme.

F(note) = F(note précédente)*2^(1/12)

Pour trouver les valeurs F, FL et FH correspondant à une fréquence donnée, on peut utiliser le programme "calcul son" en annexe.

Pour programmer la fréquence d'une voix, il suffit d'écrire les valeurs FL et FH dans les deux premiers des sept registres de contrôle de cette voix :

S = 54272

Fréquence voix 1 : POKE S + 0, FL
POKE S + 1, FH

Fréquence voix 2 : POKE S + 7, FL
POKE S + 8, FH

Fréquence voix 3 : POKE S + 14, FL
POKE S + 15, FH

Il faut noter que les registres 0 à 24 du S.I.D. ne peuvent pas être lus mais seulement écrits (WRITE ONLY).

La forme d'onde est le dessin qu'on peut obtenir en regardant le signal sonore à l'aide d'un oscilloscope. La modification de la forme d'onde change le nombre d'harmoniques du son et donc son timbre.

Qu'est-ce qu'une harmonique ? On peut dire qu'une onde sonore est une superposition d'un grand nombre d'ondes simples sinusoidales de fréquence croissante que l'on appelle les harmoniques.

Plus une forme d'onde est proche de la sinusoide, plus elle est pure. Le CBM-64 peut générer des ondes triangulaires qui se rapprochent assez bien de la sinusoide : ne sont présentes que les harmoniques impaires (la 1ère, la 3ème, etc...). Ceci donne une sonorité proche de la flûte. Il peut générer aussi des ondes en dents de scie qui contiennent toutes les harmoniques paires et impaires avec des amplitudes importantes, ce qui donne une sonorité fort riche. (trompette, etc...). Une troisième forme d'onde est possible : c'est la forme rectangulaire.

Les impulsions rectangulaires générées par le CBM-64 ont une largeur d'impulsion programmable de 0 à 4096 (soit de 0 à 100% de la largeur du rectangle). Une onde carrée parfaite peut être obtenue avec une largeur d'impulsion de 2048 (50%) qui génèrent uniquement des harmoniques impaires. Le nombre et l'amplitude des harmoniques générées dans ce mode sont très variables et permettent ainsi au S.I.D. de simuler les sonorités de très nombreux

instruments de musique. La largeur des impulsions se programme grâce à deux registres par voix. Cette programmation n'est bien sûr utile que si l'on choisit la forme d'onde rectangulaire. La largeur d'impulsion est appelée usuellement PW (Pulse Width en anglais).

Soit PW la largeur d'impulsion en % (0<=PW<=100)

```
On calcule      P  = INT(PW*40.95)      0<=P <=4096
                PH = INT(P/256)         0<=PH<=  15
                PL = P-PH*256           0<=PL<= 255
```

Programmation de la largeur d'impulsion :

```
                    S = 54252

Voix 1 :    POKE S + 2,PL
            POKE S + 3,PH

Voix 2 :    POKE S + 9,PL
            POKE S + 10,PH

Voix 3 :    POKE S + 16,PL
            POKE S + 17,PH
```

La quatrième forme d'onde possible est appelée "bruit blanc". Il s'agit en fait d'une onde de fréquence fixe mais de forme totalement aléatoire dont le contenu harmonique se modifie sans cesse. Il rappelle alors le bruit d'une chute d'eau.

Le choix d'une (ou plusieurs) formes d'ondes se fait par la mise à 1 du (ou des) bit(s) correspondant à chaque forme d'onde dans le registre de contrôle de chaque voix. Comme on le verra plus loin, on démarre une note en écrivant une valeur, appelée 'DEBUT' ou 'DE' dans le registre de contrôle. On arrête la note en écrivant la valeur "ARRET" ou "AR" dans le registre de contrôle.

Pour une forme d'onde donnée, DE et AR ont des valeurs précises :

Forme d'onde	Valeur DEBUT	Valeur ARRET
triangle	17	16
dents de scie	33	32
rectangle	65	64
bruit	129	128

Les formes d'ondes ne sont pas additives mais si plusieurs sont sélectionnées, la forme résultant de l'enveloppe est un "ET" logique des diverses formes choisies. Les seules configurations de ce type qui sont intéressantes ne font pas appel à la forme

d'onde "bruit", car, dans ce cas, le S.I.D. peut tomber dans un mode indéfini dont il ne peut sortir que grâce à un "RESET".

Forme d'onde	valeur DEBUT	valeur ARRET
portion de triangle	81	80
portion de dent de scie	97	96

Pour ces deux formes d'onde, la portion de triangle ou de dents de scie, les registres de largeur d'impulsion ont une importance fondamentale.

Les largeurs d'impulsion doivent être comprises entre PH=0 PH=8 approximativement pour que les 2 formes d'onde ci-dessus soient utilisables.

Les sonorités ainsi obtenues sont assez plaisantes (genre banjo) mais le volume sonore est plus faible qu'avec les 4 formes d'ondes simples. Il est à noter que ces deux dernières formes d'ondes ne sont pas documentées par COMMODORE.

Nous avons choisi une forme d'onde et une fréquence. Avant de lancer une note, il nous reste à définir l'enveloppe. L'enveloppe sonore d' une note est la façon dont le volume s'amplifie puis décroît au début (ATTACK et DECAY en anglais). L'enveloppe sonore se poursuit ensuite à un certain niveau plus ou moins puissant: c'est le niveau de maintien (SUSTAIN en anglais). L'enveloppe se termine par un relâchement de la note du niveau de maintien jusqu'au silence. La vitesse de ce relâchement s'appelle le taux de décroissance (RELEASE en anglais). Les initiales des mots ATTACK, DECLAY, SUSTAIN, RELEASE ont donné lieu à la contraction "ADSR" qui désigne chez COMMODORE le générateur d' enveloppe.

Le cinquième paramètre de l'enveloppe est la durée de maintien. Cette durée n'est pas programmable dans le S.I.D. mais doit être contrôlée par une boucle de délai dans le programme. Pour le programmeur, c'est la durée de la note, bien que, en réalité, la durée de la note soit la somme de la durée de maintien et de la durée de relâchement.

Voyons d'abord comment programmer les paramètres ADSR pour les trois voix du S.I.D.

Chaque paramètre A, D, S et R possède 16 valeurs possibles et nécessite donc 4 bits, soit un demi octet. Le SID regroupe donc les paramètres d'enveloppe dans 2 registres par voix : l'un pour A et D, l'autre pour S et R. Attaque, décroissance et relâchement sont des durées. Le niveau de maintien est une valeur de niveau sonore. Le paramètre 'S' (maintien) varie donc de 0 à 15, ce qui correspond à des amplitudes sonores de plus en plus gran-

des. On n'utilise que très rarement des valeurs de S inférieures à 5 sauf pour donner des effets de percussion.

```
0 <= S <= 15
```

Les valeurs possibles des paramètres A, D et R sont reprises au tableau ci-dessous.

VALEUR	DUREES EN MILLISECONDES		
	ATTAQUE A	DECROISSANCE D	RELACHEMENT R
0	2	6	6
1	8	24	24
2	16	48	48
3	24	72	72
4	38	114	114
5	56	168	168
6	68	204	204
7	80	240	240
8	100	300	300
9	250	750	750
10	500	1500	1500
11	800	2400	2400
12	1000	3000	3000
13	3000	9000	9000
14	5000	15000	15000
15	8000	24000	24000

Pour modifier le registre AD, (c'est le cinquième registre parmi les sept qui contrôlent une voix), on y inscrit la valeur:

```
AD = A*16 + D
```

Pour modifier le registre SR, (c'est le sixième registre):

```
SR = S*16 + R
```

Pour modifier les valeurs AD et SR des trois voix, il faut utiliser les instructions:

```
                    S = 54272

Voix 1              POKE S + 5,AD
                    POKE S + 6,SR

Voix 2              POKE S + 12,AD
                    POKE S + 13,SR
```

```
Voix 3          POKE S + 19,AD
                POKE S + 20,SR
```

En programmant simultanément la forme d'onde et les 4 valeurs ADSR, on peut donner à une note une sonorité tout à fait différente.

Par exemple :

```
Dents de scie ,A=5, D=8, S=5, R=9, durée = 1/2  sec :  VIOLON
Triangle      ,A=0, D=9, S=0, R=9, durée = 1/60 sec :  XYLOPHONE
Rectangle     ,A=0, D=9, S=0, R=0, durée = 1/60 sec :  PIANO
Triangle      ,A=10,D=8, S=10,R=9, durée variable   :  FLUTE
Triangle      ,A=0, D=0, S=15,R=0, durée variable   :  ORGUE
```

Il nous reste à voir comment lancer une note: heureusement, c'est très simple. Le registre 4 de chaque voix contient dans son bit 0 la clé de nos problèmes. Dès que ce bit passe de 0 à 1, le cycle 'Attaque et Décroissance' démarre, puis le niveau sonore se stabilise à la valeur de maintien exprimée, comme on l'a vu, en seizièmes du volume sonore maximum. Dès que le bit 0 du registre 4 repasse à 0, la seconde partie de l'enveloppe, c'est à dire la décroissance, se déclenche.

Pour déclencher une note, il faut utiliser les instructions suivantes :

```
                S = 54272
                DE = 17         (ou 33, 65, 129, 81, 97)
                AR = 16         (ou 32, 64, 128, 80, 96)
                DU = 250        (durée de la note en
                                 millisecondes)
Voix 1          POKE S + 4,DE
                FOR T = 0 TO DU: NEXT
                POKE S + 4,AR

Voix 2          POKE S + 11,DE
                FOR T = 0 TO DU: NEXT
                POKE S + 11,DE

Voix 3          POKE S + 18,DE
                FOR T = 0 TO DU: NEXT
                POKE S + 18,AR
```

Nous pouvons maintenant faire de la musique : on trouvera en annexe le programme "MELODIE". Modifiez-en tous les paramètres et vous serez étonnés des sonorités différentes que l'on peut en tirer.

Le programme "MELODIE" n'utilise qu'une seule voix. Il est bien certain que l'on peut déclencher et arrêter indépendamment

des notes sur chacune des trois voix pour créer des accords ou des sons plus complexes.

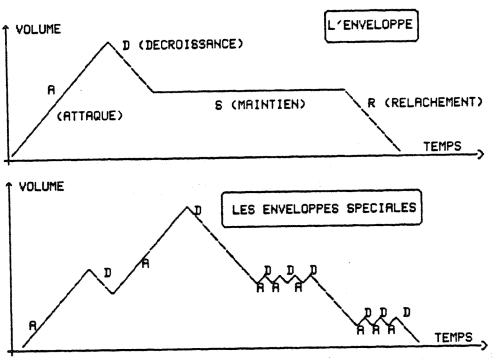
On verra un exemple d'utilisation de plusieurs voix dans le programme "HELICO" qui simule sur la voix 2 le sifflement de la turbine de l'hélicoptère et sur la voix 1, le bruit des pales avec une forme d'onde 128.

LES FONCTIONS SONORES SPECIALES

LES ENVELOPPES SPECIALES

La forme de l'enveloppe est sélectionnée comme on l'a vu ci-dessus, par les paramètres A, D, S, R et par le bit 0 du registre de contrôle. Il faut savoir que le générateur d'enveloppe, lorsqu'il est déclenché (bit de contrôle = 1), démarre la phase A (attaque) c'est à dire que le volume augmente à un taux programmé mais, et c'est là l'important, à partir du volume actuel de la voix concernée. Si la note précédente n'était pas terminée, le volume réaugmente à partir de la valeur atteinte au moment du déclenchement et non à partir de 0. Il passe ensuite, si on lui en laisse le temps, par la phase 'D' pour se stabiliser au volume 'S' de maintien.

Mais si, ici aussi, on déclenche trop tôt la remise à zéro du bit de contrôle, le cycle de décroissance du volume (R) commence instantanement, même si la note précédente est encore en phase A ou D. En conclusion, on constate qu'il est possible en contrôlant les délais entre mise à un et mise à zéro du bit de contrôle de maîtriser totalement la forme de l'enveloppe. La figure ci-dessous montre comment réaliser une enveloppe à plusieurs paliers.



Il est certain que la maîtrise totale de durées parfois très brèves est bien plus évidente en langage machine qu'en BASIC, souvent trop lent. Ici, les langages tels FORTH ou BASIC COMPILE prennent d'autant plus d'intérêt.

- Voir en annexe le programme "ENVELOPPES".

LES BALAYAGES DE PHASE

Si nous générons des sons avec une forme d'onde rectangulaire, nous disposons d'un paramètre réglable supplémentaire: la largeur d'impulsion. Cette largeur est variable de 0 à 4096. 0 et 4096 correspondent à la largeur d'impulsions de 0% et 100%. Donc, aucun son n'est produit. En ne modifiant que la valeur (PH) du registre 3 de la voix concernée, on dispose de 16 valeurs différentes, de 0 à 15. Mettons la valeur 128 dans le registre 2 (PL). Les largeurs d'impulsion correspondant a PH = 0 et PH = 15 sont donc :

PH = 0 P = 0*256 + 128 = 128 impulsion = 3%

PH = 15 P = 15*256 + 128 = 3968 impulsion = 97%

Or, on constate que ces deux impulsions sont le contraire l'une de l'autre. C'est à dire qu'elles ont des sonorités identiques mais déphasées de 180 degrés l'une par rapport à l'autre. En faisant varier rapidement la phase, on obtient un signal très riche en harmoniques diverses. Une variation aléatoire de la largeur d'impulsion donne également des effets intéresants. Par exemple des sonorités qui rappellent la balalaika.

On trouvera comme exemple en annexe le programme "BALAYAGE".

LE TREMOLO

Des effets de trémolo sont obtenus en faisant varier rapidement de façon croissante puis décroissante la valeur VL du programme "SYNCHRO" en annexe.

LA SYNCHRONISATION DES OSCILLATEURS

Chaque voix utilise le bit 1 (le bit de valeur 2) du registre 4 pour synchroniser son oscillateur principal sur celui d'une

autre voix. On peut ainsi synchroniser la fréquence de :

la voix 1 avec la voix 3 (registre 4 voix 1)
la voix 2 avec la voix 1 (registre 4 voix 2)
la voix 3 avec la voix 2 (registre 4 voix 3)

Par exemple, programmons comme usuellement la voix 3 avec une forme d'onde triangulaire et une enveloppe assez arrondie: (ADSR = 8, 6, 12, 6). Synchronisons la voix 1 avec la voix 3 en choisissant :

AR = 18 (16 + 2)

La fréquence de la note (et rien que cela) vient des registres de fréquence de la voix 3, le registre de fréquence de la voix 1 impose alors la durée d'une note. Bien entendu, si les fréquences ont été programmées identiques, rien ne se remarque. Mais si, à chaque modification de la fréquence (registre voix 3), on donne à la voix 1 une fréquence différente (plus élevée), on génère une plus ou moins grande quantité d'harmoniques complexes mais en SYNCHRONISME avec la fréquence de base. Il faut bien noter que seul le registre de fréquence est utilisé dans la voix 3 et que l'on n'est pas obligé d'écouter cette voix pour la synchroniser. On peut utiliser la fonction de synchronisation sur les 3 canaux. Cependant, il y a très peu d'intérêt à utiliser ceci en multi-voix car une seule des voix synchronisées produit un signal utile écoutable (Sauf exceptions).

LA MODULATION EN ANNEAU

Nous venons de voir comment, pour la voix 1 par exemple, nous pouvons choisir une fréquence imposée par la voix 3 et une durée de la partie utile de la note imposée par la voix 1. Nous nous proposons maintenant de garder la fréquence de la voix 1 normalement mais de déclencher, non la forme d'onde normale (triangle par exemple) mais une autre forme d'onde plus spéciale: ce sera la forme d'une onde de la voix 1 suivie d'une onde de la voix 3, suivie d'une onde de la voix 1, etc... qui constitueront la sortie de la voix 1.

Cette modulation en anneau produira, suivant le moment où la note démarre (début d'une onde voix 1, milieu d'une onde voix 2, etc...) une structure harmonique très complexe et très variable; on qualifie d'ailleurs ce type de modulation par le terme "superposition non-harmonique" tant sa décomposition en harmoniques est changeante et complexe. On obtient, comme pour la synchronisation, cet effet en modifiant un bit (ici le bit 2 de valeur 4) du registre de contrôle (le quatrième de chaque voix).

La voix 1 peut être modulée en anneau avec la voix 3
La voix 2 peut être modulée en anneau avec la voix 1
La voix 3 peut être modulée en anneau avec la voix 2

Le choix se fait en remplaçant les valeurs DE et AR par :

DE = 21 AR = 20

En effet, la modulation en anneau n'est disponible que sur les formes d'ondes triangulaires. A nouveau, le seul paramètre de la voix annexe qui est utilisé ici est la fréquence.

Les deux paragraphes ci-dessus nous amènent à conclure qu'il est souvent indésirable d'écouter la voix 3 lorsque seule sa fréquence nous intéresse pour synchroniser ou moduler la voix 1. Nous verrons plus loin que la voix 1 est en fait privilégiée car il est possible grâce au registre 24 du S.I.D. de couper totalement la sortie du 3ème oscillateur qui nous sert à moduler ou à synchroniser la voix 1.

La modulation en anneau ne peut se concevoir raisonnablement avec l'un ou l'autre des oscillateurs impliqués ayant une fréquence nulle. Usuellement, on fera varier simultanément les fréquences des deux canaux intermodulés.

On verra des exemples de ceci en annexe dans le programme "ANNEAU".

L'INTERACTION DE L'ENVELOPPE AVEC UN AUTRE PARAMETRE

Le registre 28 du SID permet de lire à tout moment l'amplitude de l'enveloppe de la voix 3 (valeurs de 0 à 255). Cette valeur peut servir par exemple à modifier la fréquence d'une des voix suivant l'intensité de la note. Les possibilités de ceci sont très nombreuses: voir entre autre un exemple dans le programme 'ANNEAU'. Il est possible ainsi, par exemple, de changer la forme d'onde pour les amplitudes inférieures à une valeur donnée.

LES FILTRES

!!! ATTENTION, à cause de problèmes de fabrication du SID chez COMMODORE, les effets obtenus avec les filtres peuvent varier fortement d'un 64 à l'autre!!! Ceci a été récemment notifié par COMMODORE aux sociétés de programmation.

L'usage de filtres de différents types permet de modifier le contenu harmonique d'une mélodie ou d'un bruit. Le SID est un synthétiseur particulièrement évolué à ce point de vue. Les filtres peuvent être utilisés soit un à un, soit combinés et on peut filtrer une ou plusieurs des voix au choix.

Le choix de la voix filtrée et la puissance de filtrage - la résonance - se programment par le même registre S + 23 :

Soit RE la force de résonance des filtres (0-15)

```
F1 = 1   si la voix 1 doit être filtrée (sinon F1=0)
F2 = 2             2                     (sinon F2=0)
F3 = 4             3                     (sinon F3=0)
FX = 8   si l'entrée extérieure doit être filtrée.
```

```
POKE S + 23,(16*RE) + F1 + F2 + F3 + FX
```

L'entrée extérieure, c'est le signal audio que l'on peut injecter de l'extérieur dans le COMMODORE 64 par la broche 5 de la prise DIN AUDIO-VIDEO. En pratique, ceci est très peu utilisé.

La fréquence de coupure des filtres se programme sur 11 bits avec des valeurs de 0 à 2047. Lees 11 bits de fréquence sont répartis entre (S + 21) et (S + 22). (S + 22) contient les 8 bits de poids fort de la fréquence. (S + 21) contient les 3 bits de poids faible dans ses bits 0, 1 et 2.

Les effets obtenus à l'aide des filtres ne nécessitent pas le choix d'une fréquence de coupure bien précise, si bien que, usuellement, on se contentera de faire varier les 8 bits de poids forts en utilisant (S + 22) uniquement. Pour ces 8 bits, une valeur de (S + 22) de 0 ou 255 donne des fréquences de coupures de 30 Hz à 12000Hz.

Le type de filtre peut être sélectionné grâce aux bits 4, 5 et 6 du registre S + 24. On se rappellera que les bits 0 à 3 de S + 24 contiennent la valeur VL du volume sonore tandis que le bit 7 contient l'indicateur de coupure de la voix 3. Les trois types de filtres possibles sont : passe-bas, passe-haut et passe-bande.

Le SID autorise le choix de plusieurs filtres à la fois, ce qui permet de sélectionner passe-bas et passe-haut simultanément : on obtient un filtre à réjection.

Un filtre passe-bas est un filtre qui ne laisse passer que les fréquences inférieures à la fréquence de coupure. Un filtre passe-haut, c'est bien entendu l'inverse. Un filtre passe-bande ne laisse passer que les fréquences avoisinant la fréquence de coupure, tandis que le filtre à réjection laisse passer toutes les fréquences sauf celles qui avoisinent la fréquence de coupure.

Programmation de (S + 24) :

soit VL = volume sonore (0 - 15)
03 = 128 pour supprimer la sortie de la voix 3
(sinon 0)
FI = 0 pas de filtrage
FI = 16 si on désire un filtre passe-bas.
FI = 32 si on désire un filtre passe-bande.
FI = 64 si on désire un filtre passe-haut.
FI = 80 si on désire un filtre à réjection.

POKE (S + 24), 03 + FI + VL

Il faut noter que, comme les autres registres (S) à (S + 23), le registre (S + 24) peut être programmé mais jamais relu. Il faut donc bien conserver les valeurs des différents paramètres utilisés pour connaître l'état de ce registre à un moment donné.

Voir en annexe le programme "FILTRES" pour les exemples.

D'AUTRES CHOSES ENCORE...

Bien des choses merveilleuses peuvent être réalisées avec le SID. C'est en expérimentant ce domaine que l'on en tirera le meilleur parti. Des effets spéciaux tels que bruits de train, applaudissements, moteurs, etc... peuvent être obtenus avec une ou deux voix programmées en générateur de bruit et en modifiant en permanence les registres de contrôle des filtres et de la synchronisation. Une excellente source d'effets spéciaux est la modulation de la valeur d'un registre par la lecture du registre d'enveloppe 3 (S + 28). On en trouvera un exemple en annexe avec le programme "EXTRATERRESTRE"(effet haut-parleur tournant ou PHASING).

LES 2 ENTREES ANALOGIQUES

les registres (S + 25) et (S + 26) du SID permettent la lecture des valeurs "POT X" et "POT Y". Ces valeurs varient de 0 à 255 suivant que les résistances des poignées de jeu sont faibles ou fortes.

Les poignées de jeu de l'un ou l'autre connecteur "CONTROL PORT" sont sélectionnées par les bits 6 et 7 du CIA-IRQ. La lecture de ces valeurs en BASIC est très peu fiable. COMMODORE recommande l'emploi d'une petite routine en langage machine. On trouvera une version de cette routine dans le programme "POIGNEES" en annexe. Ce programme interdit les interruptions puis crée le petit délai nécessaire à une lecture stable des entrées POT X et POT Y. Après commutation vers le deuxième jeu d'entrées, le processus se répète et les résultats sont rangés en mémoire pour être lus par PEEK de façon tout à fait classique.

CHAPITRE 6

LES ENTREES-SORTIES PARALLELES

Le CBM 64 possède deux circuits 6526 aussi appelés CIA (Complex Interface Adapter) qui réalisent à eux deux toutes les fonctions d'entrées-sorties à l'exception du son et de l'image.

Un CIA est en fait un interface entre le microprocesseur et une série de fils électriques qui entrent ou sortent du système. Il peut imposer un état 0 Volt ou un état 5 Volts (0 ou 1) sur chacun de ces fils ou encore lire l'état d'un fil dont la tension est imposée par un circuit extérieur.

Chaque CIA contient de plus une circuiterie complexe constituée d'une horloge dite "temps réel", de deux compteurs et d'un registre à décalage. Les CIA peuvent également générer des interruptions pour signaler au microprocesseur qu'une tâche d'entrée-sortie urgente est requise. Les deux circuits CIA sont identiques mais ils sont connectés différemment au microprocesseur.

Le CIA no 1 que nous appelerons CIA-IRQ a sa sortie d'interruption reliée à l'entrée IRQ du 6510. Cette entrée d'interruption prévient le microprocesseur qu'une tâche l'attend sur un périphérique. Toutefois, l'interruption IRQ est masquable, c'est-à-dire qu'on peut l'inhiber par programme. C'est donc sur le CIA-IRQ que sont connectées les entrées et les sortiees les moins vitales du système. Il s'agit, en l'occurence, de la lecture sur

cassette, de la lecture du clavier et des JOYSTICKS.

L'entrée CIA, que nous nommerons CIA-NMI est, comme son nom l'indique, connecté à l'entrée NMI du microprocesseur . L'interruption NMI est non-masquable et aucun programme ne peut donc l'inhiber. On lui a consacré les tâches les plus importantes. Cette fois, c'est la connexion IEEE-série qui est visée. La sélection de banc-mémoire du VIC-II et la porte parallèle-USER'S PORT- y sont également raccordées.

LE CIA-IRQ

Il occupe les adresses $DC00 à $DC0F (56320 à 56335). Ces 16 registres ont des fonctions bien précises que nous allons décrire l'une après l'autre.

LE DECODAGE DU CLAVIER

Chaque touche du clavier est un interrupteur. A l'exception de la touche RESTORE , chaque touche met en court-circuit un fil vertical et un fil horizontal d'une matrice de 8 sur 8 fils. Les 8 fils verticaux (colonne 0 à colonne 7) sont connectés aux 8 sorties du PORT A du CIA. Les 8 fils horizontaux (rangée 0 à rangée 7) sont connectés aux 8 entrées du port B du CIA.

Les 2 ports A et B sont deux groupes de 8 fils qi peuvent être individuellement programmés en entrée ou en sortie.

Les registres $DC00 et $DC01 correspondent aux ports A et B. La direction de chacun des fils des ports A et B se programme grâce aux adresses $DC02 et $DC03 qui contiennent les indications de direction correspondantes. Le port A étant programmé "tout en sortie " pendant la durée de la lecture du clavier, nous avons donc le registre $DC02 qui vaut $FF, tandis que le port B "tout en entrée" nécessite la valeur $00 en $DC03. La lecture du clavier s'effectue en modifiant les valeurs du port A en $DC00 et en relisant ensuite le port B en $DC01.

Si on écrit la valeur 1 sur un seul des fils du port A, la lecture du port B va rendre une valeur comprise entre 0 et 255 qui nous indiquera quels fils ont pris la valeur 1. Par exemple, la lecture du nombre 65 nous informe que les bits 1 et 6 ont la

valeur 1. On en déduit aisément que les deux interrupteurs situés à l'intersection du fil "colonne 1" et des fils "ligne 1" et "ligne 6"sont fermés.

Heureusement pour nous, une routine de la ROM du 64 se charge du fastidieux travail de lecture du clavier et de la conversion de cette lecture en un code caractère utilisable. Il s'agit de la routine SCNKEY en $FF9F. Cette routine s'exécute 60 fois par seconde et son résultat est lisible en BASIC à l'aide des instructions GET et INPUT. Si toutefois il est désirable de tester le clavier pour, par exemple, tester l'enfoncement de certaines touches bien précises, on peut interdire les instructions IRQ par:

```
POKE 56333, 127
```

et les rétablir par:

```
POKE 56333, 255
```

On peut lire le clavier par une routine spéciale qui écrit une valeur dans le port A et relit l'état du port B par :

```
POKE 56320, RANGEE
```

et

```
PRINT PEEK (56321)
```

Il faut bien entendu rétablir les interruptions avant tout usage de GET ou INPUT et même pour pouvoir utiliser la touche STOP. STOP + RESTORE fonctionne toujours heureusement dans ce cas.

On notera que le tampon de clavier peut contenir 10 caractères. Mais c'est modifiable. On ramène cette valeur à 1 par:

```
POKE 649,1
```

et même à 0 par :

```
POKE 649,0
```

ce qui interdit toute frappe au clavier! Rétablir la fonction ensuite par :

```
POKE 649,10
```

n'est cependant pas facile!...

CIA-IRQ

$DC00	sélection poignees			J O Y S T I C K 0 bouton dr.	gau.	bas	haut	56320
			sélection rangée clavier					
$DC01				J O Y S T I C K 1 bouton dr.	gau.	bas	haut	56321
			lecture colonne clavier					
$DC02	direction pour port $DC00 -en sortie- =$FF							56322
$DC03	direction pour port $DC01 -en entrée- =$00							56323
$DC04						partie basse		56324
			MINUTERIE A					
$DC05						partie haute		56325
$DC06						partie basse		56326
			MINUTERIE B					
$DC07						partie haute		56327
$DC08		horloge		1/10 èmes de	secondes			56328
$DC09		horloge			secondes			56329
$DC0A		horloge			minutes			56330
$DC0B	AM/PM	horloge			heures			56331
$DC0C		REGISTRE	A		DECALAGE			56332
$DC0D			contrôle d'interruptions					56333
	IRQ		cass.	RS232	horlo	min.a	min.b	
$DC0E	50/60 hertz		contrôle de la minuterie A					56334
$DC0F	horl/ rev.		contrôle de la minuterie B					56335

A noter également : l'adresse 197 contient la valeur non décodée du balayage clavier.

Pas de touche enfoncée : PEEK(197) = 64

RETURN enfoncé : PEEK(197) = 1

ESPACE enfoncé : PEEK(197) = 60

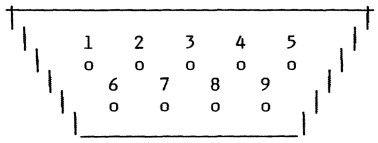
CIA-NMI

$DD00	bus série IEEE entrées DATA	CLOCK	sorties DATA	CLOCK	sortie ATN	RS232 sort. DATA	sélection banc-mémoire du VIC-II		56376
$DD01	port utilisateur								56377
$DD02	direction pour port $DD00 -en sortie- =$FF								56378
$DD03	direction pour port $DD01 -en entrée- =$00								56379
$DD04	MINUTERIE A partie basse								56380
$DD05	MINUTERIE A partie haute								56381
$DD06	MINUTERIE B partie basse								56382
$DD07	MINUTERIE B partie haute								56383
$DD08	HORLOGE 1/10èmes de secondes								56384
$DD09	HORLOGE secondes								56385
$DD0A	HORLOGE minutes								56386
$DD0B	AM/PM	HORLOGE heures							56387
$DD0C	REGISTRE A DECALAGE								56388
$DD0D	NMI	contrôle d'interruptions							56389
$DD0E	50/60 hertz	RS232	décal	contrôle de la minuterie A			min.A	min.B	56390
$DD0F	horl/ reve.	contrôle de la minuterie B							56391

Les ports A et B ont également une autre fonction : la lecture des JOYSTICKS.

Chaque JOYSTICK est constitué de 5 interrupteurs qui imposent une valeur 0 ou 1 sur certains fils d'entrée du CIA IRQ.

Voir le tableau des connexions à la page suivante.



Fonction	no de broche	bit d'entrée JOY A (2)	JOY B (1)
BAS	1	PA 0	PB 0
HAUT	2	PA 1	PB 1
DROITE	3	PA 2	PB 2
GAUCHE	4	PA 3	PB 3
POT Y	5		
BOUTON	6	PA 4	PB 4
+5Volt	7		
Masse	8		
POT X	9		

Le port A étant normalement en entrée, il suffit de lire l'état des bits 0 à 4 de l'adresse 56320 par :

JOYSTICK A (CONTROL PORT 2)

```
PRINT PEEK (56320) AND 31
```

La lecture au repos est de 31. Chaque fois qu'un interrupteur du JOYSTICK est fermé lors de la lecture, on obtient une valeur moindre car le bit correspondant vaut 0 et non plus 1.

Par exemple, on peut tester le bouton de tir par :

```
PRINT PEEK (56320)) AND 16
```

Nous avons vu que le décodage du clavier programmait le port A en sortie. Or nous l'utiliserons en entrée. Il n'y a en fait pas de contradiction car le clavier est lu par une routine d'interruption qui, lorsqu'elle n'est pas active, remet la direction du port A en entrée.

De même, on testera l'autre JOYSTICK :

JOYSTICK B (CONTROL PORT 1)

```
PRINT PEEK(56321) AND 31
```

LA SELECTION DES ENTREES ANALOGIQUES

Nous avons vu que les adresses S + 24 et S + 25 du SID permettent de lire deux valeurs analogiques. Or, nous pouvons connecter deux poignées de jeu à chaque port de contrôle ou encore une tablette graphique KOALA, une SOURIS comme la COLOR COMPUTER MOUSE de TANDY - en modifiant sa connexion - dans chaque port. Il y a donc quatre entrées analogiques sur le CBM 64. Le secret de tout ceci s'appelle "multiplexage". Le fait de mettre à 1 l'un ou l'autre des bits 6 et 7 du port A sélectionne l'une ou l'autre paire d'entrées analogiques, soit POT AX et POT AY, soit POT BX et POT BY, pour les connecter aux entrées analogiques X et Y du SID.

C'est à cause du balayage de clavier qui modifie ces deux bits soixante fois par seconde que le BASIC est incapable de lire correctement ces valeurs. Il faut donc utiliser une petite routine en langage machine qui suprime les interruptions le temps d'une lecture. le programme "POIGNEES" en annexe réalise ceci. Cette méthode est d'autre part utilisée dans le programme 'SVP' qui clôture cet ouvrage.

LES MINUTERIES DU CIA-IRQ

Les minuteries A et B du CIA sont programmables chacune sur 16 bits. Leurs fonctions sont fort complexes et le cadre de ce livre est trop étroit pour les y expliquer toutes. Sachons toutefois que plusieurs modes de comptage sont possibles: comptage continu ou unique, comptage de l'horloge du 6510 ou d'une entrée horloge extérieure: le fil CNT.

Le compteur B peut de plus compter les impulsions de sortie du compteur A, ce qui permet de chaîner les 2 compteurs et de réaliser ainsi des comptages sur 4 octets. Chaque compteur peut être à tout moment relu.

```
MINUTERIE 1 = PEEK(56324) + 256*PEEK(56325)

MINUTERIE 2 = PEEK(56325) + 256*PEEK(56326)
```

La troisième minuterie est une véritable horloge. les quatre registres qui la composent contiennent une série de compteurs qui totalisent heures, minutes, secondes et dixièmes de secondes. Le comptage de l'horloge est séquencé par les impulsions du secteur 220 Volts qui, on le sait, possède une fréquence très stable de 50 Hz. Cette horloge interne est programmable en mode 60 Hertz ou

50 Hertz. Par défaut, à l'allumage de la machine, la programmation est sur 60 Hertz. Pour passer à 50 Hertz, la fréquence en usage en Europe, on utilise ceci :

```
50 Hertz :    POKE 56334, PEEK(56334) OR 128

60 Hertz :    POKE 56334, PEEK(56334) AND 127
```

Les registres sont programmables par POKE et relisibles par PEEK sans problème. De plus, l'horloge possède la propriété de générer une interruption pour une valeur pré-programmée des registres "Heures" et "Minutes". C'est le bit 7 du registre 56335 qui détermine si les écritures dans les registres sont destinées à l'horloge ou à la programmation du réveil (bit 7 de 56335=1 pour réveil, =0 pour horloge).

```
Heures  : PRINT PEEK (56331) AND 127
Minutes : PRINT PEEK (56330)
Secondes: PRINT PEEK (56329)
Dixièmes: PRINT PEEK (56328)
AM/PM   : PRINT PEEK (56331) AND 128
```

Le compteur d'heures possède, comme on peut le voir ci-dessus, un indicateur avant-midi/après-midi dans son bit 7. Une programmation classique de ces registres est donnée en annexe avec le programme "REVEIL-MATIN".

LE REGISTRE A DECALAGE ET LES CONTROLES D'INTERRUPTION

Le registre à décalage (56332) n'est pas utilisé dans le 64. le contrôle des interruptions indique la ou les sources internes d'interruptions qui sont autorisées. Ce contrôle est réalisé par les bits 0 à 4 du registre 56333.

56333 ($DC0D)	bit 0	minuterie A
	bit 1	minuterie B
	bit 2	réveil-matin
	bit 3	registre à décalage
	bit 4	lecture cassette.

La façon de programmer ce registre est à tout le moins spéciale. Nous ne programmerons que le bit 2 qui est le seul à présenter un intérêt pratique pour le déclenchement de l'horloge.

La gestion des bits 0, 1 et 4 est réservée au KERNAL pour les accès à la cassette et à la minuterie par 1/60me de seconde qui provoque le balayage du clavier.

Pour mettre à 1 un des bits 0 à 4 - ce qui autorise la source d'interruption correspondante - il faut mettre dans le registre la valeur 128+2^(bit) :

```
POKE 56333, 128+2^(bit)
```

où (bit) est une valeur entre 0 et 4, bien entendu. Pour mettre à 0 un de ces bits et donc interdire l'interruption correspondante, il faut mettre dans le registre la valeur 2^(bit) :

```
POKE 56333,2^(bit)
```

où comme ci-dessus (bit) est un chiffre entre 0 et 4. Cette méthode non usuelle permet de modifer un bit sans se soucier des autres et sans les perturber.

La lecture du registre d'interruption par PEEK ou LDA rend une valeur où les bits 0 à 4 qui valent 1 indiquent qu'ils ont générés une interruption. Ceci a lieu pour que le processeur puisse tester quelle est la source d'une interruption. Cette opération de lecture est destructive: elle remet à 0 tous les bits d'avertissement d'interruption. La valeur lue doit donc être utilisée faute de quoi l'information est perdue. La programmation des interruptions n'a pas, elle, été modifiée lors de cette lecture.

LE CONTROLE GENERAL DES FONCTIONS

Les adresses 56334 et 56335 contiennent 16 bits qui définissent les types de comptage et les modes de fonctionnement des minuteries. Retenons seulement que :

```
56334 ($DCOE) bit 7      : 1-----> 50 Hz
                           0-----> 60 Hz

56335 ($DCOF) bit 7      : 1-----> programmer le réveil
                           0-----> programmer l'horloge.
```

Ce bref survol des fonctions du CIA-IRQ pourrait sembler insuffisant à certains. Si vous voulez utiliser et programmer les minuteries et le registre à décalage, procurez-vous chez votre revendeur COMMODORE la documentation technique du 6526 : c'est assez rébarbatif mais très complet. Une explication un peu (mais pas très) plus claire de ce circuit est donnée dans le livre "Programmer's reference guide."de COMMODORE.

LE CIA-NMI

Le second CIA occupe les adresses $DD00 à $DD0F et ne génère que des interruptions NMI. Il y a donc deux sources d'interruptions NMI dans le CBM-64: le CIA-NMI et la touche RESTORE. La fonction STOP-RESTORE est obtenue grâce à un test de la touche STOP lors de la routine de gestion de l'interruption NMI. Ce test peut être interdit (voir le programme "PROTECTEUR"). Cependant, la routine NMI,elle, ne peut jamais être interdite. Or elle dure un certain temps. Veillez donc à ne jamais pousser sur la touche RESTORE seule pendant que le 64 réalise des fonctions dont les tâches doivent avoir des durées très précises, telles que les accès à la cassette ou au bus série IEEE. les résultats peuvent être catastrophiques.

Les fonctions principales du CIA-NMI sont la gestion du bus IEEE série, celle de l'RS232, le contrôle du banc-mémoire pour le VIC-II et le port utilisateur disponible sur le connecteur arrière droit du 64. La gestion de l'RS-232 et de l'IEEE série sont très complexes. Nous n'analyserons pas leur fonctionnement ici, car tout le nécessaire à ce sujet est fait (et bien fait) par le KERNAL dont nous utiliserons ici dès lors les routines.

Par contre, les adresses 56577 (port B) et 56579 (direction du port B) sont d'un grand intérêt pour nous. Il s'agit du port utilisateur (USER'S PORT). La programmation en est identique à celle du port B du CIA-IRQ, nous n'y reviendrons donc pas. Rappellons que chaque bit à 1 dans le registre $DD03 (56579) indique que le bit correspondant du port utilisateur en $DD01 (56577) fonctionne en sortie. Et pour 0, ce sera une entrée.

Le programme en annexe "LISTING" est un exemple de la souplesse du système 64. Les vecteurs de sortie de caractères, d'OPEN et de CLOSE y sont interceptés pour aiguiller la sortie imprimante (OPEN 4,4) vers la porte parallèle pour y commander une imprimante classique avec le protocole "CENTRONICS". La grande majorité des imprimantes respectent ce standard.

Vérifiez toutefois que la résistance interne de toute charge appliquée au port utilisateur n'est pas inférrieure à 2700 Ohms, valeur minimale sous laquelle la sécurité du 6526 n'est plus garantie.

Cette résistance se mesure entre les broches "signaux" de l'imprimante (ou de tout autre dispositif) et la broche de masse. Certaines imprimantes ont une résistance interne de 1000 ohms, ce qui peut créer des problèmes. Dans les imprimantes sans problème, citons entre bien d'autres : EPSON, SEIKOSHA, OKI...

Le port A du CIA-NMI est consacré aux signaux du bus série IEEE, à la sortie RS-232 et à la sélection de banc-mémoire du VIC-II. (bits 0 et 1, adresse 56576 ou $DD00). Ce point est détaillé au chapitre 3.

Le port B peut être utilisé par le KERNAL pour le contrôle de l'RS-232. Celui-ci ne peut toutefois pas être utilisé sans un interface d'adaptation des tensions. COMMODORE vend un tel dispositif. Un autre a, par ailleurs, été décrit dans "Le livre du VIC" édité chez BCM.

Les adresses 56584 ($DD08) à 56587 ($DD0B) sont celles d'une seconde horloge, identique à celle du CIA-IRQ et dont la programmation est également autorisée. La fréquence de 50 ou 60 Hertz se programmera avec le bit 7 de l'adresse 56590 ($DD0E) et la programmation du réveil ou de l'horloge pourra se sélectionner en 56591 ($DD0F, bit 7).

LA PORTE PARALLELE DU 6510

Nous avons vu au chapitre 1 les possibilités de changement de banc-mémoire à l'aide de la porte parallèle du 6510. Cette porte possède 6 bits utiles (0 à 5) et nous n'avons encore étudié que les bits 0, 1 et 2.

Les bits 3, 4 et 5 de l'adresse 1 sont utilisés par le KERNAL pour la gestion de la cassette.

$0001	bit 3	écriture cassette (sortie)
	bit 4	interrupteurs cassette (entrée)
	bit 5	moteur cassette (sortie)

La lecture du bit 4 permet de voir si une touche du lecteur de cassette C2-N est enfoncée. Le bit 5, lui, contrôle le fonctionnement du moteur de la cassette. Ces deux bits sont cependant difficiles à manier car la routine d'interruptions du balayage clavier ré-écrit régulièrement à cette adresse. Voici donc la méthode pour arrêter le moteur par programme :

```
10   A=PEEK (1) AND 16 : B=PEEK (1) OR 32
20   POKE 192,B : POKE 1,B
30   IF A GOTO 50
40   PRINT "POUSSEZ STOP SUR CASSETTE"
50   IF NOT (PEEK (1) AND 16) GOTO 50
```

Nous allons maintenant passer en revue quelques aspects des périphériques usuels du CBM 64 habituellement raccordés au CIA-NMI: les périphériques IEEE et le lecteur de cassette.

LES PERIPHERIQUES SUR L'IEEE SERIE

LE LECTEUR DE DISQUETTES 1541

Le lecteur de disquettes 1541 est un périphérique très utile. Son fonctionnement interne est assez complexe pour faire le sujet d'un livre entier. Nous ne ferons donc qu'effleurer le sujet en donnant quelques trucs et astuces permettant d'en tirer un meilleur parti.

LE REPERTOIRE DU DISQUE

Les programmes DIR et SD permettent tous deux la lecture du répertoire du disque et le choix de l'un ou l'autre programme qui sera ensuite chargé en mémoire. L'un est plus complet mais plus lent, l'autre est plus rapide. En regardant la structure du programme DIR, on peut voir un exemple de la méthode d'accès à un secteur du disque particulier. Dans ce cas précis, il s'agit du répertoire du disque.

Le programme REORG, un peu plus complexe, effectue lui aussi la lecture du répertoire. Mais de plus, il trie en mémoire (par le tri bulle ou BUBBLE-SORT) la liste des fichiers du disque et ré-écrit cette liste triée sur le disque. Dès lors, toutes les lectures subséquentes du répertoire de ce disque le listeront par ordre alphabétique. De plus, si le répertoire contenait des trous (emplacements laissés vides par un fichier effacé), ceux-ci sont rejetés en fin de répertoire. Toute nouvelle écriture sur le disque se rangera donc en fin de répertoire.

La gestion des fichiers de données sur le disque est aussi complexe que le reste du système d'exploitation.

Entre autres, il faut savoir que les programmes ne sont que des fichiers comme les autres et que l'on peut les manipuler ainsi sans problème. Par exemple, le programme 'DATA-SUR-DISQUE' montre comment lire un programme octet par octet et recréer un nouveau programme à partir de ces données.

Un autre exemple de ce genre de manipulation se trouve dans le 'APPENDISK' qui lit un programme BASIC séquentiellement sur le disque pour en trouver la fin, lit ensuite un second programme et parallèlement, réécrit les deux à la suite l'un de l'autre dans un troisième fichier de type 'programme'. En ce qui concerne les fichiers programmes rangés sur disque, il n'est pas toujours aisé

de les recopier. Leur longueur est connue mais leur adresse de départ non. Il suffit pourtant d'aller la lire sur le disque au début du fichier.

Le petit programme 'ADRESSE DISQUE' nous donne ainsi cette adresse de départ. La recopie de ce genre de fichier peut se faire avec le moniteur MON en suivant la procédure suivante :

1. Lire l'adresse de départ du programme avec 'ADRESSE DISQUE'.

2. convertir cette adresse en hexa.

3. calculer l'adresse de fin en ajoutant à l'adresse de début la longueur du programme qui est égale à 256 fois le nombre de blocs signalé au répertoire du disque.

4. convertir cette adresse en hexa.

5. charger et lancer le moniteur MON et en sortir en tapant 'X'.

6. charger le programme en mémoire avec un LOAD"PROG",8,1.

7. entrer en moniteur avec SYS 8, enlever le disque et le remplacer par le disque destination.

8. taper la commande :

   S"PROG",08,XXXX,YYYY

   où XXXX est l'adresse de début et YYYY l'adresse de fin + 1 du programme (en hexa).

C'est tout !

Une autre solution existe pour charger plus commodément les programmes en langage machine qui ne se chargent pas à l'adresse du BASIC. En utilisant le programme 'SAUVEUR' en annexe, on transforme un programme 'mémoire haute' en un programme ordinaire qui se charge en bas de mémoire. Le SAUVEUR le remonte ensuite à sa place définitive même si celle-ci est située derrière les ROM BASIC ou KERNAL. Voir le mode d'emploi détaillé de 'SAUVEUR'.

Nous n'en dirons pas plus sur ce vaste sujet.

LES IMPRIMANTES 1526 ET MPS-801/802

Elles ont le gros avantage de reproduire les caractères graphiques COMMODORE et de ne nécessiter aucune programmation spéciale. Cependant, la qualité et la vitesse d'impression sont relativement faibles. C'est pourquoi ce livre détaille la façon d'interfacer une imprimante parallèle quelconque comme les DIABLO

à marguerite (qualité courrier) ou les très connues EPSON .

Toutefois, certaines limitations des imprimantes MPS 801 et 1526 peuvent être outrepassées en les poussant dans leurs derniers retranchements.

Ainsi en utilisant le mode graphique au lieu du mode texte, il est possible d'imprimer du texte sur 120 colonnes de largeur au lieu de 80. Il suffit pour cela de redessiner les caractères dans une matrice de points plus étroite et d'envoyer le dessin de ces caractères à l'imprimante en mode graphique. Ceci a été réalisé dans le programme "IMPRIME-120" en annexe. Une des fonctions les plus demandées est la recopie d'un écran sur l'imprimante. En mode texte, il n'y a guère de problèmes : le petit programme 'RECOPIE' effectue ceci sans problème en utilisant l'écran comme un périphérique d'entrée et l'imprimante comme périphérique de sortie, (Eh oui, même l'écran peut être considéré comme une entrée du 64).

Une procédure similaire est possible pour d'autres modes graphiques. Un écran en mode haute résolution peut ainsi être recopié sur une imprimante COMMODORE avec le programme 'IMPRIME-HR' qui tourne l'image de 90 degré vers la gauche. Il s'agit d'un impératif technique. En effet, le balayage du générateur de graphismes du 64 est horizontal tandis que celui de l'imprimante est vertical. La reproduction d'écran est donc bien plus simple en écrivant "en travers", Ne vous attendez toutefois pas à une vitesse d'impression extraordinaire. Le BASIC de IMPRIME-HR a besoin d'un bon quart d'heure pour venir à bout des 64000 points qu'il doit calculer et imprimer.

LE LECTEUR DE CASSETTES C2-N

Ce périphérique est le premier à avoir été mis en service par COMMODORE sur les premiers PET et depuis sa fiabilité ne s'est pas démentie. Ses commandes principales sont, comme vous le savez sûrement, :

```
SAVE"PROG"

LOAD"PROG"

VERIFY"PROG"
```

qui n'appellent guère de commentaires.

Mais certains usages méconnus de la C2-N sont parfois bien utiles:

Si on sauve un programme en langage machine par SAVE"PROG",1,1 lors de chaque LOAD de ce programme, la zone chargée aura tooujours, quoique l'on fasse le même emplacement en mémoire que lors du SAVE. Si par contre la zone a été sauvée par

un simple SAVE"PROG",1 ou SAVE"PROG", la zone ne sera rechargée à son adresse de départ que par LOAD"PROG",1,1.

Ce rechargement à une adresse précise est possible parce que le bloc d'en-tête du programme contient, outre le nom du programme, son adresse de chargement.

En fait, le bloc d'en-tête contient :

1 octet : adresse secondaire utilisée lors du SAVE.
2 octets: adresse de début
2 octets: adresse de fin
n octets: le nom du programme.

n, le nombre d'octets du nom est limité à 16 par l'interpréteur BASIC lors de l'analyse du mot-clé SAVE. Par contre, en langage machine, la longueur du nom peut atteindre 254 caractères. Cette particularité est utilisée dans certains programmes qui chargent uniquement un nom de programme qui est en fait le programme lui-même. L'avantage est de ne modifier aucun octet de la zone BASIC lors du LOAD. C'est intéressant dans certains cas, par exemple pour protéger des logiciels de la copie illicite.

A cause de la structure du VIC-II, l'écran doit être éteint lors des instructions SAVE et LOAD. La présence de lutins à l'écran pendant les LOAD et SAVE sur disquettes pose aussi quelquefois des problèmes. La solution est simple : Il suffit

d'éteindre les lutins avant le SAVE par :

```
POKE 53269,0:SAVE"PROG",8
```

De même, avant un LOAD, si des lutins sont à l'écran :

```
POKE 53269,0:LOAD "PROG",8
```

Un dernier truc concernant le lecteur de cassettes est le "MERGE" découvert il y a plusieurs années par le spécialiste du 'PET' Jim Butterfield.

Cette technique permet d'entrer des lignes de programme à partir d'un fichier spécial sauvé sur cassette. Il ne s'agit pas ici de coller un morceau de programme derrière celui qui est en mémoire, mais bien d'effectuer le même travail que si les lignes étaient tapées au clavier.

* Tout d'abord, il faut préparer la bande avec les lignes qui seront incluses dans le programme par l'opération de MERGE.

* Introduire ces lignes en mémoire normalement puis taper:

```
OPEN 1,1,1,"PROG2":CMD1:LIST
```

Et lorsque le curseur réapparaît, taper:

```
PRINT£1:CLOSE1:NEW
```

* Ensuite, chargeons en mémoire le programme qui doit être modifié:

```
LOAD "PROG1"
```

* Installer la bande spéciale créée ci-dessus et taper:

```
POKE19,1:OPEN1
```

* Introduire enfin la commande magique que voici :

1. effacer l'écran
2. taper 3<curseur en bas>
3. taper en mode direct :

```
PRINT CHR$(19):POKE198,1:POKE631,13:POKE153,1
```

Le 64 affiche un message d'erreur après avoir lu la cassette. C'est normal, ne vous effrayez pas! Il reste à taper :

```
CLOSE 1
```

et le tour est joué !!

C H A P I T R E 7

LA CARTE-MEMOIRE DU 64

La carte-mémoire ci-dessous ne reprend que les adresses utilisées en mémoire vive par BASIC et KERNAL pour leurs pointeurs, variables de travail,etc....

Le détail des adresses des périphériques et des mémoires mortes sont détaillés dans les chapitres qui traitent de chacun de ces sujets en particulier. Nous ne les reprendrons donc pas ici.

ADRESSES HEXA	ADRESSES DECIMAL	DESCRIPTION
$0000	0	Direction des bits du port 1 du 6510
$0001	1	Port 1 du 6510
$0002	2	Inutilisé
$0003-$0004	3- 4	Vecteur conversion flottant-entier
$0005-$0006	5- 6	Vecteur conversion entier-flottant
$0007	7	Caractère recherché
$0008	8	Drapeau:Recherche fin de mode quote
$0009	9	Colonne écran du tabulateur précédent
$000A	10	0=LOAD, 1=VERIFY
$000B	11	Pointeur tampon d'entrée/Nombre d'indices
$000C	12	Dimension par défaut pour DIM (10)

ADRESSES HEXA	ADRESSES DECIMAL	DESCRIPTION
$000D	13	Type de donnée:$FF=chaîne,$00=nombre
$000E	14	Type de donnée:$80=entier,$00=flottant
$000F	15	Drapeau:Recherche de DATA ou mode quote pendant LIST ou Ramassage d'ordure
$0010	16	Drapeau:Indice de tableau ou paramètre de fonction
$0011	17	Drapeau:$00=INPUT,$40=GET,$98=READ
$0012	18	Drapeau:Signe pour TAN ou résultat d'une comparaison
$0013	19	Drapeau:affiche '?' pendant INPUT(O/N) aussi:numéro de fichier CMD
$0014-$0015	20- 21	Valeur entière provisoire
$0016	22	Pointeur vers la pile de chaînes provisoires
$0017-$0018	23- 24	Adresse de la dernière chaîne provisoire
$0019-$0021	25- 33	Pile des chaînes provisoires
$0022-$0025	34- 37	Pointeurs provisoires
$0026-$002A	38- 42	Stockage provisoire d'un nombre flottant
$002B-$002C	43- 44	POINTEUR DE DEBUT DE BASIC
$002D-$002E	45- 46	POINTEUR DE FIN DE PROGRAMME/DEBUT DE VARIABLES
$002F-$0030	47- 48	Pointeur de début de zone tableaux
$0031-$0032	49- 50	Pointeur de fin de tableaux +1
$0033-$0034	51- 52	Pointeur de bas de zone chaînes
$0035-$0036	53- 54	Pointeur vers partie de la zone chaînes (pour ramassage d'ordures)
$0037-$0038	55- 56	POINTEUR DE SOMMET DE MEMOIRE POUR BASIC
$0039-$003A	57- 58	Numéro de la ligne BASIC en cours
$003B-$003C	59- 60	Numéro de ligne BASIC précédente
$003D-$003E	61- 62	Pointeur d'instruction BASIC pour CONT
$003F-$0040	63- 64	Numéro de ligne DATA en cours
$0041-$0042	65- 66	Pointeur vers adresse de la DATA en cours
$0043-$0044	67- 68	Vecteur d'INPUT
$0045-$0046	69- 70	Nom de la variable BASIC en cours
$0047-$0048	71- 72	Pointeur d'adresse de la valeur de la variable en cours
$0049-$004A	73- 74	Pointeur de variable indice de boucle FOR aussi:N.de périph/N.de fichier
$004B-$004C	75- 76	Pointeur provisoire
$004D	77	Octet provisoire
$004E-$0052	78- 82	Zone de stockage temporaire pour KERNAL
$0053	83	Valeur de STEP par défaut (=1)
$0054-$0055	84- 85	Vecteur pour calcul de fonctions
$0056-$0057	86- 87	Vecteur pour calcul de fonctions
$0058-$0060	88- 96	Zone de stockage temporaire
$0061	97	Accumulateur flottant principal-exposant
$0062-$0065	98- 101	Accumulateur flottant principal-mantisse
$0066	102	Signe accumulateur flottant principal
$0067	103	Pointeur vers table de polynôme
$0068	104	Bit de garde accumulateur flottant princ.
$0069	105	Accumulateur flottant secondaire-exposant
$006A-$006D	106- 109	Accumulateur flottant secondaire-mantisse

ADRESSES HEXA	ADRESSES DECIMAL	DESCRIPTION
$006E	110	Signe accumulateur flottant secondaire
$006F	111	Produit des signes des accu.flottants
$0070	112	Bit de garde accumulateur flottant sec.
$0071-$0072	113- 114	Pointeur vers tampon cassette
$0073-$008A	115- 138	CHRGOT--routine de prise de caractère
$0079	121	Point d'entrée de CHRGOT sans incrémentation du pointeur de caractères.
$007A-$007B	122- 123	Pointeur du caractère courant dans le texte BASIC
$008B-$008F	139- 143	Valeur de calcul pour RND
$0090	144	OCTET ST-STATUS :état des périphériques
$0091	145	Resultat du balayage clavier-rangée qui contient STOP/SHIFT/CTRL
$0092	146	Correction de vitesse pour cassette
$0093	147	Drapeau:$00=LOAD,$01=VERIFY
$0094	148	Drapeau:octet retenu en mémoire par la routine de sortie IEEE série (0/N)
$0095	149	Octet retenu par routine de sortie IEEE
$0096	150	Drapeau:Synchronisation obtenue sur cassette (0/N)
$0097	151	Sauvegarde de registre par le KERNAL
$0098	152	Nombre de fichiers ouverts
$0099	153	Périphérique d'entrée par défaut (0=clav)
$009A	154	Périph. de sortie par défaut (3=écran)
$009B	155	Parité du caractère lu sur cassette
$009C	156	Drapeau:octet reçu sur cassette (0/N)
$009D	157	Drapeau:$00=mode progr.,$80=mode direct
$009E	158	Compteur d'erreurs cassette 1ère passe
$009F	159	Compteur d'erreurs cassette 2èmè passe
$00A0-$00A2	160- 162	Horloge TI sur 3 octets
$00A3	163	Compteur de bits reçus sur IEEE série
$00A4	164	Octet en cours de réception IEEE série
$00A5	165	Compteur d'impulsions dans les blocs d'en-tête sur cassette.
$00A6	166	Pointeur d'octet dans le tampon cassette
$00A7	167	Bits lus sur RS-232
$00A8	168	Compteur de bits lus sur RS-232
$00A9	169	Drapeau:Bit START sur RS-232 (0/N)
$00AA	170	Octet lu sur RS-232
$00AB	171	Parité de l'octet lu sur RS-232
$00AC-$00AD	172- 173	Pointeur pour décalage d'écran
$00AE-$00AF	174- 175	Adresse de fin de chargement cassette
$00B0	176	Correction de vitesse pour cassette
$00B1	177	Octet prov. pour calcul vitesse cassette
$00B2-$00B3	178- 179	Pointeur de début de tampon cassette
$00B4	180	Compteur de bits envoyés sur RS-232
$00B5	181	Bit à envoyer sur RS-232
$00B6	182	Octet à envoyer sur RS-232
$00B7	183	Longueur du nom de fichier en cours
$00B8	184	Numéro du fichier logique en cours
$00B9	185	Adresse secondaire du fichier en cours
$00BA	186	Numéro de périph. du fichier en cours

ADRESSES HEXA	DECIMAL	DESCRIPTION
$00BB-$00BC	187- 188	Pointeur vers le nom du fichier en cours
$00BD	189	Octet en cours d'écriture sur cassette.Aussi:parité émission RS-232
$00BE	190	Compteur de phases pour écriture cassette
$00BF	191	Octet en cours de lecture sur cassette
$00C0	192	Drapeau:moteur cassette en marche
$00C1-$00C2	193- 194	Adresse de la zone périphériques
$00C3-$00C4	195- 196	Pointeur utilitaire pour le KERNAL
$00C5	197	Dernière touche enfoncée sans décodage
$00C6	198	Nombre de caractères dans tampon clavier
$00C7	199	Drapeau:$01=écriture inversée,$00=normal
$00C8	200	Pointeur de fin de ligne pour INPUT
$00C9-$00CA	201- 202	Position X,Y du curseur avant INPUT
$00CB	203	Drapeau:écriture avec SHIFT
$00CC	204	Drapeau:$00=curseur clignotant
$00CD	205	Compteur de clignotement curseur
$00CE	206	Caractère sous le curseur
$00CF	207	Drapeau:curseur inversé/normal
$00D0	208	Drapeau:INPUT ou GET
$00D1-$00D2	209- 210	Pointeur de ligne-écran en cours
$00D3	211	Position du curseur dans la ligne
$00D4	212	Drapeau:$00=Mode quote
$00D5	213	Longueur ligne-écran en cours (40-80)
$00D6	214	Numéro de ligne de 40 c. sous le curseur
$00D7	215	Somme de vérification pour cassette
$00D8	216	Nombre de caract.'INSERT' à remplir
$00D9-$00F0	217-240	Table des poids forts des adresses de lignes de l'écran
$00F1-$00F2	241- 242	Pointeur fin de dernière ligne-écran+1
$00F3-$00F4	243- 244	Pointeur vers curseur dans la RAM couleur
$00F5-$00F6	245- 246	Vecteur de décodage clavier
$00F7-$00F8	247- 248	Pointeur tampon d'entrée RS-232
$00F9-$00FA	249- 250	Pointeur tampon de sortie RS-232
$00FB-$00FE	251- 254	ESPACE LIBRE EN PAGE ZERO
$00FF	255	Octet provisoire pour BASIC
$0100-$010A	256- 266	Zone de conversion flottant-chaîne
$0100-$013E	256- 318	Table des erreurs en cours de récupération en lecture cassette
$0100-$01FF	256- 511	PILE DU MICROPROCESSEUR 6510
$0200-$0258	512- 600	Tampon d'entrée du KERNAL (pour CHRIN)
$0259-$0262	601- 610	Table des fichiers logiques ouverts
$0263-$026C	611- 620	Table des n. de périph. des fichiers
$026D-$0276	621- 630	Table des adresses sec. des fichiers
$0277-$0280	631- 640	TAMPON DE CLAVIER (10 caractères)
$0281-$0282	641- 642	Pointeur de bas de mémoire pour KERNAL
$0283-$0284	643- 644	Pointeur haut de mémoire pour KERNAL (sert à définir l'adresse des tampons pour RS-232)
$0285	645	Délai d'attente KERNAL pour IEEEparal.
$0286	646	Couleur de tracé en cours

ADRESSES HEXA	ADRESSES DECIMAL	DESCRIPTION
$0287	647	Couleur de fond sous le curseur
$0288	648	Numéro de page-écran (<=127) pour BASIC
$0289	649	Taille du tampon clavier
$028A	650	Drapeau:$80=Touches à répétition
$028B	651	Vitesse de répétition des touches
$028C	652	Délai avant lère répétition
$028D	653	Type de table décodage clavier en cours
$028E	654	Dernier enfoncement touche SHIFT
$028F-$0290	655- 656	Vecteur de décodage clavier
$0291	657	Drapeau:$00=pas de SHIFT,$80=SHIFT autor
$0292	658	Drapeau:$00=décalage écran vers le bas autorisé
$0293	659	RS-232 REGISTRE DE CONTROLE
$0294	660	RS-232 REGISTRE DE COMMANDE
$0295-$0296	661- 662	Vitesse non-standard pour RS-232
$0297	663	RS-232 REGISTRE D'ETAT
$0298	664	RS-232 Nombre de bits restant à envoyer
$0299-$029A	665- 666	Durée d'un bit sur l'RS-232 en microsec.
$029B	667	RS-232: pointeur fin du tampon d'entrée
$029C	668	RS-232: Numéro de page du tampon d'entrée
$029D	669	RS-232:Numéro de page tampon de sortie
$029E	670	RS-232: pointeur fin de tampon de sortie
$029F-$02A0	671-672	Sauvegarde du vecteur d'interruption pendant les opérations cassette
$02A1	673	Registre interne pour l'RS-232
$02A2	674	Sauvegarde du registre d'interruption du CIA-NMI pendant opérations cassette
$02A3	675	Stockage temporaire en lecture cassette
$02A4	676	Sauvegarde du registre de controle du CIA-NMI pendant opérations cassette
$02A5	677	Nombre de caractères entre curseur et début de ligne de 40 car. suivante
$02A6	678	TYPE DE CBM64:$00=USA,$01=EUROPE
$02A7-$02FF	679- 767	ZONE LIBRE
$0300-$0301	768- 769	Vecteur impression mess.erreurs BASIC
$0302-$0303	770- 771	Vecteur redémarrage du BASIC
$0304-$0305	772- 773	Vecteur compactage de ligne BASIC
$0306-$0307	774- 775	Vecteur de LIST pour mot-clé compacté
$0308-$0309	776- 777	Vecteur d'exécution de mot-clé
$030A-$030B	778- 779	Vecteur d'évaluation d'expression
$030C	780	Sauvegarde de l'accumulateur pour SYS
$030D	781	Sauvegarde de X pour SYS
$030E	782	Sauvegarde de Y pour SYS
$030F	783	Sauvegarde des drapeaux du 6510 pour SYS
$0310-$0312	784- 786	Saut(JMP) vers routine USR
$0313	787	Inutilisé
$0314-$0315	788- 789	Vecteur routine d'interruption IRQ
$0316-$0317	790- 791	Vecteur routine de BREAK (BRK)
$0318-$0319	792- 793	Vecteur routine d'interruption NMI
$031A-$031B	794- 795	Vecteur d' OPEN
$031C-$031D	796- 797	Vecteur de CLOSE

ADRESSES HEXA	DECIMAL	DESCRIPTION
$031E-$031F	798- 799	Vecteur de routine CHKIN
$0320-$0321	800- 801	Vecteur de routine CHKOUT
$0322-$0323	802- 803	Vecteur de routine CLRCHN
$0324-$0325	804- 805	Vecteur de routine CHRIN
$0326-$0327	806- 807	Vecteur de routine CHROUT
$0328-$0329	808- 809	Vecteur de routine STOP
$032A-$032B	810- 811	Vecteur de routine GETIN
$032C-$032D	812- 813	Vecteur de routine CLALL
$032E-$032F	814- 815	Vecteur de routine USR
$0330-$0331	816- 817	Vecteur de routine LOAD
$0332-$0333	818- 819	Vecteur de routine SAVE
$0334-$033B	820- 827	Inutilisé
$033C-$03FB	828- 1019	Tampon de cassette
$03FC-$03FF	1020- 1023	Inutilisé
$0400-$07E7	1024- 2023	Mémoire écran à l'allumage du 64
$07F8-$07FF	2024- 2047	Pointeurs de lutins à l'allumage du 64
$0800-$9FFF	2048-40959	ZONE MEMOIRE POUR BASIC: TEXTE,VARIABLES,TABLEAUX,ESPACE LIBRE, CHAINES
$8000-$9FFF	32768-40959	ZONE POUR PROGRAMMES AUTO-START (en RAM ou en cartouche ROM externe)
$A000-$BFFF	40960-49151	ROM BASIC (8K)
$C000-$CFFF	49152-53247	ZONE LIBRE (4K)
$D000-$DFFF	53248-57343	RAM COULEUR ou ENTREES/SORTIES
$E000-$FFFF	57344-65535	ROM KERNAL (8K)

Pour le détail des entrées/sorties et des ROM, consulter le chapitre correspondant:

ANNEXE 1

PROGRAMMES DEMONSTRATIFS

ACTIVITE

affiche le contenu d'une page mémoire de 256 octets, bit par bit à l'aide de lutins. Ce programme, une fois activé, reste en fonction et n'occupe AUCUN octet en mémoire. Il n'utilise ni interruptions, ni langage machine. Curieux n'est-ce pas ?

```
1 REM MONITEUR D'ACTIVITE POUR CBM 64
2 REM
8 REM PAR  B.MICHEL LE LIVRE DU CBM 64
9 REM
10 PRINT"[CLR][JAUNE]     MONITEUR D'ACTIVITE MEMOIRE"
20 PRINT"[C.BAS][C.BAS][BLANC] CE PROGRAMME,UNE FOIS ACTIVE, UTILISE"
30 PRINT"[C.BAS]4 LUTINS POUR AFFICHER LE CONTENU DES"
35 PRINT"[C.BAS]256 OCTETS D'UNE PAGE MEMOIRE A CHOISIR"
40 PRINT"[C.BAS]DANS LES 16K INFERIEURS."
50 PRINT"[C.BAS][C.BAS][C.BAS] LA PAGE 1 CONTIENT LA PILE DES ADRES-"
55 PRINT"[C.BAS]SES DE RETOUR DES SOUS-ROUTINES.
60 PRINT"[C.BAS]LA PAGE 0 ,QUI CONTIENT NOMBRE D'INFOR-"
65 PRINT"[C.BAS]MATIONS VITALES EST DONNEE EN DETAILS."
70 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS]POUSSEZ [RVS ON]RETURN[RVS OFF] PO
     UR CONTINUER"
90 POKE198,0:WAIT198,1:GETA$
100 INPUT"[CLR]NUMERO DE PAGE (0-255)   0[C.GAUCHE][C.GAUCHE][C.GAUCHE]
     ";A
120 IFA>63 GOTO 100
130 IF(A>16)AND(A<32)THENPRINT"[C.BAS]PAGE INTERDITE (GENERATEUR DE CAR
     ACTERE)":GOTO70
```

```
140 A=A*4:B=2040
150 POKE53277,15:REM EXPANSION X
160 FOR I=0TO3:POKE B+I,A+I:NEXT I
170 FOR I=0TO3:POKE53287+I,1:NEXT I:REM COULEUR
180 POKE53269,15:REM AUTORISE LUTINS 0,1,2,3
200 FOR I=0TO3:POKE53248+2*I,39:NEXT I:REM POSIITION X
220 POKE53264,15:REM MSB X 0-3
240 FOR I=0TO3:POKE53249+2*I,50+I*42:NEXT I:REM POSIT.Y
260 POKE53277,15:REM EXPANSION X
280 POKE53271,15:REM EXPANSION Y
300 FOR I=0TO3:POKE53287+I,1:NEXT I:REM COULEUR
320 IF A=0 GOTO 1000
330 PRINT"[CLR]";
335 IF A<>4THEN PRINT"[C.BAS][C.BAS]RUN[HOME]";:END
340 PRINT"CECI EST LA PILE"
345 PRINT"[C.BAS]LE SOMMET DE PILE EST"
350 PRINT"[C.BAS]EN BAS EN 512"
360 PRINT"[C.BAS][C.BAS][C.BAS]RUN[C.HAUT][C.HAUT][C.HAUT]":END
1000 PRINT"[CLR][RVS ON][JAUNE]PAGE0 ACTIVITE MEMOIRE[RVS OFF][JAUNE L.
     ]"
1010 P$="[HOME]":FORI=0TO28:P$=P$+"[C.DROITE]":NEXTI
1020 FORI=1TO23:P$=P$+"[C.BAS]":NEXTI
1030 PRINTLEFT$(P$,30);"  0"
1035 PRINTLEFT$(P$,40);"128"
1040 PRINTLEFT$(P$,50);"255"
1050 P$="[HOME]"+RIGHT$(P$,36)
1060 PRINTLEFT$(P$,15)"001:SELECTION
1070 PRINTLEFT$(P$,16)"DE BANC-MEMOIRE
1080 PRINTLEFT$(P$,26)"160:HORLOGE AU
1090 PRINTLEFT$(P$,27)"    1/60 DE SEC.
1100 PRINTLEFT$(P$,31)"205-207:GESTION
1110 PRINTLEFT$(P$,32)"DU CURSEUR
1999 PRINT"FIN"
55554 END
55555 SAVE"@0:ACTIVITE",8

READY.
```

ADRESSE-DISQUE.

donne l'adresse de chargement d'un programme sauvé sur disquette.

```
10 REM ADRESSE-DISQUE
20 :
30 C$=CHR$(0):INPUT"NOM DU PROGRAMME";F$
40 OPEN1,8,2,F$:GET#1,A$,B$:CLOSE1
50 PRINT:PRINT"ADRESSE DE DEBUT = ";
60 PRINTASC(A$+C$)+256*ASC(B$+C$):END
55555 SAVE"@0:ADRESSE-DISQUE",8

READY.
```

ANNEAU

Démonstration de la modulation en anneau du S.I.D.

```
10 REM — MODULATION ANNULAIRE —
20 :
30 PRINT"[CLR]NORMAL                    (1)"
31 PRINT"PROPORTIONNEL   AU VOLUME  (2)":PRINT
40 PRINT"MODULATION ANNULAIRE":PRINT
41 PRINT"CHOIX DE LA FREQUENCE DE LA VOIX AUX.":PRINT
42 PRINT"PROPORTIONNELLE (*2)       (3)"
43 PRINT"PROPORTIONNELLE (*257/256) (4)"
44 PRINT"FIXE            (1900 HZ)  (5)"
45 PRINT"PROPORTIONNELLE AU VOLUME  (6)"
60 INPUT "(1,2,3,4,5,6 OU 0=FIN) ";B
65 IF B=0 THEN END
70 S=54272:GOSUB900:VL=15:AR=16+4
71 IFB<3THEN AR=16
75 DE=AR+1
80 AD= 9*16+11:SR=12*16+9:POKE S+24,VL
110 RESTORE:POKE S+5,AD:POKE S+6,SR
111 POKE S+19,AD:POKE S+20,SR
120 READ N,D:IF N=0THEN GOSUB900:RUN
125 POKE S+1,N:POKES+4,DE
126 POKE S+15,N:POKES+18,17
130 ON B GOSUB 300,400,500,600,700,800
140 POKE S+4,AR
150 ON B GOSUB 300,450,450,450,450,850
160 GOTO120
200 DATA25,4,28,4,25,4,25,4,25,2,28,2
210 DATA32,12,25,4,28,4,19,4,19,2,19,2
230 DATA21,1,24,1,25,4,24,2,19,4,0,0
299 :
300 FOR TD=0TO50*D:NEXT:RETURN:REM NORMAL
399 :
400 REM             NORM. PROP AU VOLUME
405 POKE S+15,N:POKES+18,17
410 FOR TD=0TO8*D
415 POKE S,PEEK(S+28)/4:NEXT:RETURN
420 :
450 REM                 FIN PROP AU VOLUME
455 POKES+18,16:GOTO410
499 :
500 REM                       PROP. *2
510 POKE S+15,N*2:GOTO300
550 REM                 FIN PROPORTIONNEL
555 POKES+18,AR:GOTO300
599 :
600 REM                 PROP. *257/256
610 POKE S+14,N:GOTO300
699 :
700 REM                 FIXE
710 POKE S+15,128:GOTO300
799 :
800 REM                 PROP AU VOLUME
810 POKES+18,17
820 FOR TD=0TO5*D
830 POKE S+14,PEEK(S+28)/4
840 NEXT:RETURN
850 REM                 FIN PROP AU VOLUME
860 POKES+18,16:GOTO 820
899 :
900 FOR T=S TO S+24:POKE T,0:NEXT:RETURN
55555 SAVE"@0:ANNEAU",8

READY.
```

APPENDISK

Concatène deux programmes sauvés sur disque bout à bout. Attention, la numérotation des programmes doit être prévue en conséquence.

Répondre successivement aux questions "premier programme?", "deuxième programme?" et "programme destination". Attention, si ce dernier existe déjà, il sera écrasé par le nouveau programme créé par APPENDISK.

L'opération peut être exécutée plusieurs fois.

```
10 PRINT"[CLR]CONCATENATION DE PROGRAMMES"
150 PRINT"[C.BAS]NOM 1 ER PROGRAMME        ";:INPUTA$:Z$=A$
160 PRINT"[C.BAS]NOM 2 EME  PROGRAMME      ";:INPUTB$
170 OPEN15,8,15:PRINT#15,"I1":OPEN5,8,5,A$+",P,R"
190 GOSUB370
210 PRINT"[C.BAS]NOM PROGRAMME DESTINATION ";:INPUTC$
220 OPEN6,8,6,"@0:"+C$+",P,W"
230 GOSUB370
240 GET#5,D$:IFST<>0THEN280
250 IFD$=""THEND$=CHR$(0)
260 IFFTHENPRINT#6,E$;
270 F=-1:E$=D$:GOTO240
280 CLOSE5:OPEN5,8,5,B$+",P,R"
300 GOSUB370
310 GET#5,A$:GET#5,A$
320 GET#5,D$:T=ST
330 IFD$=""THEND$=CHR$(0)
340 PRINT#6,D$;
350 IFT=0THEN320
351 PRINT"[C.BAS][C.BAS][C.BAS]";
356 PRINTZ$" + "B$" ——>"S$":"C$
359 FORTD=1TO500:NEXT
360 CLOSE5:CLOSE6:CLOSE15:GOTO1000
361 REM*****************
370 INPUT#15,ER,ER$,ET,ES
380 IFER<>0THEN391
390 RETURN
391 PRINT"[CLR][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][RVS ON
    ][C.DROITE][C.DROITE]";
400 PRINTER","ER$","ET","ES;
410 FORTD=1TO500:NEXT
420 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS][C.BAS]":END
1000 PRINT"[C.BAS][C.BAS]ENCORE UNE FOIS ? (OUI OU NON)"
1010 GETA$:IFA$=""GOTO1010
1020 IFA$="O"GOTO150
1030 IFA$="N"GOTO1200
1040 GOTO1010
1200 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS] AU REVOIR
    !"
1210 FORI=1TO350:NEXT
1220 PRINT"[CLR]":NEW
55555 SAVE"@0:APPENDISK",8

READY.
```

ARC-EN-CIEL.

Affichage de 256 couleurs à l'écran par un mixage proportionnel des 16 couleurs de fond avec les 16 couleurs de tracé. Bel exemple d'utilisation du mode caractère étendu.

```
0 REM MODE GRAPHIQUE 6
5 DIM A$(7,3),CO$(15)
10 REM PREND EN ROM LES CODES DE COULEURS
12 FOR I=0TO15:READC:CO$(I)=CHR$(C):NEXTI
14 REM COPIE 64 CARACTERES ROM -> RAM
15 RA=32768:RO=53248:MA=64*8
16 PRINT"[CLR][NOIR]UN INSTANT, S.V.P..."
20 POKE56334,0:POKE1,PEEK(1)AND251
30 FORI=0TO MA:POKE RA+I,PEEK(RO+I)
40 NEXT I:POKE1,PEEK(1)OR4:POKE56334,1
60 POKE55,0:POKE56,128:REM SOMMET MEM.
70 POKE56576,197:REM VIC-II BANC 2
90 POKE53272,32:REM EC:34816,GEN:32768
110 REM DIT AU KERNAL OU EST L'ECRAN
120 POKE648,(34816/256):PRINTCHR$(147)
130 PRINT"[ROUGE]      # $ % & ´ ( ) *"
210 POKE53265,91:REM PASSE EN MODE ETENDU
215 POKE53280,11:REM COULEUR BORD
220 POKE53281,11:POKE53282,12:POKE53283,15:POKE53284,1
230 REM MODIFIE LES CARACTERES #$%&´()*
235 C=-1
240 FOR I=33048TO33104STEP8:C=C+1
250 FOR J=I TO I+7:POKEJ,0:REM OCTET J
260 FOR B=0 TO 7           :REM BIT   B
270 IF(RND(0)*7)>C THEN POKE J,PEEK(J)OR(2^B)
280 NEXTB,J,I
290 PRINT"[CLR]";
300 REM CREE LE TABLEAU DES GRIS     *
310 AD=1024:FORI=0TO7:FORJ=0TO1
320 A$(I,J)=CHR$(35+I+J*128)
330 A$(I,J+2)=CHR$(18)+A$(I,J)+CHR$(146)
340 NEXTJ,I
345 REM DESSIN PALETTE DES COULEURS
350 FORK=0TO3:FOR I=0TO3
360 FOR C=0TO3:PRINTCO$(4*K+C);
370 FOR J=0TO7
390 PRINTA$(J,I);:NEXTJ,C:PRINT:NEXTI,K
410 PRINT"POUSSER ESPACE"
420 GETA$:IFA$=""GOTO420
450 PRINT"[CLR]":FORK=0TO3:FOR I=0TO3
460 FOR CO=0TO3:FORJ=0TO7
490 PRINTCO$(4*I+CO);A$(J,K);:NEXT J,CO:PRINT:NEXTI,K
500 PRINT"POUSSER ESPACE"
997 GETA$:IFA$=""GOTO 997
998 POKE56576,199:POKE53272,21:POKE648,4
999 POKE53265,27:PRINT"[CLR]":END
1000 DATA144,5,28,159,156,30,31,158
1010 DATA129,149,150,151,152,153,154,155
55555 SAVE"@0:ARC-EN-CIEL",8

READY.
```

AZERTY

Programme en langage machine occupant les adresses 49152 à 49983, chargé en mémoire par un programme BASIC, il est activé ou réactivé après STOP-RESTORE par SYS 49152.

Après exécution du programme, le 64 est toujours QWERTY. Le fait d'enfoncer simultanément les touches LOGO (C=) et CTRL fait passer le clavier en mode AZERTY. La frappe des lettres donne les minuscules. Avec SHIFT, on obtient les majuscules. Les touches de la rangée supérieure donne normalement la ponctuation française soit : ç, é, è, etc...

Noter que le M est passé sur la touche :. La touche "N commercial" nous donne l'accent circonflexe.

L'enfoncement de LOGO et CTRL nous ramène en QWERTY normal. Comme d'habitude, LOGO + SHIFT inverse majuscules et minuscules. De plus, AZERTY rajoute la fonction suivante au clavier :

L'enfoncement simultané de SHIFT et CTRL en mode AZERTY inverse les tables de décodage de clavier "normal" et "avec SHIFT", ce qui a pour effet d'inverser les deux signes de chaque touche. Le but est de pouvoir entrer aisément des commandes BASIC en majuscules.

Si le programme est désactivé par STOP-RESTORE, le réactiver par SYS 49152. Désactiver avant les accès à la cassette.

```
5 REM  AZERTY POUR CBM-64
6 REM                    PAR A.SURNY 1984
7 REM  ACCES PAR SYS 49152
8 REM
10 CS=0:FOR I=49152TO49983:READ A:POKE I,A:CS=CS+A:PRINT".";:NEXT I
20 IF CS<>98419THEN STOP
30 SYS49152:REM ACTIVE L'AZERTY
40 END
49152 DATA 120,173,2,221,9,3,141,2
49160 DATA 221,173,0,221,41,252,141,0
49168 DATA 221,169,56,141,24,208,169,204
49176 DATA 141,136,2,165,1,41,249,133
49184 DATA 1,169,224,133,254,169,208,133
49192 DATA 252,160,0,132,251,132,253,177
49200 DATA 251,145,253,230,251,230,253,208
49208 DATA 246,230,252,230,254,208,240,169
49216 DATA 216,133,252,169,242,133,254,169
49224 DATA 8,133,251,133,253,177,251,145
49232 DATA 253,230,251,230,253,165,253,201
49240 DATA 217,208,242,169,243,133,254,169
49248 DATA 136,133,253,169,48,133,251,169
49256 DATA 193,133,252,177,251,145,253,230
49264 DATA 251,230,253,208,246,165,254,201
49272 DATA 251,240,4,169,251,208,222,132
49280 DATA 251,169,240,133,252,169,244,133
49288 DATA 254,177,251,73,255,145,253,230
49296 DATA 251,230,253,208,244,230,252,230
49304 DATA 254,240,12,165,254,201,248,208
49312 DATA 232,133,252,169,252,208,224,140
49320 DATA 52,3,169,208,141,143,2,169
49328 DATA 192,141,144,2,165,1,9,6
49336 DATA 133,1,88,96,234,234,234,234
49344 DATA 76,66,235,76,72,235,76,79
49352 DATA 235,234,234,234,234,234,234,234
49360 DATA 173,141,2,201,6,208,13,205
49368 DATA 142,2,240,228,173,24,208,73
49376 DATA 4,141,24,208,173,24,208,41
49384 DATA 4,201,4,208,214,173,141,2
49392 DATA 201,3,240,210,201,5,208,16
49400 DATA 205,142,2,240,195,173,52,3
49408 DATA 73,8,141,52,3,76,224,234
49416 DATA 10,201,8,144,2,169,6,13
```

```
49424 DATA 52,3,170,189,32,193,133,245
49432 DATA 189,33,193,133,246,76,224,234
49440 DATA 176,193,241,193,50,194,115,194
49448 DATA 180,194,245,194,50,194,115,194
49456 DATA 60,65,60,6,62,102,62,0
49464 DATA 60,65,60,102,126,96,60,0
49472 DATA 60,65,0,56,24,24,60,0
49480 DATA 60,65,60,102,102,102,60,0
49488 DATA 60,65,102,102,102,102,62,0
49496 DATA 48,28,60,6,62,102,62,0
49504 DATA 48,28,60,102,126,96,60,0
49512 DATA 48,28,102,102,102,102,62,0
49520 DATA 12,56,60,102,126,96,60,0
49528 DATA 102,0,60,102,126,96,60,0
49536 DATA 102,0,0,56,24,24,60,0
49544 DATA 0,60,96,96,96,60,12,56
49552 DATA 28,32,28,34,28,2,28,0
49560 DATA 0,28,34,28,0,0,0,0
49568 DATA 24,60,102,0,0,0,0,0
49576 DATA 255,255,255,255,255,255,255,255
49584 DATA 20,13,29,136,133,134,135,17
49592 DATA 34,122,113,39,119,115,101,1
49600 DATA 40,114,100,189,99,102,116,120
49608 DATA 183,121,103,33,98,104,117,118
49616 DATA 188,105,106,182,44,107,111,110
49624 DATA 41,112,108,45,58,109,191,59
49632 DATA 60,35,184,19,1,61,36,61
49640 DATA 38,95,4,185,32,2,97,3
49648 DATA 255,148,141,157,140,137,138,139
49656 DATA 145,51,90,81,52,87,83,69
49664 DATA 1,53,82,68,54,67,70,84
49672 DATA 88,55,89,71,56,66,72,85
49680 DATA 86,57,73,74,48,63,75,79
49688 DATA 78,190,80,76,164,47,77,64
49696 DATA 46,62,42,37,147,1,61,222
49704 DATA 43,49,95,4,50,32,2,65
49712 DATA 131,255,148,141,157,144,141,142
49720 DATA 143,145,150,219,176,151,173,174
49728 DATA 178,1,152,221,186,153,169,192
49736 DATA 163,92,154,172,165,155,162,161
49744 DATA 181,171,41,179,92,48,167,187
49752 DATA 180,170,166,175,182,220,62,91
49760 DATA 164,60,168,223,93,147,1,61
49768 DATA 222,63,129,95,4,149,160,2
49776 DATA 177,131,255,255,255,255,255,255
49784 DATA 255,255,255,28,23,1,159,26
49792 DATA 19,5,255,156,18,4,30,3
49800 DATA 6,20,24,31,25,7,158,2
49808 DATA 8,21,22,18,9,10,146,13
49816 DATA 11,15,14,255,16,12,255,255
49824 DATA 27,0,255,28,255,29,255,255
49832 DATA 31,30,255,144,6,255,5,255
49840 DATA 255,17,255,255,20,13,29,136
49848 DATA 133,134,135,17,51,90,81,52
49856 DATA 87,83,69,1,53,82,68,54
49864 DATA 67,70,84,88,55,89,71,56
49872 DATA 66,72,85,86,57,73,74,48
49880 DATA 63,75,79,78,190,80,76,164
49888 DATA 47,77,64,46,62,42,37,19
49896 DATA 1,61,222,43,49,95,4,50
49904 DATA 32,2,65,3,255,148,141,157
49912 DATA 140,137,138,139,145,34,122,113
49920 DATA 39,119,115,101,1,40,114,100
49928 DATA 189,99,102,116,120,183,121,103
49936 DATA 33,98,104,117,118,188,105,106
49944 DATA 182,44,107,111,110,41,112,108
49952 DATA 45,58,109,191,59,60,35,184
49960 DATA 147,1,61,36,61,38,95,4
49968 DATA 185,32,2,97,131,255,255,255
49976 DATA 255,255,255,255,255,255,255,255
55555 SAVE"@0:AZERTY",8

READY.
```

BALAYAGE.

Démonstration du SID employé en balayage de phase et en inversion de phase. A défaut d'être musical, ce programme démontre quelques fantastiques possibilités sonores du 64.

```
10 REM BALAYAGE DE PHASES
20 :
30 S=54272:GOSUB500:VL=15:AR=64:DE=AR+1
40 PRINT"NORMAL (1)":PRINT"PHASE ALEATOIRE (2)"
45 PRINT"INVERSION DE PHASE (3)"
50 PRINT"BALAYAGE DE PHASE (4)"
60 INPUT"1,2,3,4 OU 0=FIN";B
70 IF B=0 THEN END
80 PL=128:POKES+2,PL
90 AD=3*16+2:SR=14*16+5:POKE S+24,VL
100 RESTORE:POKE S+5,AD:POKE S+6,SR
110 READ N,D:IF N=OTHEN GOSUB500:RUN
120 POKE S+1,N:POKES+4,DE
130 ON B GOSUB 300,600,700,800
140 POKE S+4,AR
150 ON B GOSUB 300,600,700,800
160 GOTO110
199 :
200 DATA25,4,28,4,25,4,25,4,25,2,28,2
210 DATA32,12,25,4,28,4,19,4,19,2,19,2
230 DATA21,1,24,1,25,4,24,2,19,4,0,0
299 :
300 FOR TD=0TO70*D:NEXT:RETURN:REM NORMAL
399 :
500 FORT=S TO S+24:POKET,0:NEXT:RETURN:REM SILENCE
599 :
600 FOR TD=0TO6*D:POKE S+3,15*RND(0):W=ATN(W):REM ALEATOIRE
610 NEXT:RETURN
699 :
700 FOR TD=0 TO2*D:REM INVERSION
710 POKE S+3,4:W=ATN(W):POKES+3,11:W=ATN(W)
720 NEXTTD:RETURN
799 :
800 FOR K=0TOD:FOR TD=0TO15STEP2:REM BALAYAGE
810 POKE S+3,TD:NEXT TD,K:RETURN
899 :
55555 SAVE"@0:BALAYAGE",8

READY.
```

BASIC-FR

Transformation de l'interpréteur BASIC après recopie de la ROM en RAM. Il faut taper 'RUN' et attendre une bonne minute pour rendre actif BASIC-FR. De petits points apparaîssent à l'écran à chaque modification qui s'effectue dans l'interpréteur. Le message "PRET" qui remplace le "READY" habituel vous signale que BASIC-FR est actif.

Si STOP-RESTORE désactive BASIC-FR, on peut le rétablir par: POKE 1, 54.

Essayez par exemple : PRINT SQR(-3)
pour admirer un message d'erreur en français.

```
6 REM BASIC FRANCAIS PAR RECOPIE
7 REM ET MODIFICATION DE LA ROM BASIC
8 REM
10 FORI=4096TO49151:REM RECOPIE EN RAM
20 POKEI,PEEK(I):NEXTI
30 GOSUB 1000  :REM MODIFIE LE BASIC
40 POKE1,54:END:REM MISE EN SERVICE
999 REM *********************************
1000 REM MODIFIE LE BASIC
1010 POKE41842,69 :REM IN -> EN
1020 AD=41848:GOSUB2000:REM READY->PRET
1030 POKE41833,0:REM SUPPRIME "ERROR"
1040 AD=49152:REM MESSAGES EN $C000
1050 TA=41768:REM ADRESSES DES MESSAGES
1060 FOR MS=1TO29:GOSUB3000
1065 GOSUB2000:NEXT MS
1070 B=INT(AD/256):POKE43086,B
1080 B=AD-B*256:POKE43084,B
1090 READA$:A$=CHR$(13)+CHR$(10)+A$
1100 L=LEN(A$)
1110 GOSUB3000:GOSUB2045:REM MESS.BREAK
1120 POKEAD,0:AD=AD+1:RETURN
1999 REM *********************************
2000 REM MET UN MESSAGE EN MEMOIRE
2010 REM A L'ADRESSE AD AVEC LE BIT 7
2020 REM DU DERNIER CARACTERE = 1
2030 REM AD EST PRET POUR LE SUIVANT.
2040 READ A$:L=LEN(A$)
2045 REM GOSUB ICI AVEC A$ ET L
2050 FOR I=1TO L-1
2060 B=ASC(MID$(A$,I,1)):POKE AD,B
2070 AD=AD+1:NEXT I
2080 POKE AD,ASC(RIGHT$(A$,1))OR128
2999 PRINT".";:AD=AD+1:RETURN
3000 REM ECRIT EN TA LA VALEUR AD
3010 REM SUR 2 OCTETS: PARTIE BASSE,
3020 REM PARTIE HAUTE.A LA FIN ,TA EST
3030 REM PRET POUR LE SUIVANT
3040 B=INT(AD/256):POKE TA+1,B
3050 B=AD-B*256:POKE TA,B
3999 TA=TA+2:RETURN
10000 DATA"PRET  ":REM PRET+2 ESPACES
10010 DATA"ERREUR:TROP DE FICHIERS"
10020 DATA"ERREUR:FICHIER DEJA OUVERT"
10030 DATA"ERREUR:FICHIER PAS OUVERT"
10040 DATA"ERREUR:FICHIER INTROUVABLE"
10050 DATA"ERREUR:PERIPHERIQUE ABSENT"
10060 DATA"ERREUR:PAS OUVERT POUR ENTREE"
10070 DATA"ERREUR:PAS OUVERT POUR SORTIE"
10080 DATA"ERREUR:NOM DE FICHIER MANQUQNT"
10090 DATA"ERREUR:NUMERO DE PERIPHERIQUE ILLEGAL"
10100 DATA"ERREUR:INSTRUCTION "NEXT" NON PRECEDEE DE "FOR""
10110 DATA"ERREUR DE SYNTAXE"
10120 DATA"ERREUR INSTRUCTION "RETURN" NON PRECEDEE DE "GOSUB""
10130 DATA"ERREUR:PLUS DE DONNEES"
10140 DATA"ERREUR:VALEUR NON AUTORISEE"
10150 DATA"ERREUR DE DEBORDEMENT"
10160 DATA"ERREUR:PLUS DE MEMOIRE LIBRE"
10170 DATA"ERREUR:NUMERO DE LIGNE INEXISTANT"
10180 DATA"ERREUR:INDICE NON VALABLE"
10190 DATA"ERREUR:TABLEAU REDIMENSIONNE"
10200 DATA"ERREUR:DIVISION PAR ZERO"
10210 DATA"ERREUR:INTERDIT EN MODE DIRECT"
10220 DATA"ERREUR:TYPE NON CONCORDANT"
10230 DATA"ERREUR:CHAINE DE CARACTERE TROP LONGUE"
10240 DATA"ERREUR DE DONNEE DANS LE FICHIER"
10250 DATA"ERREUR:FORMULE TROP COMPLEXE"
10260 DATA"IMPOSSIBLE DE CONTINUER APRES CE TYPE  D'ERREUR"
10270 DATA"ERREUR:FONCTION NON DEFINIE"
10280 DATA"ERREUR:VERIFICATION INCORRECTE"
10290 DATA"ERREUR DE CHARGEMENT"
10300 DATA"ARRET PROVOQUE ":REM UN ESPACE A LA FIN
55555 SAVE"BASIC-FR"

READY.
```

CALCUL SON

Ce programme calcule les valeurs à écrire dans les registres du SID en fonction de la fréquence désirée. Le résultat affiché est donné sous la forme (valeur, partie basse, partie haute) en décimal. les fréquences supérieures à 3848 HZ ne peuvent pas être utilisées.

Entrer la valeur 0 pour sortir du programme.

```
5 REM CALCUL LES VALEURS HAUTE ET BASSE
6 REM A ECRIRE DANS LES REGISTRES DE
7 REM FREQUENCE DU CBM-64 EUROPEEN
8 :
9 :
10 INPUT "FREQUENCE ";A:IF A=0THEN END
20 F=INT(A/.05872535+.5)
30 FH=INT(F/256)
40 FL=F-FH*256
50 PRINT"  F , FL , FH :";F;FL;FH
60 RUN
55555 SAVE"@0:CALCUL SON",8

READY.
```

CENTRO-BAS.

Programme permettant de tester la connection parallèle 'CENTRONICS' entre le port utilisateur et une imprimante. Le brochage du câble est indiqué en remarque dans le texte du programme. Noter que les broches A à M du port utilisateur sont les connections inférieures.

```
5 REM IMPRIMANTE PARALLELE SUR PORT UTILISATEUR POUR CBM 64
6 REM CE CABLE EST COMPATIBLE AVEC VIZAWRITE
7 REM
8 REM
9 REM
10 REM
11 REM
12 REM CABLE : PRINTER : CBM - 64
13 REM         ::::::::::::::::::::
14 REM         STROBE  : U.PORT  M
15 REM         ACK     : U.PORT  B
16 REM         GROUND  : U.PORT  A
17 REM         DATA 0-7: U.PORT  C-L
18 REM
19 REM
20 GOSUB 110:REM INITIALISATION
30 :
40 INPUTA$
50 IF A$="END"THENEND
60 A$=A$+CHR$(13)+CHR$(10)
70 FORI=1 TO LEN(A$):C=ASC(MID$(A$,I,1)):GOSUB170:NEXT
80 A$="":GOTO 40
90 :
100 :
110 REM****INITIALISE REGISTRES DU CIA
```

```
120 V=56576:REM ADRESSE DU CIA #2
130 POKE V+3,255:POKE V+1,0:REM PORT B EN SORTIE (DONNEES)
140 POKE V+2,PEEK(V+2)OR4:REM PORTA BIT2 EN SORTIE (STROBE)
150 RETURN
160 :
170 REM******ENVOIE CARACTERE (CODE ASCII = C)
180 POKE V+1,C :REM CARACTERE A ENVOYER
190 SS=PEEK(V):REM ENVOI DE L'IMPULSION STROBE SUR LE PORT A BIT 2
200 POKE V,SS AND 251
210 POKE V,SS
220 C=PEEK(V+13):IFC<>16 GOTO220 :REM ATTEND IMPULSION ACK
230 RETURN
55555 SAVE"@0:CENTRO.BAS",8

READY.
```

CLAVIER SONORE.

Petit programmme en langage machine qui se charge dans le tampon de cassette. Il génère un BIP à chaque frappe de touche.

Attention, il n'y a qu'un seul BIP pour les touches à répétition. Si l'activation est perdue par STOP-RESTORE, on peut réactiver par SYS 828. Désactiver avant tout accès à la cassette, bien entendu!

```
10 REM CLAVIER SONORE
15 REM *********************************
20 FOR AD= 828 TO 883:READ A
30 CS=CS+A:POKE AD,A:NEXT AD
40 IFCS <> 6183 THEN PRINT"ERREUR":STOP
50 SYS828:END
60 REM *********************************
828 DATA120,169,073,141,143,002,169,003
836 DATA141,144,002,088,096,165,197,205
844 DATA126,003,240,032,141,126,003,169
852 DATA010,141,024,212,141,005,212,169
860 DATA004,141,006,212,169,033,141,001
868 DATA212,141,000,212,141,004,212,169
876 DATA032,141,004,212,076,072,235,001
877 REM CHECKSUM DE 828 A 883 = 6183
55555 SAVE"@0:CLAVIER SONORE",8

READY.
```

CLEAR-SRC.

Il s'agit de la source assembleur d'une routine d'initialisation d'image à haute résolution. On peut assembler ce programme avec l'assembleur standard vendu par COMMODORE. Une version assemblée de ce programme est utilisée dans "MODES MIXTES".

```
00001  0000             ;*****************************************
00002  0000             ;* PROGRAMME 'CLEAR.OBJ'                 *
00003  0000             ;,  EFFACE L'ECRAN BITMAP ET LA          ,
00004  0000             ;* RAM COULEUR A UNE VALEUR PREDEFINIE   *
00005  0000             ;* ACCES PAR: POKE 49360,COULEUR:SYS 49361  *
00006  0000             ;*****************************************
00007  0000             ;ATTENTION, DANS CE LISTING:
00008  0000             ;'PLUS GRAND QUE' EST REPRESENTE PAR: '¯'
00009  0000             ;'PLUS PETIT QUE' EST REPRESENTE PAR: '^'
00010  0000             ;*****************************************
00011  0000             ;
00012  0000                     * = $C0D0       ; ICI 49360
00013  C0D0             ;
00014  C0D0             COUL   = $CC3C        ;ADRESSE DE LA RAM COULEUR
00015  C0D0             FINCOU = $D000        ;ADRESSE FIN DE RAM COULEUR +1
00016  C0D0             ECRAN  = $E000        ;ADRESSE DEBUT ECRAN BITMAP
00017  C0D0             FINEC  = $0000        ;ADRESSE DE FIN D'ECRAN BITMAP +1
00018  C0D0             ;
00019  C0D0  03          COLOR  .BYT $03       ;IL FAUT POKER ICI LA COULEUR DE L'ECRAN
00020  C0D1             ; ACCES PAR :  SYS ( 49361 )
00021  C0D1  A9 3C      CLEAR  LDA #<COUL
00022  C0D3  85 62             STA $62         ;ADRESSE DE TRAVAIL
00023  C0D5  A9 CC             LDA #>COUL
00024  C0D7  85 63             STA $63
00025  C0D9  A9 00             LDA #<FINCOU    ;ADRESSE DE FIN
00026  C0DB  85 64             STA $64
00027  C0DD  A9 D0             LDA #>FINCOU
00028  C0DF  85 65             STA $65
00029  C0E1  AD D0 C0          LDA COLOR
00030  C0E4  A0 00             LDY #$00
00031  C0E6  20 03 C1  CLEAR1  JSR POKTST     ;ECRIT , INCREMENTE ADR, TESTE
00032  C0E9  D0 FB             BNE CLEAR1
00033  C0EB  A9 00             LDA #<ECRAN
00034  C0ED  85 62             STA $62
00035  C0EF  A9 E0             LDA #>ECRAN
00036  C0F1  85 63             STA $63
00037  C0F3  A9 00             LDA #<FINEC
00038  C0F5  85 64             STA $64
00039  C0F7  A9 00             LDA #>FINEC     ;ET DE MEME POUR LA REMISE
00040  C0F9  85 65             STA $65         ;A ZERO DE L'ECRAN BITMAP
00041  C0FB  A9 00             LDA #$00
00042  C0FD  20 03 C1  CLEAR2  JSR POKTST     ;EFFACE UN OCTET ECRAN
00043  C100  D0 FB             BNE CLEAR2
00044  C102  60                RTS
00045  C103             ;
00046  C103  91 62      POKTST STA ($62),Y     ;POKE
00047  C105  E6 62             INC $62         ;INCREMENTE L'ADRESSE
00048  C107  D0 02             BNE POK2
00049  C109  E6 63             INC $63
00050  C10B  A6 63      POK2   LDX $63         ;COMPARE AVEC L'ADRESSE DE FIN+1
00051  C10D  E4 65             CPX $65
00052  C10F  D0 04             BNE POKRTS
00053  C111  A6 62             LDX $62
00054  C113  E4 64             CPX $64
00055  C115  60         POKRTS RTS
```

CODE-ECRAN.

Petite routine qui boucle pour vous afficher en permanence le code ASCII et le code POKE d'une touche frappée au clavier. Les caractères qu'on ne peut afficher par POKE donnent le code-écran 32 (espace). Pour sortir du programme, taper RETURN.

```
5 REM   CODE-ECRAN
6 REM
7 REM AFFICHE UN CARACTERE, SON CODE ASCII
8 REM ET SON CODE-ECRAN (POUR POKE)
9 REM
10 CS=0:FORI=679TO730:READA:POKEI,A:CS=CS+A:NEXTI
20 IFCS<>7665THENSTOP
30 SYS 679
679 DATA 32,228,255,240,251,170,201,13
687 DATA 208,1,96,169,147,32,210,255
695 DATA 169,255,133,212,138,32,210,255
703 DATA 169,32,32,210,255,169,0,32
711 DATA 205,189,169,32,32,210,255,169
719 DATA 0,133,212,174,0,4,32,205
727 DATA 189,232,208,204
55554 END
55555 SAVE"@0:CODE-ECRAN",8

READY.
```

COPIE CARA.BAS.

Recopie du générateur de caractères de ROM en RAM. Les paramètres RAM, ROM sont les adresses de destination et de début du déplacement. MAX est la taille du bloc à recopier. Ces paramètres sont bien sûr modifiables à volonté.

```
10 RAM=20000:ROM=53248:MAX=16
20 POKE56334,PEEK(56334)AND254
30 POKE1,PEEK(1)AND251
40 FORI=0TOMAX:POKERAM+I,PEEK(ROM+I):NEXTI
50 POKE1,PEEK(1)OR4
60 POKE56334,PEEK(56334)OR1
99 END
55555 SAVE"@0:COPIECARA.BAS",8

READY.
```

COPIE ROM.

Ce programme en langage machine est chargé par quelques lignes de BASIC dans le tampon cassette. Une fois utilisé, il peut être effacé. L'ordinateur continue à fonctionner mais avec les ROMS BASIC et KERNAL hors service et remplacées par une copie de leur contenu en RAM aux mêmes adresses. A utiliser avant de tenter des modifications dans le BASIC ou le KERNAL. Peut être réactivé par SYS 828 après une perte de contrôle par STOP-RESTORE.

```
10 REM RECOPIE LES ROM BASIC ET KERNAL
20 REM EN RAM ET LES DECONNECTE
30 REM LE BASIC EST DONC MODIFIABLE
40 CS=0:FORI=828TO882:READA:POKEI,A
50 CS=CS+A:NEXT
60 IFCS<>6061THENSTOP
70 SYS828
80 PRINT"PLUS DE ROM EN SERVICE":END
```

```
828 DATA169,160,141,092,003,169,192,141
836 DATA111,003,032,089,003,140,111,003
844 DATA169,224,141,092,003,032,089,003
852 DATA169,053,133,001,096,160,000,162
860 DATA000,132,098,134,099,177,098,145
868 DATA098,200,208,249,238,092,003,173
876 DATA092,003,201,000,208,231,096
877 REM CHECKSUM= 6061 DE 828 A 882
55555 SAVE"@0:COPIEROM",8

READY.
```

COULEUR.

Petit programme permettant de modifier le choix des couleurs de fond et de bord de l'écran.

```
10 INPUT"[CLR]BORD,FOND";A,B:POKE 53280,A:POKE 53281,B
11 PRINT"[NOIR][RVS ON]                "0
12 PRINT"[BLANC][RVS ON]               "1
13 PRINT"[ROUGE][RVS ON]               "2
14 PRINT"[CYAN][RVS ON]                "3
15 PRINT"[POURPRE][RVS ON]               "4
16 PRINT"[VERT][RVS ON]                "5
17 PRINT"[BLEU][RVS ON]                "6
18 PRINT"[JAUNE][RVS ON]                 "7
21 PRINT"[ORANGE][RVS ON]                  "8
22 PRINT"[ORANGE L.][RVS ON]                 "9
23 PRINT"[ROSE][RVS ON]                "10
24 PRINT"[CYAN L.][RVS ON]                 "11
25 PRINT"[POURPRE L.][RVS ON]                "12
26 PRINT"[VERT L.][RVS ON]               "13
27 PRINT"[BLEU L.][RVS ON]               "14
28 PRINT"[JAUNE L.][RVS ON]                "15
30 PRINT"POUSSER ESPACE POUR FINIR"
31 GET A$:IF A$=""GOTO 31
32 IF A$=" " THEN END
99 RUN
55555 SAVE"@0:COULEUR",8

READY.
```

DATAMAKER.

Transformation d'une zone mémoire en lignes de DATA. La somme de vérification de la zone est également calculée. En fin de programme, les lignes de DATA sont présentes à l'écran. En poussant quelquefois RETURN, les lignes sont entrées en mémoire. Il suffit ensuite de taper :

```
1 <RETURN>
2 <RETURN>
3 <RETURN>
4 <RETURN>
5 <RETURN>
6 <RETURN>
555555 <RETURN>
```

pour effacer les lignes du "DATAMAKER" et ne conserver en mémoire que les lignes générées.

```
0 INPUT"[NOIR][CLR]DEBUT,FIN";D,F:PRINT"[CLR][C.BAS][C.BAS]"
1 FORI=DTOFSTEP8:PRINTI"DATA";:FORJ=0TO6
2 GOSUB 3:PRINT",";:NEXTJ:GOSUB3:PRINT:NEXTI:END
3 A$=STR$(PEEK(I+J)):B$=RIGHT$(A$,LEN(A$)-1):CS=CS+PEEK(I+J)
4 A$=RIGHT$("000"+B$,3):PRINTA$;
5 IF(I+J)=FTHENPRINT:PRINT:PRINTI+1;"REM [CYAN]CHECKSUM="CS"DE"D"A"F"[H
    OME]":END
6 RETURN
55555 SAVE"@0:DATAMAKER",8

READY.
```

DATA-SUR-DISQUE

Ce programme BASIC est un générateur de programmes. Il saisit un programme enregistré sur disque et le transforme en un nouveau programme (BASIC) qui contient sous forme de DATA la suite des octets du programme lu. Ce programme est à utiliser pour transformer un programme en langage machine en mémoire haute en un chargeur BASIC. On pourra éventuellement reprendre ces lignes dans un programme BASIC qui utilise la routine en question. Après réponse à la question "nom de programme à lire", un nouveau programme est généré qui porte le même nom suivi de ".D". Une première lecture du programme source permet d'en déterminer les adresses de début et de fin. Ensuite le programme ".D" est généré à la deuxième passe. Un dièse est affiché à l'écran lors de la génération d'une ligne de BASIC. On ne peut évidemment pas créer par ce procédé un programme chargeur de bas de mémoire car il s'écraserait lui-même. Cependant, générer un programme BASIC en mémoire basse est possible en modifiant le bas de BASIC avec INIMEM avant d'utiliser le programme ".D". Essayez, c'est d'une efficacité surprenante !

```
10 REM LIT UN PROGRAMME SUR DISQUE ET LE
20 REM RANGE EN DATA DANS UN AUTRE PROGRAMME
30 INPUT"NOM DE PROGRAMME A LIRE";F1$
40 F2$=F1$+".D":PRINT:PRINT"NOUVEAU PROGRAMME = ";F2$:PRINT
50 OPEN15,8,15:OPEN8,8,8,F1$+",P,R":OPEN9,8,9,"@0:"+F2$+",P,W"
60 GOSUB 1000:REM ERREUR DISQUE ?
70 GET#8,L$,H$:PRINT#9,CHR$(1)CHR$(4);
80 LI$=CHR$(1)+CHR$(1):C$=CHR$(0):PRINT#9,LI$;
90 FOR I=1TO10:READA:PRINT#9,CHR$(A);:NEXT I
100 L=ASC(L$+C$):H=ASC(H$+C$):SA=L+256*H
110 PRINT#9,MID$(STR$(SA),2);
120 PRINT"ADRESSE DE DEBUT=";SA:PRINT:CS=0:EA=SA
130 GET#8,A$:IFST<>0 GOTO 150
140 EA=EA+1:CS=CS+ASC(A$+C$):GOTO 130
150 CS=CS+ASC(A$+C$)
160 CLOSE8:OPEN8,8,8,F1$+",P,R":GET#8,A$,A$
170 EA=EA:PRINT"ADRESSE DE FIN =";EA:PRINT
180 PRINT#9,CHR$(164);MID$(STR$(EA),2);
190 FOR I=1TO29:READA:PRINT#9,CHR$(A);:NEXT I
200 PRINT#9,MID$(STR$(CS),2)CHR$(167)CHR$(144);C$;
500 C=1:PRINT#9,LI$CHR$(L)CHR$(H)CHR$(131)CHR$(32);
510 GET#8,A$:S=ST:PRINT#9,MID$(STR$(ASC(A$+C$)),2);
520 C=C+1:IF C=9 OR S<>0 GOTO 540
530 PRINT#9,",";:GOTO 510
540 PRINT"#";:IF S GOTO 570
550 L=L+8:IF L>255 THENL=L-256:H=H+1
560 PRINT#9,C$;:GOTO 500
570 PRINT#9,C$C$C$;
```

```
580 CLOSE8:CLOSE9:CLOSE15:PRINT:PRINT"PROGRAMME ";
590 PRINT F2$" CREE SUR DISQUE":END
999 :
1000 REM LIT LE CANAL ERREUR DISQUE
1010 INPUT#15,W,X$,Y,Z:IF W=0THEN RETURN
1020 PRINT"ERREUR "W;X$;Y;Z:GOTO 160
1999 :
2000 DATA10,0,67,83,178,48,58,129,73,178
2010 DATA58,135,65,58,151,73,44,65,58,67,83
2020 DATA178,67,83,170,65,58,130,73,0,58
2030 DATA8,20,0,139,67,83,179,177
2999 :
55555 SAVE"@0:DATA-SUR-DISQUE",8

READY.
```

DE-NEW.

Ce programme imprime sur l'imprimante ou sur l'écran la suite d'instructions nécessaires pour récupérer un programme BASIC en mémoire après NEW ou RESET. Cette suite d'instructions est à introduire en mode direct. Sous forme de programme, le fonctionnement est impossible car le fait de charger un programme détruirait le texte à récupérer encore présent en mémoire. Une bonne idée est de recopier ce texte et de le garder à portée de main, cela peut toujours servir.

```
10 INPUT"ECRAN(3) OU IMPR. (4)";A
20 OPEN4,A:CMD4
30 PRINT
40 PRINT:PRINT:PRINT"DE-NEW : RECUPERATION D'UN PROGRAMME BASIC EN MEMO
     IRE
50 PRINT:  PRINT"          APRES RESET OU NEW.":PRINT:PRINT
55 PRINT"A INTRODUIRE EN MODE DIRECT, BIEN ENTENDU !!!!
60 PRINT:PRINT"POKE46,159:CLR
70 PRINT"FOR I=2054 TO 2136:IF PEEK(I-1)THEN NEXT"
80 PRINT"J=INT(I/256):POKE 2050,J:POKE 2049,I-J*256"
90 PRINT"LIST    (MAIS PAS ENCORE RUN !!)
100 PRINT"FOR I=2056 TO 159*256:IF PEEK(I-1)+PEEK(I-2)+PEEK(I-3)THEN NE
     XT
110 PRINT"J=INT(I/256):POKE 256,J:POKE257,I-J*256
120 PRINT"(DELAI D'ATTENTE SUIVANT LONGUEUR DU PROGRAMME A RECUPERER )

130 PRINT"POKE 46,PEEK(256):POKE 47,PEEK(257)
140 PRINT"RUN  ( OU AU MOINS CLR )
150 PRINT"ET VOILA          ....C'EST RECUPERE !!"
160 PRINT#4:CLOSE4
999 END
55555 SAVE"@0:DE-NEW",8

READY.
```

DE-NEW-AUTO.

Petit programme en langage machine chargé par 16 lignes de BASIC. Il se loge en 32768 à 32798. Après avoir redescendu le sommet de BASIC pour se protéger (ce qui diminue la zone BASIC de 8K), il attend tranquillement que le bouton 'RESET' soit enfoncé !!! Ce programme est la principale motivation de ceux qui prendront la peine d'installer ce bouton!!!

Dès que le RESET a lieu, le KERNAL teste l'adresse 32768 pour y détecter une éventuelle cartouche ROM. DE-NEW-AUTO simule cette dernière à s'y méprendre (pour un 6510, en tout cas!). Le KERNAL passe donc le contrôle à DE-NEW-AUTO qui récupère le programme BASIC en mémoire.

Pour essayer, exécutez 'DE-NEW-AUTO', puis essayez de le lister : c'est impossible car il s'est effacé par le NEW de la ligne 30. Enfoncez RESET et tapez 'LIST', le programme est de nouveau là!

Le revers de la médaille est malheureusement désagréable. La procédure DE-NEW-AUTO ne réinitialise pas toutes les variables en page 0. Si dès lors certaines de celles-ci étaient détruites par erreur, elles ne seraient pas restaurées. DE-NEW-AUTO fonctionne cependant pour la grande majorité des cas.

```
1 REM DE-NEW-AUTO
2 REM
3 REM PERMET DE RECUPERER UN PROGRAMME BASIC
4 REM EN POUSSANT SUR RESET
5 REM APRES RECUPERATION, TAPER "CLR"
6 REM DESCEND LE SOMMET DE BASIC EN $8000
9 :
10 CS=0:FORI=32768TO32798:READA:POKEI,A:CS=CS+A:NEXTI
20 IFCS<>28477THENSTOP
30 POKE52,128:POKE56,128:NEW
32768 DATA 9,128,94,254,195,194,205,56
32776 DATA 48,169,8,141,2,8,32,51
32784 DATA 165,24,165,34,105,2,133,45
32792 DATA 165,35,105,0,133,46,96
55555 SAVE"@0:DE-NEW-AUTO",8

READY.
```

DESSIN.

Au chargement, de petites étoiles apparaissent au fur et à mesure de la création en mémoire des dessins de lutins à partir des 'DATA'. Le curseur-lutin-crayon apparaît ensuite au centre de l'écran.

Le curseur se déplace à volonté en manipulant le joystick raccordé au CONTROL PORT 2. Le bouton de tir ramène le curseur au centre. La touche 'CLR' efface l'écran dont la résolution est ici de 40*25 (1 pixel = 1 caractère). Les touches de fonctions modifient le type de curseur :

F1 : crayon avec pointe
F3 : crayon sans pointe
F5 : gomme
F7 : curseur invisible

Remarquer l'usage de la variable 'BIC' qui indique le type de curseur.

```
3 REM ********************************
4 REM *                              *
5 REM * DESSIN AVEC JOYSTICK  2  EN  *
6 REM *                              *
7 REM * BASSE RESOLUTION MONOCHROME  *
8 REM *                              *
9 REM ********************************
10 :
20 PRINT"[CLR]UN INSTANT, S.V.P."
30 XMAX=40:YMAX=25:HOME=1024:CO=55296
60 GOSUB 1000
70 PRINT"[CLR]":FOR I=CO TO CO+1024
80 POKE I,0:NEXT I:REMPLIT RAM COULEUR
90 X=20:Y=12:BIC=1
95 A$=" "
100 REM BOUCLE PRINCIPALE *************
110 A=127-PEEK(56464):REM JOYSTICK 2
115 GET A$
120 IF(AAND16)THEN X=20:Y=12:BIC=2:REM BOUTON DE TIR
130 IF A$= "[CLR]" THEN GOTO 70       :REM      EFFACE ECRAN
140 IF(AAND4) THEN X=X-1 :REM A GAUCHE
150 IF(AAND8) THEN X=X+1 :REM A DROITE
160 IF(AAND1) THEN Y=Y-1 :REM EN HAUT
170 IF(AAND2) THEN Y=Y+1 :REM EN BAS
180 IFA$=CHR$(133)THENBIC=1:REM F1 SORT LA PLUME
185 IFA$=CHR$(134)THENBIC=2:REM F3 RENTRE LA PLUME
187 IFA$=CHR$(135)THENBIC=3:REM F5 GOMME
188 IFA$=CHR$(136)THENBIC=0:REM F7 RIEN DU TOUT
200 IF X>=XMAX THEN X=XMAX-1
210 IF X<0 THEN X=0
220 IF Y>=YMAX THEN Y=YMAX-1
230 IF Y<0 THEN Y=0
250 CUR=HOME+X+(Y*XMAX)
260 GOSUB 2000
370 ON(BIC+1)GOSUB3000,4000,5000,6000
380 IF BIC=1THEN POKE CUR,160:REM ECRIT
390 IF BIC=3THEN POKE CUR,32:REM EFFACE
990 GOTO100
999 ********************************
1000 REM CONFIGURE LES 3 SPRITES
1020 VIDEO=53248
1030 PLUME=832:REM SPRITE-BLOCS NUMERO 13,14,15
1125 FOR H=0 TO 2
1130 FOR I=0 TO 20:READ A$:FOR K=0 TO 2:T=0:FOR J=0 TO 7:B=0
1131 IF MID$(A$,J+K*8+1,1)="Q"THEN B=1
1135 T=T*2+B:NEXT J:PRINT"*";:POKE832+64*H+I*3+K,T:NEXTK,I,H
1140 POKE VIDEO+29,1 :REM EXPANSION HORIZONTALE
1150 POKE VIDEO+23,1 :REM EXPANSION VERTICALE
1160 POKE VIDEO+28,1 :REM SPRITE MULTICOLORE
1170 POKE VIDEO+37,0 :REM COULEUR 1 = NOIR
1180 POKE VIDEO+38,4 :REM COULEUR 2 = MAUVE
1190 POKE VIDEO+39,1 :REM COULEUR SPRITE = BLANC
1990 RETURN
1999 ********************************
2000 REM POSITIONNE LE SPRITE 0
2010 XX=24+X*8:YY=50+Y*8
2030 IF XX < 256 THEN POKE VIDEO+16, 0:GOTO 2100
2040 XX=XX-256:POKE VIDEO+16,1
2100 POKE VIDEO,XX:REM HORIZ. MIN=24,MAX=336
2200 POKE VIDEO+1,YY:REM VERT. MIN=50,MAX=242
2990 RETURN
2999 ********************************
3000 REM SPRITE0 INVISIBLE
3990 POKE VIDEO + 21,0 :RETURN
3999 ********************************
4000 REM SPRITE0 = CRAYON ET POINTE
4010 POKE 2040,13:REM ADRESSE SPRITE0
4990 POKE VIDEO+21,1:RETURN
```

```
4999 **********************************
5000 REM SPRITE0 = CRAYON SANS POINTE
5010 POKE 2040,14:REM ADRESSE SPRITE0
5990 POKE VIDEO+21,1:RETURN
5999 **********************************
6000 REM SPRITE0 = GOMME
6010 POKE 2040,15:REM ADRESSE SPRITE0
6990 POKE VIDEO+21,1:RETURN
6999 **********************************
10000 DATA"                        "
10010 DATA"                        "
10020 DATA"                        "
10030 DATA"    Q QQ                "
10040 DATA"    Q QQ Q              "
10050 DATA"    QQ Q Q Q            "
10060 DATA"  Q Q Q Q Q Q            "
10070 DATA"    Q Q Q Q Q Q Q       "
10080 DATA"      Q Q Q Q Q Q Q     "
10090 DATA"        Q Q Q Q Q Q Q   "
10100 DATA"          Q Q Q Q Q Q Q "
10110 DATA"            Q Q Q Q Q Q "
10120 DATA"              Q Q Q Q   "
10130 DATA"                Q Q     "
10140 DATA"                        "
10150 DATA"                        "
10160 DATA"                        "
10170 DATA"                        "
10180 DATA"                        "
10190 DATA"                        "
10200 DATA"                        "
15000 DATA"                        "
15010 DATA"                        "
15020 DATA"                        "
15030 DATA"  Q Q Q                 "
15040 DATA"  Q Q Q Q               "
15050 DATA"  Q Q Q Q Q             "
15060 DATA"  Q Q Q Q Q Q           "
15070 DATA"    Q Q Q Q Q Q Q       "
15080 DATA"      Q Q Q Q Q Q Q     "
15090 DATA"        Q Q Q Q Q Q Q   "
15100 DATA"          Q Q Q Q Q Q Q "
15110 DATA"            Q Q Q Q Q Q "
15120 DATA"              Q Q Q Q   "
15130 DATA"                Q Q     "
15140 DATA"                        "
15150 DATA"                        "
15160 DATA"                        "
15170 DATA"                        "
15180 DATA"                        "
15190 DATA"                        "
15200 DATA"                        "
20000 DATA"                        "
20010 DATA"                        "
20020 DATA"                        "
20030 DATA"    Q Q Q Q Q           "
20040 DATA"    Q QQQQQQQ Q         "
20050 DATA"    QQQ QQQQQQQ Q       "
20060 DATA"    QQQQQ QQQQQQQ Q     "
20070 DATA"      QQQQQ QQQQQQQ Q   "
20080 DATA"        QQQQQ QQQQQQQ Q "
20090 DATA"          QQQQQ Q Q Q Q Q"
20100 DATA"            QQQ QQQQQQQ Q"
20110 DATA"              Q QQQQQQQ Q"
20120 DATA"                Q Q Q Q Q"
20130 DATA"                        "
20140 DATA"                        "
20150 DATA"                        "
20160 DATA"                        "
20170 DATA"                        "
20180 DATA"                        "
20190 DATA"                        "
20200 DATA"                        "
55555 SAVE"@0:DESSIN",8

READY.
```

DESSIN BITMAP.

Au chargement, une petite période d'attente est nécessaire, pendant que le 64 construit ses lutins. Ensuite l'écran haute-résolution est effacé, puis la RAM couleur est remise à blanc. Le crayon apparaît au centre de l'écran et se déplace avec les touches "CRSR". Les touches "CLR" et "HOME" fonctionnent également. La fonction "CLR" efface l'écran mais commme il s'agit d'un programme BASIC, le temps d'exécution est de l'ordre de 60 secondes. Les touches de fonction modifient le type de curseur:

F1 : crayon avec pointe
F2 : crayon sans pointe
F3 : gomme
F4 : curseur invisible.

Le dessin est en noir et blanc avec une résolution de 320*200.

```
0 REM PROGRAMME DE DESSIN POUR CBM-64
1 REM
2 REM ECRAN DE X POINTS HORIZONTAUX
3 REM       ET Y POINTS VERTICAUX
4 :
5 REM *********************************
6 REM *                               *
7 REM * HAUTE RESOLUTION MONOCHROME   *
8 REM *                               *
9 REM *********************************
10 :
11 :
12 :
13 :
20 PRINT"[CLR]UN INSTANT, S.V.P.":    REM  EFFACE L'ECRAN
30 XM=320:YM=200
50 CO=1024
60 GOSUB 1000
65 POKE 53265,PEEK(53265)OR32:REM MODE BIT MAP
67 HO=8192:POKE 53272,PEEK(53272)OR 8  :REM ADRESSE BIT MAP
70 FOR C= 0 TO 7999:POKE HO+C,0:NEXTC:REM EFFACE L'ECRAN
80 FOR C= 0 TO 999:POKECO+C,1:NEXTC:REM NOIR SUR BLANC
90 X = XM/2:Y = YM/2:BIC = 1
94 :
95 A$=" ":GOTO 120
99 REM*********************************
100 REM BOUCLE PRINCIPALE
110 GET A$
111 IFA$=""GOTO110
120 IF A$="[HOME]"THEN X=0:Y=0  :REM CURSEUR HOME
130 IF A$="[CLR]"GOTO 70        :REM EFFACE ECRAN
140 IF A$="[C.GAUCHE]"THEN X=X-1     :REM CURSEUR A GAUCHE
150 IF A$="[C.DROITE]"THEN X=X+1     :REM CURSEUR A DROITE
160 IF A$="[C.HAUT]"THEN Y=Y-1       :REM CURSEUR EN HAUT
170 IF A$="[C.BAS]"THEN Y=Y+1     :REM CURSEUR EN BAS
180 IF A$=CHR$(133)THEN BIC=1 :REM F1 SORT LA PLUME
185 IF A$=CHR$(134)THEN BIC=2 :REM F3 RENTRE LA PLUME
187 IF A$=CHR$(135)THEN BIC=3 :REM F5 GOMME
188 IF A$=CHR$(136)THEN BIC=0 :REM F7 RIEN DU Td
200 IF X=XM THEN X=XM-1
210 IF X<0  THEN X=0
220 IF Y=YM THEN Y=YM-1
230 IF Y<0  THEN Y=0
250 R=INT(Y/8):C=INT(X/8):L=Y AND 7:B=7-(X AND 7):BY=HO+R*320+C*8+L
```

```
260 GOSUB 2000
300 :
370 ON (BIC+1) GOSUB 3000,4000,5000,6000
380 IF BIC=1 THEN GOSUB 7000 :REM ECRIT
390 IF BIC=3 THEN GOSUB 8000 :REM EFFACE
990 GOTO100
999 ************************************
1000 REM CONFIGURE LES 3 SPRITES
1020 VI=53248
1030 PL=832:REM SPRITE-BLOCS NUMEROS 13,14 ET 15
1125 FOR H=0TO2
1130 FOR I=0TO20:READ A$:FOR K=0TO2:T=0:FOR J=0TO7:B=0
1131 IFMID$(A$,J+K*8+1,1)="Q"THENB=1
1135 T=T*2+B:NEXTJ:PRINT"*";:POKE832+64*H+I*3+K,T:NEXTK,I,H
1140 POKEVI+29,0:REM EXPANSION HORIZONTALE
1150 POKEVI+23,0:REM EXPANSION VERTICALE
1160 POKEVI+28,1:REM MODE MULTICOLORE
1170 POKEVI+37,0:REM COULEUR 1 = NOIR
1180 POKEVI+38,4:REM COULEUR 2 = MAUVE
1190 POKEVI+39,9:REM COULEUR SPRITE = BLANC
1990 RETURN
1999 ************************************
2000 REM POSITIONNE LE SPRITE 0
2010 XA=XX:XX=23 + X
2020 YA=YY:YY=46 + Y
2030 IFXX<256THENPOKEVI+16,0:GOTO 2100
2040 XX=XX-256:POKEVI+16,1
2100 FORXB=XATOXXSTEPSGN(XX-XA):POKE VI,XB:NEXT:REM POS EN X
2200 FORYB=YATOYYSTEPSGN(YY-YA):POKE VI+1,YB:NEXT:REM POS EN Y
2990 RETURN
2999 ************************************
3000 REM SPRITE0 INVISIBLE
3990 POKEVI+21,0:RETURN:REM SPRITE INVISIBLE
3999 ************************************
4000 REM SPRITE0 = CRAYON ET POINTE
4010 POKE 2040,13:REM ADRESSE SPRITE0
4990 POKE VI+21,1:RETURN:REM SPRITE VISIBLE
4999 ************************************
5000 REM SPRITE0 = CRAYON SANS POINTE
5010 POKE2040,14:REM ADRESSE SPRITE0
5990 POKEVI+21,1:RETURN:REM SPRITE VISIBLE
5999 ************************************
6000 REM SPRITE0 = GOMME
6010 POKE2040,15:REM ADRESSE SPRITE0
6990 POKEVI+21,1:RETURN:REM SPRITE VISIBLE
6999 ************************************
7000 REM ECRIT UN POINT
7020 POKE BY,PEEK(BY)OR 2^B
7900 RETURN
7999 ************************************
8000 REM EFFACE
8020 POKE BY,PEEK(BY)AND(255-2^B)
8900 RETURN
8999 ************************************
9999 ************************************
10000 DATA"                        "
10010 DATA"                        "
10020 DATA"                        "
10030 DATA"    Q QQ                "
10040 DATA"    Q QQ Q              "
10050 DATA"    QQ Q Q Q            "
10060 DATA"  Q Q Q Q Q Q            "
10070 DATA"    Q Q Q Q Q Q Q       "
10080 DATA"      Q Q Q Q Q Q Q     "
10090 DATA"        Q Q Q Q Q Q Q   "
10100 DATA"          Q Q Q Q Q Q Q "
10110 DATA"            Q Q Q Q Q   "
10120 DATA"              Q Q Q Q   "
10130 DATA"                Q Q     "
10140 DATA"                        "
10150 DATA"                        "
10160 DATA"                        "
10170 DATA"                        "
```

```
10180 DATA"                          "
10190 DATA"                          "
10200 DATA"                          "
15000 DATA"                          "
15010 DATA"                          "
15020 DATA"                          "
15030 DATA"  Q Q Q                   "
15040 DATA"  Q Q Q Q                 "
15050 DATA"  Q Q Q Q Q               "
15060 DATA"  Q Q Q Q Q Q             "
15070 DATA"    Q Q Q Q Q Q           "
15080 DATA"      Q Q Q Q Q Q         "
15090 DATA"        Q Q Q Q Q Q       "
15100 DATA"          Q Q Q Q Q Q     "
15110 DATA"            Q Q Q Q Q     "
15120 DATA"              Q Q Q Q     "
15130 DATA"                Q Q       "
15140 DATA"                          "
15150 DATA"                          "
15160 DATA"                          "
15170 DATA"                          "
15180 DATA"                          "
15190 DATA"                          "
15200 DATA"                          "
20000 DATA"                          "
20010 DATA"                          "
20020 DATA"                          "
20030 DATA"  Q Q Q Q Q               "
20040 DATA"  Q QQQQQQQ Q             "
20050 DATA"  QQQ QQQQQQQ Q           "
20060 DATA"  QQQQQ QQQQQQQ Q         "
20070 DATA"    QQQQQ QQQQQQQ Q       "
20080 DATA"      QQQQQ QQQQQQQ Q     "
20090 DATA"        QQQQQ Q Q Q Q Q"
20100 DATA"          QQQ QQQQQQQ Q"
20110 DATA"            Q QQQQQQQ Q"
20120 DATA"              Q Q Q Q Q"
20130 DATA"                          "
20140 DATA"                          "
20150 DATA"                          "
20160 DATA"                          "
20170 DATA"                          "
20180 DATA"                          "
20190 DATA"                          "
20200 DATA"                          "
55554 END
55555 SAVE"@0:DESSIN BITMAP",8

READY.
```

DESSIN MOYRES

Au chargement, un petit délai est nécessaire pour la construction des lutins. Ensuite le lutin-crayon apparaît au centre de l'écran. Il trace en noir et blanc avec une résolution de 50*80 points en mode graphique 1, en utilisant des quarts de caractères.

Les touches 'CRSR' sont fonctionnelles ainsi que "CLR" et "HOME", bien que cette dernière ramène le curseur au centre de l'écran. Les touches de fonction modifient le type de curseur.

F1 : crayon avec pointe
F2 : cayon sans pointe
F3 : gomme
F7 : curseur invisible.

Noter la façon plus tassée et plus rapide de codage des lutins par rapport au programme "DESSIN".

```
6 REM ************************************
7 REM * DESSIN POUR COMMODORE 64 EN    *
8 REM * MOYENNE RESOLUTION MONOCHROME  *
9 REM ************************************
20 PRINT"[CLR]      UN INSTANT, S.V.P.":    REM  EFFACE L'ECRAN
30 XMAX = 80:YMAX = 50
40 HOME = 1024
50 CO   = 55296
51 DIMA(15):FORI=0TO15:READA(I):NEXT
52 DIMAA(255):FORI=0TO15:AA(A(I))=I:NEXT
60 PRINT"[CLR]":FOR I=CO TO CO+1024
70 POKE I,0:NEXT I:  REM REMPLIT LA RAM COULEUR
80 GOSUB 1000
85 X=40:Y=25:BIC=1
92 DATA32,126,124,226,123,97,255,236,108,127,225,251,98,252,254,160
95 A$=" ":GOTO 120
99 REM************************************
100 REM BOUCLE PRINCIPALE
110 GET A$
120 IF A$="[HOME]"THEN X=40:Y=25:REM CURSEUR HOME
130 IF A$="[CLR]"GOTO 70        :REM EFFACE ECRAN
140 IF A$="[C.GAUCHE]"THEN X=X-1     :REM CURSEUR A GAUCHE
150 IF A$="[C.DROITE]"THEN X=X+1     :REM CURSEUR A DROITE
160 IF A$="[C.HAUT]"THEN Y=Y-1     :REM CURSEUR EN HAUT
170 IF A$ = "[C.BAS]" THEN Y = Y+1        :REM      CURSEUR EN BAS
180 IF A$ = CHR$(133)THEN BIC= 1   :REM F1 SORT LA PLUME
185 IF A$ = CHR$(134)THEN BIC= 2   :REM F3 RENTRE LA PLUME
187 IF A$ = CHR$(135)THEN BIC= 3   :REM F5 GOMME
188 IF A$ = CHR$(136)THEN BIC= 0   :REM F7 RIEN DU TOUT
200 IF X=XMAX THEN X=XMAX-1
210 IF X<0    THEN X=0
220 IF Y=YMAX THEN Y=YMAX-1
230 IF Y<0    THEN Y=0
245 S=INT(X/2):T=INT(Y/2)
250 CU=HOME+XMAX/2*T+S:CL=AA(PEEK(CU))
260 GOSUB 2000
360 IF A$=""GOTO 100
370 ON(BIC+1)GOSUB 3000,4000,5000,6000
380 IF BIC=1 THEN GOSUB 7000 :REM ECRIT
390 IF BIC=3 THEN GOSUB 8000 :REM EFFACE
990 GOTO 100
999 ************************************
1000 REM CONFIGURE SPRITES 13,14 ET 15
1020 VIDEO=53248
1030 PLUME=832
1130 FORH=832TO1022:READA:POKEH,A:NEXTH
1140 POKEVIDEO+29,1:REM EXPANS.HORIZ.
1150 POKEVIDEO+23,1:REM EXPANS.VERTIC.
1160 POKEVIDEO+28,1:REM MULTICOLORE
1170 POKEVIDEO+37,0:REM COULEUR 1=NOIR
1180 POKEVIDEO+38,4:REM COULEUR 2=MAUVE
1190 POKEVIDEO+39,1:REM SPRITE = BLANC
1990 RETURN
1999 ************************************
2000 REM POSITIONNE LE SPRITE 0
2010 XA=XX:XX=21 + X*4
2020 YA=YY:YY=45 + Y*4
2030 IFXX<256THENPOKEVIDEO+16,0:GOTO 2100
2040 XX=XX-256:POKEVIDEO+16,1
2100 POKE VIDEO,XX:REM POS EN X
```

```
2200 POKE VIDEO + 1,YY:REM POS EN Y
2990 RETURN
2999 **********************************
3000 REM SPRITE0 INVISIBLE
3990 POKEVIDEO+21,0:RETURN
3999 **********************************
4000 REM SPRITE0 = CRAYON ET POINTE
4010 POKE 2040,13:REM ADRESSE SPRITE0
4990 POKE VIDEO + 21,1:RETURN:REM SPRITE VISIBLE
4999 **********************************
5000 REM SPRITE0 = CRAYON SANS POINTE
5010 POKE2040,14:REM ADRESSE SPRITE0
5990 POKEVIDEO+21,1:RETURN:REM SPRITE VISIBLE
5999 **********************************
6000 REM SPRITE0 = GOMME
6010 POKE2040,15:REM ADRESSE SPRITE0
6990 POKEVIDEO+21,1:RETURN:REM SPRITE VISIBLE
6999 **********************************
7000 REM ECRIT
7010 U=X-2*S:V=Y-2*T:CH=-(U+1)*((V=0)+4*(V=1)):REM CH=1,2,4 OU 8 :NUM.
     DU CARRE
7020 POKECU,A(CHORCL)
7900 RETURN
7999 **********************************
8000 REM EFFACE
8010 U=X-2*S:V=Y-2*T:CH=-(U+1)*((V=0)+4*(V=1)):REM CH=1,2,4 OU 8 :NUM.
     DU CARRE
8020 IFCLANDCHTHENPOKE CU,A(CLAND(15-CH))
8900 RETURN
8999 **************************************
10000 DATA0,0,0,0,0,0,0,0,22,0,0,22
10010 DATA128,0,26,160,0,42,168,0,10,170
10020 DATA0,2,170,128,0,170,160,0,42
10030 DATA168,0,10,168,0,2,168,0,0,160
10040 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10050 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10060 DATA0,42,0,0,42,128,0,42,160,0,42
10070 DATA168,0,10,170,0,2,170,128,0,170
10080 DATA160,0,42,168,0,10,168,0,2,168,0
10090 DATA0,160,0,0,0,0,0,0,0,0,0,0,0,0,0
10100 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10110 DATA0,21,80,0,23,244,0,29,253
10120 DATA0,31,127,64,7,223,208,1,247
10130 DATA244,0,125,85,0,29,253,0,5,253
10140 DATA0,1,85,0,0,0,0,0,0,0,0,0,0,0,0
10150 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
55555 SAVE"@0:DESSIN MOYRES",8

READY.
```

DESSIN MULTI-MOY

Programme de dessin en mode caractères multicolores avec joystick en résolution moyenne 40 * 80 points. Les 256 caractères programmables sont utilisés en mode multicolore qui permet des dessins en 4 couleurs sans aucune erreur. Les 4 couleurs sont modifiables en ligne 40.

La sélection de la couleur de tracé se fait avec le bouton de tir. L'écran peut s'effacer avec la touche 'CLR' ou 'HOME'. Notez l'utilisation de 6 boucles FOR-NEXT imbriquées pour programmer les 256 caractères.

```
5 REM DESSIN 50*80 EN 4 COULEURS
6 :
10 POKE 51,0: POKE 52,56
11 POKE 55,0: POKE 56,56: CLR
20 I=0: J=0: K=0: L=0: M=0: N=0
30 EC= 1024: CAR=14336: CO=55296: VIC=53248
40 DIMX(10),Y(10),F(3),P(3)
41 F(0)=1: F(1)=2: F(2)=6: F(3)=5:REM CHOIX DES COULEURS
42 P(0)=4: P(1)=64: P(2)=1: P(3)=16
50 DATA1,1,1,-1,1,0,0,0,-1,1,-1,-1,-1,0
51 DATA0,0,0,1,0,-1,0,0
60 FOR I=0 TO 10: READ X(I),Y(I): NEXTI
70 GOTO 1000
99 :
100 FOR I=CO TO CO+999: POKE I, F(3)+8: NEXT: RETURN
199 :
200 FOR I=EC TO EC+999: POKE I,0: NEXT
210 X=0: Y=0: RETURN
299 :
300 GET A$
310 IF A$="" THEN A=0
320 IF A$<>"" THEN A=ASC(A$)
330 RETURN
399 :
400 DIR=PEEK(56320)
410 IF DIR=111 THEN 480
430 DIR=DIR-117
440 IF DIR>=0 THEN DX=X(DIR): DY=Y(DIR)
460 GOTO 510
480 CL=CL+1: IF CL>3 THEN CL=0
500 FOR K=0 TO 200: NEXT K
510 RETURN
999 :
1000 FOR I=0 TO 3: FOR J=0 TO 3: FOR K=0 TO 3
1030 PRINT"*";
1040 FOR L=0 TO 3
1050 C1=L + 4*K + 16*J + 64*I
1060 FOR M=0 TO 1
1070 IF M=0 THEN PK=K*5*16 + L*5
1080 IF M=1 THEN PK=I*5*16 + J*5
1090 FOR N=0 TO 3
1100 POKE CAR + 8*C1 + 4*M + N, PK
1110 NEXT N,M,L,K,J,I
1120 POKE VIC+24, (PEEK(VIC+24) AND 240) OR 14
1130 POKE VIC+22, PEEK(VIC+22) OR 16
1140 FOR I=0 TO 2: POKE VIC+33+I,F(I): NEXT I
1150 GOSUB 100: GOSUB 200
3000 CL=0
3010 GOSUB 300: REM BOUCLE PRINCIPALE
3020 GOSUB 400
3025 POKE VIC+32,F(CL)
3040 IF A=19 THEN GOSUB 200
3050 X=X+DX: Y=Y+DY
3060 IF X<0  THEN X=0
3070 IF X>79 THEN X=79
3080 IF Y<0  THEN Y=0
3090 IF Y>49 THEN Y=49
3100 C1=EC + INT(Y/2)*40 + INT(X/2)
3110 PO=P(2*(X-INT(X/2)*2) + Y-INT(Y/2)*2)
3120 PK=PEEK(C1)
3130 PK= PK AND (255-3*PO)
3140 PK= PK OR (CL*PO)
3150 POKE C1,PK
3160 GOTO 3010
55555 SAVE"@0:DESSIN MULTI-MOY",8

READY.
```

DIR.

Le programme "DIR" affiche le nom du disque et les noms des fichiers situés sur celui-ci à raison de 44 programmes au maximum. Ensuite, comme l'indique la ligne du bas d'écran, le 64 attend la frappe d'une touche de fonction :

F1 affiche le message d'erreur éventuel du disque.
attend ensuite une frappe sur F1 pour revenir au mode normal.

F3 charge un programme en adresse absolue.

F5 efface un programme du disque.

F7 charge un programme à l'adresse de début du BASIC.

Pour ces 3 dernières fonctions, la question "quel numéro" apparaît dans la ligne au bas de l'écran et la fonction est effectuée dès réponse par un numéro compris entre 1 et 44 et désignant le programme.

```
0 REM TABLE DES MATIERES DISQUES AVEC CHARGEMENT AUTOMATIQUE
1 IFF=1 GOTO 1800
6 POKE53280,7:POKE53281,5
10 OPEN2,8,15:OPEN1,8,0,"$0":GET#1,A$,A$
14 DIM N$(44):S$="
15 Q$="[HOME][RVS OFF][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS]
     [C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C
     .BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS]  [RVS ON]"
16 SS$="                                          [RVS OFF]  "
17 SS$=SS$+"  [RVS ON]                                        [RVS OFF]"
18 M$="    F1      F3      F5      F7    [RVS OFF]  "
19 M$=M$+"  [RVS ON]  STATUS   LOAD,1   EFFACE   LOAD   "
20 GOSUB2000:PRINT"[CLR] [BLANC][RVS ON]"N$(0)S$" "
30 FORH=1TO44
31 GOSUB2000:IFST<>0GOTO1000
33 IFC$<>"R" THEN N$(H)="":GOTO 31
34 H$=STR$(H):H$="[BLEU]"+RIGHT$(" 0"+RIGHT$(H$,LEN(H$)-1),2)+"[BLANC]
     "
35 PRINTH$;LEFT$(N$(H)+S$,17);:NEXTH
1000 H=H-1
1030 PRINTQ$;M$;:GOSUB4000
1050 IFASC(A$)<133THEN1030
1060 ONASC(A$)-132GOTO1100,1300,1500,1700
1100 PRINTQ$;SS$;Q$;"DISQUE :  ";:REM ------STATUS
1110 GET#2,A$:IFA$<>CHR$(13)THEN PRINTA$;:GOTO 1110
1199 GOSUB4000:GOTO1030
1300 PRINTQ$SS$Q$"QUEL NUMERO ?";:GOSUB3000:IFA=0 GOTO 1030:REM LOAD AB
     SOLU
1320 POKE53280,254:POKE53281,246
1330 PRINT"[RVS OFF][CLR][CYAN]LOAD"CHR$(34);N$(A);CHR$(34);",8,1[HOME]
     ";
1340 POKE631,19:POKE632,13:POKE198,2:CLOSE1CLOSE2:END
1500 PRINTQ$SS$Q$"QUEL NUMERO ?";:GOSUB3000:IFA=0 GOTO 1030:REM EFFACE

1520 PRINTQ$SS$Q$"ON EFFACE [RVS OFF]"N$(A)"[RVS ON] (O/N) ?";
1525 GOSUB4000:IFA$<>"O"GOTO 1030
1530 PRINT#2,"S0:";N$(A):RUN
1700 PRINTQ$SS$Q$"QUEL NUMERO ?";:GOSUB3000:IFA=0 GOTO 1030:REM LOAD NO
     RMAL
1710 POKE53280,254:POKE53281,246
1720 PRINT"[RVS OFF][CLR][CYAN]LOAD"CHR$(34);N$(A);CHR$(34);",8"
```

```
1730 POKE631,19:POKE632,13:POKE198,2:CLOSE1:CLOSE2:END
1800 PRINT"[CLR]ADRESSE  DE FIN ="256*PEEK(46)+PEEK(45):END
2000 GET#1,A$,A$,A$,B$:REM LIT UNE LIGNE
2090 GET#1,B$:IF ST<>0 GOTO1000:REM FIN
2100 IF B$<>CHR$(34) GOTO 2090
2110 GET#1,B$:IF B$<>CHR$(34)THEN N$(H)=N$(H)+B$:GOTO 2110
2120 GET#1,C$:IF C$=CHR$(32) THEN 2120
2140 GET#1,C$:IFST<>0 GOTO 1000
2160 GET#1,B$:IF B$<>"" GOTO 2160
2999 RETURN
3000 GOSUB4000:B$=A$:GOSUB4000:A=VAL(B$+A$):RETURN:REM GET 2 CHIFFRES
4000 POKE198,0:WAIT198,1:GETA$:PRINTA$;:RETURN:REM GET 1 CARACTERE
55555 SAVE"@0:DIR",8

READY.
```

ENVELOPPES.

Programme de démonstration des enveloppes sonores complexes du SID. Les enveloppes complexes sont obtenues en modifiant pendant la durée de la note la valeur du bit DE/AR de déclenchement/arrêt de l'enveloppe. (trémolo, etc...)

```
10 REM ENVELOPPES COMPLEXES
11 :
20 S=54272:GOSUB500:VL=15:AR=16:DE=AR+1
80 AD=8*16+6:SR=12*16+6:POKE S+24,VL
90 PRINT"1 ENVELOPPE NORMALE
91 PRINT"2 ENVELOPPE SPECIALE CROISSANTE"
92 PRINT"3 ENVELOPPE SPECIALE DECROISSANTE"
93 PRINT"4 ENVELOPPE SPECIALE TREMOLO":PRINT
94 INPUT"1 ,2 ,3 ,4 OU 0=FIN";A
95 IF A=0 THEN GOSUB 500:END
110 RESTORE:POKE S+5,AD:POKE S+6,SR
120 READ N,D:IF N=0 THEN GOSUB500:RUN
125 POKE S+1,N
130 ON A GOSUB 600,700,800,900
140 FOR TD=0TO40:NEXT:GOTO120
200 DATA25,4,28,4,25,4,25,4,25,2,28,2
210 DATA32,12,25,4,28,4,19,4,19,2,19,2
230 DATA21,1,24,1,25,4,24,2,19,4,0,0
500 FORT=S TO S+24:POKET,0:NEXT:RETURN
599 :
600 REM ENVELOPPE NORMALE
610 POKE S+4,DE:FOR TD=0TO50*D:NEXT
698 POKE S+4,AR:RETURN
699 :
700 REM ENVELOPPE SPECIALE 1
710 FOR T=0TO 7:POKES+4,AR
715 FOR TD=T TO 0STEP-1:NEXT TD
720 POKES+4,DE
725 FOR TD=T TO 0STEP-1:NEXT TD,T
798 POKE S+4,AR:RETURN
799 :
800 REM ENVELOPPE SPECIALE 2
810 FOR T=0TO 7:POKES+4,AR
815 FOR TD=0 TO T:NEXT TD
820 POKES+4,DE
825 FOR TD=0 TO T:NEXT TD,T
898 POKE S+4,AR:RETURN
899 :
900 REM ENVELOPPE SPECIALE 3
910 FOR T=0TO 5:POKES+4,AR
915 FOR TD=0 TO 2*T:NEXT TD
```

```
920 POKES+4,DE
925 FOR TD=2*T TO 0STEP-1:NEXT TD,T
998 POKE S+4,AR:RETURN
999 :
55555 SAVE"@0:ENVELOPPES",8

READY.
```

EXTRATERRESTRE

Programme de démonstration sonore où l'on montre les effets spéciaux obtenus par un balayage cyclique de la fréquence du filtre passe-bande (effet haut-parleur tournant).

```
10 REM — CONCERT EXTRATERRESTRE —
15 REM BALAYAGE DE FREQUENCE DU FILTRE PASSE-BANDE
20 :
30 PRINT"[CLR]POUSSER ESPACE"
70 S=54272:GOSUB500:AR=32:DE=AR+1
80 AD=0:SR=15*16+0     :REM ADSR VOIX1
90 POKE S+24,15+128+32:REM SUPP VOIX3,PASSE-BANDE
100 POKE S+23,15*16+1 :REM FILTRE VOIX 1
110 POKE S+5,AD:POKE S+6,SR
120 POKE S+14,11       :REM FREQ.BALAYAGE
121 POKE S+18,16       :REM TRIANGLE VOIX 3
125 POKES+4,DE         :REM DEBUT SON
130 GOSUB600           :REM BALAYAGE
140 POKES+4,AR         :REM FIN DE SON
160 GOSUB500:END
499 :
500 FOR T=S TOS+24:POKE T,0:NEXT:RETURN
599 :
600 REM DELAI ET BALAYAGE FILTRE
610 POKES+22,(PEEK(S+27)+60)AND255:REM FREQ.FILTRE
620 POKE S+1,106ANDPEEK(S+27)      :REM FREQ. VOIX 1
630 GETA$:IFA$="" GOTO 610
699 RETURN
55555 SAVE"@0:EXTRATERRESTRE",8

READY.
```

FIGURES.

(Dessin avec fenêtre sur 80 colonnes)

Ce programme est à l'image ce que le traitement de texte est au texte. Il permet de créer, modifier, sauver, rappeler et imprimer des figures utilisant les caractères majuscules et graphiques du 64. Ce programme a été utilisé pour générer certaines figures de ce livre. La taille maximum du dessin est de 79*40 caractères. Les fonctions de décalage (scrolling) horizontal et vertical sont réalisées en langage machine.

Au démarrage, la question "Récupération ou normal" permet de récupérer une image éventuellement toujours en mémoire. La première fois, il faut toujours répondre "N".

Le curseur est un petit "T" dont le pied indique la direction du prochain déplacement. Cette direction peut être modifiée par les touches "CRSR". Tous les caractères peuvent être employés pour dessiner. La position X,Y du curseur est en permanence affichée dans le coin supérieur droit de l'écran.

La touche "HOME" ramène le curseur au coin supérieur gauche de l'écran (un morceau de la figure) tandis que "CLR" le ramène au coin supérieur gauche de la figure. Le curseur se déplace entre 0 et 78 en horizontal, entre 0 et 39 en vertical. Une série de 16 fonctions spéciales est disponible en frappant deux caractères dont le premier est la "flèche à gauche". Si vous ne connaissez pas par coeur la commande à effectuer, un écran d'aide (HELP) apparaît deux secondes après la frappe de la flèche à gauche. Les fonctions A et B définissent une zone qui peut ensuite être encadrée, inversée, efacée par les fonctions C à G. Les autres fonctions concernent les déplacements rapides de curseur et les effacements et insertions de lignes et de colonnes.

La touche de fonction "F5" sauve l'image sur le disque. La fonction peut être annulée par un "D".

La touche de fonction "F6" charge une image du disque. En cas d'erreur disque, taper simplement sur "RETURN" lance la comande 'GO TO 10' qui reprend le programme en cours.

La touche "F7" permet la recopie de l'écran sur une imprimante COMMODORE MPS801 ou 1526. (Les autres imprimantes n'ont pas été testées avec ce programme). Une question est posée "numéro de figure ?". Ce numéro sera repris en bas du dessin sur l'imprimante.

```
 3 REM TRAITEMENT D'IMAGES EN
 4 REM CARACTERES GRAPHIQUES COMMODORE
 5 REM============================================================
10 IFC=0 GOTO 1000
15 REM ************************************ RECOPIE D'ECRAN
16 PT=A(V)+U:P2=INT(PT/256):P1=PT-P2*256
17 POKE190,P1:POKE191,P2:SYS28928:SYS29184
18 POKE45,PEEK(29000):POKE46,PEEK(29001)
20 REM ************************************AFFICHE X,Y CURSEUR
21 C1=X+U:C2=INT(C1/10):POKE1059,176+C2
22 C2=C1-10*C2:POKE1060,176+C2
23 C1=Y+V:C2=INT(C1/10):POKE1062,176+C2
24 C2=C1-10*C2:POKE1063,176+C2
25 REM ********************** LIT LE CARACTERE SOUS CURSEUR
26 P=1024+X+40*Y:B=PEEK(P):POKE54272+P,0
27 POKEP,Z(Z)+R:GETA$:IFA$<>""GOTO 31
29 POKEP,B:GETA$:IFA$=""GOTO 27:REM **** LECTURE DU CLAVIER
30 POKEP,Z(Z)+R
31 A=ASC(A$)
32 IF D(A)=0 THENPOKEA(V+Y)+U+X,R+B(A):GOTO400
40 ONO(A)GOTO50,51,52,53,54,55,56,60,70,720,730
42 ONO(A)-11GOTO 740,750,760,3000,4000,5000,2000,2500,6000
50 U=0:V=0                    :REM CLR
51 X=0:Y=0          :GOTO10  :REM HOME
52 XX= 0:YY=+1:Z=4:GOTO400:REM CRSR BAS
53 XX= 0:YY=-1:Z=3:GOTO400:REM CRSR HAUT
54 XX=+1:YY= 0:Z=2:GOTO400:REM CRSR DROITE
```

```
55 XX=-1:YY= 0:Z=1:GOTO400:REM CRSR GAUCHE
56 X=-1:U=0:Y=Y+1:YY=0:XX=1:Z=2:R=0
57 IFY>24THENY=24:V=V+1:IFV>15THENV=15
58 GOTO 420:REM   RETURN
60 R=128:GOTO420:           REM RVS
70 R=  0:GOTO420:           REM OFF
399 REM ******************************** MET A JOUR X,Y,U,V
400 X=X+XX:Y=Y+YY
420 IFX<0THENX=0:U=U-1:IFU<0THENU=0
430 IFY<0THENY=0:V=V-1:IFV<0THENV=0
440 IFX>39THENX=39:U=U+1:IFU>39THENU=39
450 IFY>24THENY=24:V=V+1:IFV>15THENV=15
599 GOTO10:REM ******************** FIN DE BOUCLE PRINCIPALE
720 V=V+1:IFV>15THENV=15:REM SCROLL EN BAS
721 Y=Y-1:IFY<0THENY=0
729 GOTO 10
730 U=U-1:IFU<0THENU=0:  REM SCROLL A DROITE
731 X=X+1:IFX>39THENX=39
739 GOTO 10
740 V=V-1:IFV<0THENV=0:  REM SCROLL EN HAUT
741 Y=Y+1:IFY>24THENY=24
749 GOTO 10
750 U=U+1:IFU>39THENU=39:REM SCROLL A GAUCHE
751 X=X-1:IFX<0THENX=0
759 GOTO 10
760 PRINT"[CLR]EFFACEMENT[C.BAS][C.BAS]SUR (O/N)";:INPUTA$:IFA$="O"THEN
    C=0
999 GOTO10
1000 REM ******************** INIT TEXTE & MEMOIRE EN $7000
1010 POKE55,0:POKE56,98:CLR:C=1
1030 DIM A(39),B(255),D(255),H$(255):A(0)=28512
1050 FORI=1TO39:A(I)=A(I-1)-80:NEXTI
1080 POKE53280,15:POKE53281,6:PRINT"[CLR][JAUNE L.]";:XX=1:YY=0
1100 POKE649,2:POKE650,255:REM TOUCHES A REPETITION
1200 CS=0:FORI=28928TO28983:READA:POKEI,A:CS=CS+A:NEXTI
1210 PRINT"# ";:IFCS<>7643THENSTOP
1220 CS=0:FORI=29184TO29274:READA:POKEI,A:CS=CS+A:NEXTI
1225 PRINT"# ";:IFCS<>11074THENSTOP
1230 PRINTCHR$(8);:POKE53272,21
1240 POKE29000,PEEK(45):POKE29001,PEEK(46):REM AD.FIN PRG
1255 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS]"
1260 PRINT"          TRAITEMENT D' IMAGES[C.BAS][C.BAS][C.BAS]"
1400 FORI=1TO20:READF:D(F)=I:NEXTI
1410 FORI=64TO96:B(I)=I-64:NEXT
1430 PRINT"          # ";:FORI=192TO255:B(I)=I-128:NEXT
1450 PRINT"# ";:FORI=97TO191:B(I)=I-64:NEXT
1470 PRINT"# ";:FORI=32TO63 :B(I)=I:NEXT:B(34)=44:PRINT"# ";
1490 Z(1)=115:Z(2)=107:Z(3)=113:Z(4)=114:Z=2
1500 R$=CHR$(18):RO$=CHR$(146):X1=0:X2=79:Y1=0:Y2=39
1505 N$=CHR$(15):N1$=CHR$(15):N2$=CHR$(8)
1510 FORI=0TO31:H$(I)=CHR$(I+64)
1520 H$(128+I)=R$+H$(I)+RO$:NEXT:PRINT"# ";
1530 FORI=32TO63:H$(I)=CHR$(I)
1540 H$(I+128)=R$+H$(I)+RO$:NEXT:PRINT"# ";
1550 FORI=64TO95:H$(I)=CHR$(I+32)
1560 H$(I+128)=R$+H$(I)+RO$:NEXT:PRINT"# ";
1570 FORI=96TO127:H$(I)=CHR$(I+64)
1580 H$(I+128)=R$+H$(I)+RO$:NEXT:PRINT"# ";
1590 PRINT"[CLR][C.BAS][C.BAS][C.BAS][RVS ON]N[RVS OFF]ORMAL OU [RVS ON
    ]R[RVS OFF]ECUPERATION";
1600 INPUT A$:IFA$="R"GOTO10
1999 SYS28951:GOTO 10:REM ************ EFFACE MEMOIRE TEXTE
2000 FOR I=X+U TO78:POKEA(Y+V)+I,PEEK(A(Y+V)+I+1):NEXT
2499 POKEA(Y+V)+79,32:GOTO 10
2500 IFPEEK(A(Y+V)+78)<>32 GOTO 10
2520 FORI=78TOX+U+1STEP-1
2530 POKEA(Y+V)+I,PEEK(A(Y+V)+I-1):NEXT
2999 POKEA(Y+V)+X+U,32:GOTO 10
3000 PRINT"[CLR]SAVE[C.BAS][C.BAS]":REM ************************** SA
    VE TEXTE
3015 IF LEN(N$) THENPRINT"NOM ACTUEL = "N$
3017 PRINT"[C.BAS][C.BAS][RVS ON]D[RVS OFF] = RETOUR DESSIN, [RVS ON]RE
    TURN[RVS OFF] POUR LE NOM ACTUEL"
```

```
3020 M$="":INPUT "[C.BAS][C.BAS][C.BAS]NOM  ";M$
3025 IFM$="D" GOTO 10
3027 IF LEN(M$) THEN N$=M$:GOTO 3030
3029 IF LEN(N$)=0 GOTO 3020
3030 C=1:T$=CHR$(34)+"@0:FIG"+N$+CHR$(34)
3035 PRINT"[CLR][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS
     ][C.BAS][C.BAS][C.BAS][C.BAS][C.DROITE][C.DROITE][C.DROITE][C.DRO
     ITE][C.DROITE][C.DROITE][C.DROITE][C.DROITE][C.DROITE]PATIENCE ..
     ..."
3040 PRINT"[HOME]SAVE";T$;"[BLEU],8,1:POKE43,1:POKE44,8:GOTO3100"
3050 POKE43,48:POKE44,99:POKE45,176:POKE46,111:REM DEBUT&FIN
3060 POKE198,2:POKE631,19:POKE632,13:REM TAMPON CLAVIER
3070 END
3100 POKE43,1:POKE44,8
3110 POKE45,PEEK(29000):POKE46,PEEK(29001)
3999 PRINT"[CLR][JAUNE L.]":GOTO 10
4000 PRINT"[CLR][BLANC]LOAD[C.BAS][C.BAS]":REM ************************
     LOAD TEXTE
4020 N$="":INPUT"NUMERO DE FIGURE";N$
4025 IF LEN(N$)=0 GOTO 10
4030 T$="0:FIG"+N$:PRINT"[CLR][BLANC]LOAD"
4060 PRINT"[C.BAS][C.BAS]EN CAS D'ERREUR, TAPER GOTO 10"
4070 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS]GOTO 10
4100 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS]          PATIENCE ..."
4105 PRINT"[HOME][C.BAS][C.BAS][C.BAS][C.BAS]";
4110 LOAD T$,8,1
4999 END
5000 PRINT"[CLR]PRINT":REM ************************ PRINT TEXTE
5020 PRINT"[C.BAS]ETES-VOUS AU SOMMET DE PAGE ?"
5050 GETA$:IFA$=""GOTO 5050
5060 IFA$<>"O"GOTO 10
5070 IF LEN(N$) GOTO 5100
5080 N$="0":INPUT"NUMERO DE FIGURE ";N$:GOTO 5070
5100 OPEN4,4
5110 T$="FIGURE N."+N$
5120 FOR I=0 TO 39:JJ=78:FOR J=78 TO 2 STEP-1
5140 IF PEEK(A(I)+J)=32 THEN JJ=JJ-1:NEXT J
5150 PRINT#4,N1$;
5230 FOR J=0 TO JJ:PRINT#4,H$(PEEK(A(I)+J));:NEXT J
5250 PRINT#4,N2$:NEXT:PRINT#4:PRINT#4:PRINT#4,N1$;T$
5300 FORI=0TO36:PRINT#4:NEXT I:CLOSE4
5999 GOTO 10
6000 REM ***************************** FONCTIONS SPECIALES
6005 FOR I=1TO70:GET A$:IF A$<>""GOTO6220
6006 NEXT I
6015 PRINT"[CLR]        [RVS ON]FONCTIONS SPECIALES  "
6020 PRINT"[C.BAS][C.BAS][C.BAS][RVS ON]A[RVS OFF]  COIN SUPERIEUR GAUC
     HE D'UN RECTANGLE
6030 PRINT"[RVS ON]B[RVS OFF]  COIN INFERIEUR DROIT  D'UN RECTANGLE
6040 PRINT"[RVS ON]C[RVS OFF]  EFFACER  LE RECTANGLE
6050 PRINT"[RVS ON]D[RVS OFF]  EFFACER  LE CONTENU DU RECTANGLE
6060 PRINT"[RVS ON]E[RVS OFF]  INVERSER LE RECTANGLE
6070 PRINT"[RVS ON]F[RVS OFF]  TRACER   LE RECTANGLE-COINS ARRONDIS
6080 PRINT"[RVS ON]G[RVS OFF]  TRACER   LE RECTANGLE-COINS VIFS
6090 PRINT"[RVS ON]H[RVS OFF]  CURSEUR A FOND A GAUCHE
6100 PRINT"[RVS ON]I[RVS OFF]  CURSEUR A FOND A DROITE
6110 PRINT"[RVS ON]J[RVS OFF]  CURSEUR A FOND EN HAUT
6120 PRINT"[RVS ON]K[RVS OFF]  CURSEUR A FOND EN BAS
6130 PRINT"[RVS ON]L[RVS OFF]  INSERER   UNE LIGNE
6140 PRINT"[RVS ON]M[RVS OFF]  SUPPRIMER UNE LIGNE
6150 PRINT"[RVS ON]N[RVS OFF]  INSERER   UNE COLONNE
6160 PRINT"[RVS ON]O[RVS OFF]  SUPPRIMER UNE COLONNE
6170 PRINT"[RVS ON]1..9[RVS OFF]  DEPLACE CURSEUR DE N POSITIONS
6200 PRINT"[C.BAS][RVS ON]_[RVS OFF]  [RVS ON]RETOUR AU DESSIN[RVS OFF]
     "
6210 GETA$:IFA$=""GOTO6210
6220 PRINTA$;:A=ASC(A$)-48
6225 IF(A<10)AND(A>0) GOTO 6800 :REM DEPL.CURSEUR
6230 A=A-16:IFA>15 GOTO 10
6240 ON A GOTO 6300,6330,6360,6390,6420,6450,6480,6510:REM A-H
6250 ON A-8 GOTO 6540,6570,6600,6630,6660,6690,6720:REM I-O
6300 REM *A* COIN SUP. GAUCHE
6310 X1=X+U:Y1=Y+V:GOTO10
```

```
6330 REM *B* COIN INF DROIT
6340 X2=X+U:Y2=Y+V:GOTO10
6351 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6352 IFA$<>" " GOTO 10
6360 REM *C* EFFACER RECTANGLE
6363 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6372 IFA$<>" " GOTO 10
6373 FORI=Y1TOY2:POKEA(I)+X1,32:POKEA(I)+X2,32:NEXT
6374 FORI=X1TOX2:POKEA(Y1)+I,32:POKEA(Y2)+I,32:NEXT
6376 GOTO 10
6390 REM *D* EFFACE CONTENU RECTANGLE
6393 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6395 IFA$<>" " GOTO 10
6400 FOR I=Y1+1TO Y2-1:FOR J=X1+1TO X2-1:POKE A(I)+J,32:NEXT J,I
6410 GOTO10
6420 REM *E* INVERSE LE CONTENU DU RECTANGLE
6421 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6422 IFA$<>" " GOTO 10
6430 FORI=Y1+1TOY2-1:FORJ=X1+1TO X2-1:A=A(I)+J
6435 B=PEEK(A):IFB>127 GOTO 6442
6440 POKEA,B+128:GOTO6445
6442 POKEA,B-128
6445 NEXT J,I:GOTO 10
6450 REM *F* TRACE RECTANGLE ARRONDI
6451 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6452 IFA$<>" " GOTO 10
6453 FORI=X1TOX2:POKEA(Y1)+I,64:POKEA(Y2)+I,64:NEXTI
6454 FORI=Y1TOY2:POKEA(I)+X1,93:POKEA(I)+X2,93::NEXTI
6455 POKEA(Y1)+X1,85
6456 POKEA(Y1)+X2,73
6457 POKEA(Y2)+X1,74
6458 POKEA(Y2)+X2,75
6459 GOTO 10
6480 REM *G* TRACE RECTANGLE VIF
6481 GOSUB 9000:REM AFFICHE COORD.RECTANGLE
6482 IFA$<>" " GOTO 10
6491 FORI=X1TOX2:POKEA(Y1)+I,64:POKEA(Y2)+I,64:NEXTI
6492 FORI=Y1TOY2:POKEA(I)+X1,93:POKEA(I)+X2,93:NEXTI
6493 POKEA(Y1)+X1,112
6494 POKEA(Y1)+X2,110
6495 POKEA(Y2)+X1,109
6496 POKEA(Y2)+X2,125
6497 GOTO 10
6510 REM *H* CURSEUR A GAUCHE
6520 U=0:X=0:GOTO 10
6540 REM *I* CURSEUR A DROITE
6560 U=39:X=39:GOTO 10
6570 REM *J* CURSEUR EN HAUT
6580 V=0:Y=0:GOTO 10
6600 REM *K* CURSEUR EN BAS
6610 V=15:Y=24:GOTO 10
6630 REM *L* INSERE LIGNE/DECALE EN BAS
6635 FOR J=0 TO 79:FORI=38 TO Y+V STEP-1
6640 POKEA(I+1)+J,PEEK(A(I)+J):NEXT I
6643 PRINT".";:POKEA(Y+V)+J,32:NEXT J
6645 GOTO 10
6660 REM *M* SUPPRIME LIGNE/DECALE EN HAUT
6665 FOR J=0TO 79:FOR I=Y+V TO 38
6670 POKE A(I)+J,PEEK(A(I+1)+J):NEXT I
6673 PRINT".";:POKE A(39)+J,32:NEXT J
6675 GOTO 10
6690 REM *N* INSERER COLONNE
6695 FOR J=0 TO 39:FORI=78 TO X+U STEP-1
6700 POKE A(J)+I+1,PEEK(A(J)+I):NEXT I
6703 PRINT".";:POKE A(J)+X+U,32:NEXT J
6705 GOTO 10
6720 REM *O* SUPPRIME COLONNE
6725 FOR J=0 TO 39:FOR I=X+U TO 78
6730 POKE A(J)+I,PEEK(A(J)+I+1):NEXT I
6733 PRINT".";:POKE A(J)+79,32:NEXT J
6735 GOTO 10
6800 REM *1-9* DEPLACEMENTS CURSEUR
6810 X=X+A*XX:Y=Y+A*YY
```

```
6820 IF X< 0 THEN U=U+X: X= 0: IF U< 0 THEN U=0
6830 IF Y< 0 THEN V=V+Y: Y= 0: IF V< 0 THEN V=0
6840 IF X>39 THEN U=U+X-39: X=39: IF U>39 THEN U=39
6850 IF Y>24 THEN V=V+Y-24: Y=24: IF V>15 THEN V=15
6999 GOTO 10
9000 REM AFFICHE COORD. RECTANGLE
9005 IFY2-Y1<3THENA$="TROP PETIT EN Y":GOTO9100
9006 IFX2-X1<3THENA$="TROP PETIT EN X":GOTO9100
9010 PRINT: PRINT"(X,Y)=("X1","Y1") -> ("X2","Y2")"
9020 PRINT"[RVS ON] [RVS OFF] SI OK,SINON [RVS ON]ESPACE[RVS OFF]"
9030 GETA$: IFA$=""GOTO9030
9040 RETURN
9100 A$="*":RETURN
10000 : LE SENS DU DEPLACEMENT EST DEFINI PAR
10020 :LES TOUCHES CRSR.LA DIRECTION EST
10030 :MEMORISEE JUSQU'AU CHANGEMENT SUIVANT
10040 :LE PROG. LANGAGE MACHINE RECOPIE LA
10050 :PORTION DE TEXTE IDOINE DANS LA ZONE ECRAN
10060 :ET REMET A JOUR LA RAM COULEUR
28928 DATA173,134,002,162,000,157,000,216
28936 DATA157,000,217,157,000,218,157,000
28944 DATA219,232,224,000,208,239,096,169
28952 DATA032,162,048,134,100,162,099,134
28960 DATA101,160,000,145,100,230,100,208
28968 DATA002,230,101,166,100,224,176,208
28976 DATA242,166,101,224,111,208,236,096
28977 REM CHECKSUM= 7643 DE 28928 A 28983
29184 DATA162,004,134,101,162,000,134,100
29192 DATA160,000,177,190,145,100,200,192
29200 DATA040,208,247,032,027,114,144,240
29208 DATA076,069,114,165,100,024,105,040
29216 DATA133,100,165,101,105,000,133,101
29224 DATA165,190,056,233,080,133,190,165
29232 DATA191,233,000,133,191,165,100,201
29240 DATA232,208,008,165,101,201,007,208
29248 DATA002,056,096,024,096,169,005,162
29249 REM CHECKSUM= 8443 DE 29184 A 29251
29256 DATA034,157,000,216,232,224,040,208
29264 DATA248,162,152,142,034,004,232,142
29272 DATA037,004,096
29273 REM CHECKSUM= 9113 DE 29200 A 29274
30000 DATA147,19,17,145,29,157,13,18,146,133,134
30010 DATA137,138,140,135,139,136,20,148,95
55555 SAVE"@0:FIGURES",8
```

READY.

FILTRES

Programme de démonstration des différents types de filtres du SID. Il y a la possibilité de faire un balayage cyclique de la fréquence du filtre. A ne pas écouter si vous êtes mélomane !!

```
10 REM ------ FILTRES ------
20 :
30 PRINT"NORMAL            (1)"
35 PRINT"FILTRE PASSE-BAS  (2)"
40 PRINT"FILTRE PASSE-HAUT (3)"
45 PRINT"FILTRE PASSE-BANDE(4)"
50 PRINT"FILTRE REJECTION  (5)"
55 PRINT"FILTRE SUIVEUR    (6)"
60 PRINT:INPUT "(1,2,3,4,5 OU 0=FIN) ";B
61 IF B=0 THEN END
65 PRINT:INPUT"BALAYAGE(O/N)";A$
66 F1=16:C=1:IFA$="O"THENC=2
70 S=54272:GOSUB500:VL=15:AR=16:DE=AR+1
80 AD=7*16+5:SR=12*16+3:POKE S+24,VL
90 ON B GOSUB 1100,1200,1300,1400,1500,1600,1100
110 RESTORE:POKE S+5,AD:POKE S+6,SR
120 READ N,D:IF N=0THEN GOSUB500:RUN
```

```
125 POKE S+1,N:POKES+4,DE
130 ON C GOSUB 600,700
140 POKES+4,AR
150 ON C GOSUB 600,700
160 GOTO120
200 DATA25,4,28,4,25,4,25,4,25,2,28,2
210 DATA32,12,25,4,28,4,19,4,19,2,19,2
230 DATA21,1,24,1,25,4,24,2,19,4,0,0
500 FOR T=S TO S+24:POKE T,0:NEXT:RETURN
600 FOR TD=0TO50*D:NEXT:RETURN
699 :
700 REM BALAYAGE FREQUENCE DE FILTRAGE
710 FOR TD=0TO20
720 IFF>159THENF1=-16
730 IFF< 80THENF1=+16
740 F=F+F1
780 POKE S+22,F
790 NEXT
799 RETURN
1100 POKE S+24,VL/2:REM VOLUME MOYEN
1199 RETURN
1200 REM INIT.PASSE-BAS
1210 POKE S+24,VL+16:REM PASSE-BAS
1220 POKE S+23,15*16+1:REM RESONANCE FORTE,VOIX 1
1230 POKE S+22, 40:REM FREQUENCE
1299 RETURN
1300 REM INIT.PASSE-HAUT
1310 POKE S+24,VL+64:REM PASSE-HAUT
1320 POKE S+23,15*16+1:REM RESONANCE FORTE,VOIX 1
1330 POKE S+22, 60:REM FREQUENCE
1399 RETURN
1400 REM INIT.PASSE-BANDE
1410 POKE S+24,VL+32:REM PASSE-BANDE
1420 POKE S+23,15*16+1:REM RESONANCE FORTE,VOIX 1
1430 POKE S+22, 60:REM FREQUENCE
1499 RETURN
1500 REM INIT.REJECTION
1510 POKE S+24,VL+16+64:REM REJECTION
1520 POKE S+23,15*16+1:REM RESONANCE FORTE,VOIX 1
1530 POKE S+22, 50:REM FREQUENCE
1599 RETURN
1600 REM INIT. SUIVEUR
1610 POKE S+24,VL+32:REM PASSE-BANDE
1620 POKE S+23,15*16+1:REM RESONANCE FORTE,VOIX 1
1630 POKE S+22,N*2:REM FREQUENCE
1699 RETURN
55555 SAVE"@0:FILTRES",8

READY.
```

FREQUENCES

Ce programme calcule la table des fréquences des différentes notes de DO 0 à LA dièse 7. La table est listée à l'écran ou à l'imprimante sous la forme:

Numéro de gamme, nom de la note, fréquence en Hertz, valeur de F sur 16 bits, de FL et FH (partie basse et partie haute de F)

```
100 REM AFFICHAGE DES 8 OCTAVES
110 REM UTILISABLES SUR CBM-64
120 REM AVEC VALEURS POUR LES REGISTRES
130 REM DE FREQUENCE
140 :
```

```
160 INPUT"ECRAN(3) OU IMPRIMANTE(4)";C
170 IF(C<3)OR(C>4)THEN160
180 OPEN4,C:DIM NO$(11)
190 FORI=0 TO 11 :READ NO$(I):NEXT
300 GOSUB 2000:F=15.4338531:U=11:G=-1
500 REM BOUCLE PRINCIPALE
520 F=F*(2^(1/12)):FI=INT(F/.05872535+.5)
530 FH=INT(FI/256):FL=FI-FH*256
540 U=U+1:IFU=12THEN U=0:G=G+1
570 GOSUB 1000: PRINT#4,G;" ";NO$(U),
600 PRINT#4,F$;FI$;FL$;FH$
990 IF F>3690THEN CLOSE4:END
992 GETA$:IFA$<>""THEN GOSUB 2000
998 IF F<3690 GOTO 500
999 :
1000 REM FORMATTE F,FI,FL,FH
1010 F$ =RIGHT$("          "+STR$(INT(F+.5)),8)
1020 FI$=RIGHT$("          "+STR$(FI),8)
1030 FL$=RIGHT$("      "+STR$(FL),6)
1040 FH$=RIGHT$("      "+STR$(FH),6)
1050 RETURN
1999 :
2000 PRINT:PRINT"POUSSEZ SUR ESPACE":PRINT:PRINT
2010 GETA$:IFA$=""GOTO2010
2020 PRINT"  NOTE     FREQUENCE     F      FL     FH":PRINT#4
2030 IFC=4THENPRINT#4,"  NOTE              FREQUENCE";
2040 IFC=4THENPRINT#4,"      F      FL     FH":PRINT#4
2050 RETURN
2999 :
10000 DATA"DO  ","DO #","RE  ","RE #","MI  ","FA  ","FA #
10010 DATA "SOL ","SOL#","LA  ","LA #","SI
55555 SAVE"@0:FREQUENCES",8

READY.
```

GLITCH.

Ce programme montre la différence entre un simple changement de couleur de fond de l'écran et la méthode plus sophistiquée où l'on attend la fin d'un balayage d'image pour changer de couleur.

Le 'GLITCH' en anglais est le parasite, le défaut de l'image que l'on observe lors d'un changement de couleur non synchronisé.

```
1 REM CHANGEMENT DE COULEUR ECRAN
2 REM AVEC SYNCHRONISATION SUR LE
3 REM BALAYAGE DE L'IMAGE
4 :
5 POKE53281,6
6 GOSUB600
8 PRINT"[CLR][BLANC]"
9 :
10 PRINT"[HOME]SYNCHRO ":FOR I=4TO15
20 POKE781,I:SYS679
30 FOR TD=1TO100:NEXT TD
40 GETA$:IFA$=""THEN NEXT I:GOTO 10
50 I=15:NEXTI
90 :
100 REM MODE NORMAL POUR COMPARER !!
110 PRINT"[HOME]NORMAL  ":FORI=4TO15
120 POKE 53281,I
130 FOR TD=1TO100:NEXT
140 GETA$:IFA$=""THEN NEXT I:GOTO 110
150 I=15:NEXTI
```

```
160 GOTO 5
199 :
600 REM CHARGE LE SYNCHRONISATEUR
610 RESTORE:CS=0:FORI=679TO698
620 READ A:POKEI,A:CS=CS+A:NEXT I
630 IFCS<>2767THEN STOP
679 DATA120,173,017,208,041,128,208,249
687 DATA173,017,208,041,128,240,249,142
695 DATA033,208,088,096
696 REM CHECKSUM= 2767 DE 679 A 698
699 RETURN
700 REM ON PEUT SYNCHRONISER L'ECRITURE
701 REM DANS N'IMPORTE QUEL REGISTRE
702 REM AVEC LE BALAYAGE D'ECRAN.METTRE
703 REM L'ADRESSE (PARTIE BASSE, PARTIE
704 REM HAUTE) DANS LES DATA 695 ET 696
705 REM ACTUELLEMENT, IL Y A 33 ET 208
706 REM SOIT $D021 (COULEUR DE FOND 0 )
707 REM MODIFIER AUSSI CS EN LIGNE 630.
708 :
55555 SAVE"@0:GLITCH",8

READY.
```

HELICO

Bruitage d'hélicoptère. La voix 1 du S.I.D. génère le bruit des pales tandis que la voix 2 simule le sifflement de la turbine. Pousser sur la barre d'espace pour arrêter.

```
10 REM BRUITAGE HELICOPTERE POUR CBM-64
20 :
100 S1=54272:S2=S1+7:GOSUB 500
110 REM                          VOIX 1
120 POKE S1+ 5, 32 :REM A= 2 D= 0
130 POKE S1+ 6,160 :REM S=15 R= 0
140 REM                          VOIX 2
150 POKE S2+ 1,255 :REM FREQ.AIGU
160 POKE S2+ 5,  0 :REM A= 0 D= 0
170 POKE S2+ 6, 80 :REM S= 5 R= 0
180 :
190 POKE S1+24, 15 :REM VOLUME A FOND
200 REM                DEBUT     VOIX 1
210 POKE S1+ 4,129 :REM BRUIT
220 REM                DEBUT     VOIX 2
230 POKE S2+ 4, 17 :REM TRIANGLE
240 :
250 FOR I=22TO 0STEP-4:POKE S1+1,I:NEXT
260 FOR I=0TO  44STEP2:POKE S1+1,I:NEXT
270 GET A$:IFA$="" GOTO 240
299 GOSUB500:LIST
500 FOR I=0TO24:POKE S1+I,0:NEXT:RETURN
55555 SAVE"@0:HELICO",8

READY.
```

HEXA

Programme de conversion décimal-->hexa et hexa-->décimal. Cette dernière conversion couvre les nombres hexadécimaux codés sur 2 ou sur 4 caractères

```
10 PRINT"[CLR]CONVERSIONS DECIMAL - HEXA [C.BAS][C.BAS][C.BAS]"
20 PRINT"[RVS ON]F1[RVS OFF]          DEC -> HEX
22 PRINT"[RVS ON]F3[RVS OFF] 1 OCTET  HEX -> DEC
23 PRINT"[RVS ON]F5[RVS OFF] 2 OCTETS HEX -> DEC
25 PRINT"[C.BAS]CHOIX ";:GOSUB500:A=ASC(W$)-132:IFA<1ORA>3 GOTO 10
30 PRINT:ON A GOTO 100,300,400
40 PRINT"[C.BAS]POUSSER [RVS ON]F7[RVS OFF] POUR CONTINUER"
50 GOSUB500:IFW$<>CHR$(136) GOTO 50
60 GOTO 10
100 INPUT"DEC";AD:GOSUB 1000:PRINT"           "AD$:GOTO 40
300 INPUT"HEX";OC$:IFLEN(OC$)<>2 GOTO 10
310 GOSUB 4000:PRINT"           "OC:GOTO 40
400 INPUT"HEX";AD$:IFLEN(AD$)<>4 GOTO 10
410 GOSUB 3000:PRINT"           "AD:GOTO 40
500 GETW$:IFW$=""GOTO500
599 RETURN
999 END
1000 AD$="":D=AD:REM------------------------2 OCTETS  DEC->HEX
1010 IFD=0GOTO1040
1020 A=INT(D/16):AD$=MID$("0123456789ABCDEF",1+D-A*16,1)+AD$
1030 D=A:GOTO1010
1040 AD$=RIGHT$("0000"+AD$,4):RETURN
2000 OC$="":D=OC:REM------------------------1 OCTET   DEC->HEX
2010 IFD=0GOTO2040
2020 A=INT(D/16):OC$=MID$("0123456789ABCDEF",1+D-A*16,1)+OC$
2030 D=A:GOTO2010
2040 OC$=RIGHT$("00"+OC$,2):RETURN
3000 D=0:REM------------------------------2 OCTETS  HEX->DEC
3020 FORI=1TO4:A=ASC(MID$(AD$,I,1))-48
3030 D=D*16+A+(A>9)*7:NEXT:AD=D:RETURN
4000 D=0:REM------------------------------1 OCTET   HEX->DEC
4010 FORI=1TO2:A=ASC(MID$(OC$,I,1))-48
4020 D=D*16+A+(A>9)*7:NEXT:OC=D:RETURN
4999 :
55555 SAVE"@0:HEXA",8

READY.
```

IMPRIME-120

Après un certain délai pour l'établissement des tables internes, le programme envoie en mode graphique une chaîne de caractères à l'imprimante (MPS801 ou 1526). Un générateur de caractères plus étroit permet de loger 120 caractères là où normalement 80 seulement prenaient place.

```
40000 REM **********************
40010 REM AUTEUR: THIERRY SALOME
40030 REM DATA DES CARACTERES
40040 REM **********************
40050 DATA128,128,128,128,128,223,128,128: REM   ET !
40060 DATA135,128,135,128,252,168,252,128: REM " ET #
40070 DATA198,255,177,128,241,136,199,128: REM $ ET %
40080 DATA182,233,246,128,128,135,128,128: REM & ET '
40090 DATA156,162,193,128,193,162,156,128: REM ( ET )
40100 DATA152,188,140,128,136,188,136,128: REM * ET +
40110 DATA192,176,128,128,136,136,136,128: REM , ET -
40120 DATA128,192,128,128,240,136,135,128: REM . ET /
40130 DATA255,221,255,128,132,130,255,128: REM 0 ET 1
40140 DATA243,201,231,128,193,201,182,128: REM 2 ET 3
40150 DATA143,136,252,128,239,201,249,128: REM 4 ET 5
40160 DATA255,201,249,128,243,137,135,128: REM 6 ET 7
40170 DATA255,201,255,128,207,201,255,128: REM 8 ET 9
40180 DATA128,200,128,128,192,180,128,128: REM : ET ;
40190 DATA144,168,196,128,168,168,168,128: REM < ET =
```

```
40200 DATA196,168,144,128,130,217,134,128: REM > ET ?
40210 DATA251,217,255,128,254,137,254,128: REM @ ET A
40220 DATA255,197,186,128,190,193,193,128: REM B ET C
40230 DATA255,193,190,128,255,201,193,128: REM D ET E
40240 DATA255,137,129,128,190,193,249,128: REM F ET G
40250 DATA255,136,255,128,193,255,193,128: REM H ET I
40260 DATA193,255,129,128,255,136,246,128: REM J ET K
40270 DATA255,192,192,128,255,130,255,128: REM L ET M
40280 DATA255,188,255,128,190,193,190,128: REM N ET O
40290 DATA255,137,134,128,190,225,254,128: REM P ET Q
40300 DATA255,153,230,128,198,201,177,128: REM R ET S
40310 DATA129,255,129,128,255,192,255,128: REM T ET U
40320 DATA191,192,191,128,191,248,191,128: REM V ET W
40330 DATA247,136,247,128,135,248,135,128: REM X ET Y
40340 DATA241,201,199,128,255,193,193,128: REM Z ET [
40350 DATA254,213,195,128,193,193,255,128: REM \ ET ]
40360 DATA130,255,130,128,136,156,136,128: REM ^ ET
40370 DIMTC$(256),TL$(6,1): REM TC$=TABLEAU CARATERES; TL$=TABLEAU LIGN
      ES
40380 FORI=32TO95:READA,B,C,D: REM DANS CODE ASCII, CARACTERES   A
40390 TC$(I)=CHR$(A)+CHR$(B)+CHR$(C)+CHR$(D):NEXT: REM DEFINITION DU CA
      RACTERE
40400 TC$(255)=CHR$(248)+CHR$(136)+CHR$(252)+CHR$(128): REM DEFINITION
      DE PI
40410 OPEN3,4:PRINT#3,CHR$(8): REM IMPRIMANTE, CHR$(8) = MODE GRAPHIQUE

40420 REM ******************************
40430 REM DEMONSTRATION
40440 REM ******************************
40450 VI$="0          0          0          0          0          0          0
      "
40460 VI$=VI$+"    0          0          0          1          1          1"

40470 GOSUB40630
40480 VI$="0          1          2          3          4          5          6
      "
40490 VI$=VI$+"    7          8          9          0          1          2"

40500 GOSUB40630
40510 VI$="12345678901234567890123456789012345678901234567890123456789012
      3456"
40520 VI$=VI$+"78901234567890123456789012345678901234567890123456789012345
      67890"

40530 GOSUB40630
40540 VI$="SET DE CARACTERES:    ! "+CHR$(34)+" # $ % & ' ( ) * + , - .
      / 0 1 "
40550 VI$=VI$+"2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M
      N O P "
40560 VI$=VI$+"Q R":GOSUB40630
40570 VI$="S T U V W X Y Z [ \ ] ^ _ ":GOSUB40630:PRINT#3,CHR$(15):CLO
      SE3:END
40580 REM ******************************
40590 REM VI$=VARIABLE DE LA ROUTINE CONTENANT CHAINE A IMPRIMER
40600 REM TL$(6,1)=6 SECTIONS DE LIGNE POUR UNE LIGNE D'IMPRESSION
40610 REM I=COMPTEUR DE CARACTERES DANS VI$
40620 REM ******************************
40630 L=0:I=1
40640 IFL=6THEN40700: REM LIGNE COMPLETE
40650 L=L+1:TL$(L,1)="":J=1
40660 IFMID$(VI$,I,1)=""THEN40700: REM FIN DE LA VARIABLE VI$
40670 TL$(L,1)=TL$(L,1)+TC$(ASC(MID$(VI$,I,1))): REM CARACTERE DANS SEC
      TION
40680 I=I+1:J=J+1:IFJ=63THEN40640: REM FIN D'UNE SECTION DE LIGNE
40690 GOTO40660: REM CONTINUE A REMPLIR LA SECTION
40700 FORI=1TOL:PRINT#3,TL$(I,1);:NEXT:PRINT#3,CHR$(128): REM IMPRESSIO
      N LIGNE
40710 RETURN
55555 SAVE"@0:IMPRIME-120",8

READY.
```

IMPRIME-HR.

Ce programme recopie l'écran sur l'imprimante, quel que soit le mode graphique utilisé : caractères normaux, programmables, modes étendus, modes BITMAP, modes couleur.

Soyez cependant attentifs aux quelques remarques suivantes:

1. Pour les caractères normaux, il faut au préalable recopier le générateur de caractères en RAM.

2. Pour les images situées derrière les ROM BASIC ou KERNAL, il faut remplacer les PEEK du programme par USR et charger en mémoire la routine USR-PEEK.

3. Les modes couleurs peuvent donner des résultats bons ou mauvais. Les meilleurs résultats sont obtenus lorsque les couleurs claires sont représentées dans le BITMAP par les deux bits '00', les couleurs de luminosité moyenne par des '01' et '10', les couleurs sombres par '11'. Ceci n'est pas toujours possible avec les programmes commerciaux.

4. Pour recopier une image-écran d'un programme de jeu ou autre programme commercial, il suffit de repérer dans le programme l'adresse des zones écran et générateur utilisées. Ensuite, on lance le jeu puis, au moment désiré, on effectue un RESET, on charge ensuite IMPRIME-HR et le tour est joué!

5. On peut inverser l'image en négatif en remplaçant "CHR$(" par "CHR$(255-" dans les lignes 55 et 1010.

6. Les écrans rangés sur disque avec le programme "KOALA-PAINTER" peuvent être imprimés en les chargeant par :

LOAD "...",8,1

Le premier caractère du nom est un CHR$(129). A la question "adresse bitmap", répondre 24576.

```
0 REM IMPRIME-HR
1 REM            RECOPIE L'ECRAN
2 REM            SUR IMPRIMANTE 1526 OU MPS 801
3 REM LES MODES SUPPORTES SONT: BITMAP, CARACTERES PROGRAMMABLES,
4 REM CARACTERES ETENDU
5 REM (POUR LES CARACTERES NORMAUX,RECOPIER LE GENERATEUR EN RAM)
6 REM NOTER QUE L'IMAGE SERA TOURNEE DE 90 DEGRES
7 REM VERS LA DROITE PAR RAPPORT A L'ECRAN.
9 :
10 OPEN4,4:DIM Z(199):C8$=CHR$(8)
11 INPUT"[C.BAS]C.BAS]BITMAP(1),CARACTERES(2),ETENDU(3)";N
12 IF N=1THEN S=8192:INPUT"ADRESSE BITMAP (RETURN=8192)";S
13 IF N=2THEN EC=34816:INPUT"ADRESSE ECRAN (RETURN=34816)";EC
14 IF(N=2)OR(N=3)THEN GB=32768:INPUT"ADRESSE GEN.CAR.(RETURN=32768)";GB

15 CO=0:C1=1:C7=7:C8=8:CA=127:CB=128
16 FORI=0TO7:H(I)=2^I:NEXT
20 FOR I=39 TO CO STEP-1:PRINT#4,C8$;:FOR J=COTO24
30 FOR K=CO TO C7:Q=J*C8+K
40 ON N GOSUB 2000,3000,4000
```

```
50 Z(Q)=Z(Q)+M*H(Y)
55 PRINT#4,CHR$((Z(Q)AND CA)+CB);
60 Z(Q)=Z(Q)/128:NEXT K,J:PRINT#4
80 Y=Y+C1:IF Y=C7 THEN Y=C0:GOTO 1000
100 NEXT I
102 :
109 R=C1
1000 PRINT#4,C8$;:FOR L=C0 TO 199
1010 PRINT#4,CHR$((Z(L)AND CA)+CB);
1020 Z(L)=Z(L)/CB:NEXT L:PRINT#4
1040 IF R=C0 THEN 100
1050 CLOSE 4:END
1051 :
2000 REM LECTURE D'UN OCTET DU BITMAP (J,I, LIGNE K)
2040 M1=S+320*J+I*C8+K
2050 M=PEEK(M1)
2998 RETURN
2999 :
3000 REM LECTURE D'UN OCTET DU CARACTERE(J,I, LIGNE K)DANS LE GENERATEU
     R
3040 M1=PEEK(EC+I+40*J):REM CODE ECRAN DU CARACTERE
3050 M=PEEK(GB+M1*C8+K):REM OCTET DU GENERATEUR
3998 RETURN
3999 :
4000 REM LECTURE D'UN OCTET DU CARACTERE(J,I, LIGNE K)
4040 M1=PEEK(EC+I+40*J)AND 63:REM CODE ECRAN DU CARACTERE SUR 6 BITS
4050 M=PEEK(GB+M1*C8+K):REM OCTET DU GENERATEUR
4998 RETURN
55554 END
55555 SAVE"@0:IMPRIME-HR",8

READY.
```

INIMEM

Redescend le bas de mémoire en 16384 pour protéger du BASIC le banc-mémoire 1 du VIC-II aux fins d'utilisations graphiques. Il reste 20K de libre pour le BASIC. A utiliser entre autres avant de lancer "PETITMON" qui permet de charger le programme "MON" en mémoire basse.

```
10 REM INITIALISE BAS DE BASIC
30 :
40 POKE 16384,0:POKE43,1:POKE44,64
50 :
60 NEW
55555 SAVE"@0:INIMEM",8

READY.
```

ITALIQUES.

Recopie 64 caractères de ROM en RAM et modifie ensuite ces caractères de 'normal' en 'italique' en multipliant par 2 pour les décaler vers la droite les octets de la moitié inférieure de

chaque caractère. Pousser sur STOP pour conserver les caractères modifiés jusqu'au prochain STOP-RESTORE.

```
0 REM CARACTERES ITALIQUES POUR CBM 64
1 :
2 :
10 REM RECOPIE 64 CARACTERES DE ROM EN RAM
15 RAM=32768:ROM=53248:MAX=256*8
16 PRINT"[CLR]UN PETIT INSTANT, S.V.P..."
20 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
30 FORI=0TOMAX:POKERAM+I,PEEK(ROM+I):NEXTI
40 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
50 REM PROTEGE LE SOMMET DE MEMOIRE EN 32768
60 POKE55,0:POKE56,128
70 REM PROGRAMME LE VIC-II POUR LE BANC 2
80 POKE56576,197
90 REM ECRAN EN 34816, GEN.CAR EN 32768
100 POKE53272,32
110 REM DIT AU KERNAL OU EST L'ECRAN
120 POKE648,(34816/256):PRINTCHR$(147)
130 PRINT"A B C D E F G H I J K L M N O P Q R S T"
135 PRINT"[C.BAS][C.BAS]U V W X Y Z "
140 PRINT"[C.BAS][C.BAS]0 1 2 3 4 5 6 7 8 9 ! # $ % & ' ( )"
145 PRINT"[C.BAS][C.BAS]+ - *  = , / ";CHR$(34)
150 PRINT"[C.BAS][C.BAS]< > ? [ ] : ; = ^ _ \"
160 FORJ=32768TO32768+64*8STEP8
170 FORI=4TO7:POKEI+J,(PEEK(I+J)*2)AND255
180 NEXTI,J
190 PRINT"[C.BAS][C.BAS][C.BAS]POUSSER SUR LA BARRE D'ESPACE"
200 PRINT"[C.BAS]OU STOP POUR CONSERVER LES CARACTERES"
210 PRINT"[C.BAS]INCHANGES."
220 GETA$:IFA$=""GOTO 220
230 POKE56576,199:POKE53272,21:POKE648,4:PRINT"[CLR]"
240 END
55555 SAVE"@0:ITALIQUES",8

READY.
```

JOYSTICK.

Démonstration de la manière d'utiliser le JOYSTICK pour déplacer le curseur dans l'écran. Le curseur laisse une trace derrière lui sauf lorsque le bouton de tir est enfoncé. A l'arrêt, l'enfoncement du bouton provoque le clignotement du curseur. La touche 'CLR' efface l'écran.

```
1 REM**********************************
2 REM*    UTILISATION DU JOYSTICK     *
3 REM*                                *
4 REM*         PAR P. BRUNET          *
5 REM*      COPYRIGHT BCM 1983        *
6 REM**********************************
7 :
8 REM MISE A BLANC DE LA RAM COULEUR
10 FOR I=55295TO56295:POKE I,1:NEXT I
32 REM CREATION DE LA TABLE DES MOV.
33 :
40 DIM PO(11)
50 FOR I=1 TO 11:READPO(I):NEXT I
80 DATA41,-39,1,0,39,-41,-1,0,40,-40,0
82 REM**********************************
83 REM*   DEBUT DU PROGRAMME           *
84 REM**********************************
85 :
```

```
86 REM EFFACE LA RAM ECRAN
90 FOR I=1024 TO 2023:POKEI,32:NEXT I
120 PO=1024:REM POSITION DE DEPART
121 :
122 REM JOY:VALEUR DU REGISTRE
123 REM      JOYSTICK N.2
124 :
125 REM*********************************
126 REM*   DEBUT BOUCLE PRINCIPALE     *
127 REM*********************************
128 :
130 JOY=PEEK(56320)
132 REM TEST DE TOUCHE POUR EFFACER
135 IF PEEK(197)=51 GOTO 90
137 REM P:INDICE DANS LA TABLE DES MOV.
140 P=JOY-100
142 REM SI P>16 PAS DE TRAINEE
150 IF P>16 THEN P=P-16 : GOTO 170
152 REM EFFACEMENT DU POINT
160 POKE PO,32
162 REM CALCUL DE LA NOUVELLE POSITION
170 PO=PO+PO(P)
172 REM TEST POUR SCROLLING
180 IF PO<1024 THEN PO=PO+999
190 IF PO>2023 THEN PO=PO-999
192 REM AFFICHAGE DU POINT ET BOUCLAGE
193 :
200 POKE PO,42:GOTO 130
201 :
55555 SAVE"@0:JOYSTICK",8

READY.
```

LISTING-BAS ET LISTING SRC.

Programme en langage machine qui occupe les adresses 52525 à 53108. Il modifie le KERNAL pour la gestion du périphérique numéro 4 (l'imprimante). La sortie se fait alors sur la porte parallèle vers une imprimante au standard "CENTRONICS". C'est avec ce programme que la plupart des listes de ce livre ont été imprimées. Les codes spéciaux COMMODORE sont traduits en clair et les lignes-suite sont décalées pour augmenter la lisibilité.

Pour réaliser le câble nécessaire, voir les remarques du programme CENTRO-BAS.

```
10 CS=0:FORI=52525TO53108:READA:POKEI,A:CS=CS+A:NEXTI
20 IFCS<>57167THENSTOP
30 SYS52525
99 END
52525 DATA 169,100,141,38,3,169,205,141
52533 DATA 39,3,169,37,141,32,3,169
52541 DATA 206,141,33,3,169,60,141,28
52549 DATA 3,169,206,141,29,3,169,255
52557 DATA 141,3,221,169,0,141,1,221
52565 DATA 141,89,206,173,2,221,9,4
52573 DATA 141,2,221,173,13,221,96,141
52581 DATA 85,206,142,86,206,140,87,206
52589 DATA 165,154,201,4,208,22,173,85
52597 DATA 206,160,0,32,229,205,32,149
52605 DATA 205,173,85,206,174,86,206,172
52613 DATA 87,206,24,96,173,85,206,174
```

```
52621 DATA 86,206,172,87,206,76,202,241
52629 DATA 192,0,208,95,201,13,208,10
52637 DATA 32,167,205,160,0,140,89,206
52645 DATA 169,10,141,1,221,173,0,221
52653 DATA 41,251,141,0,221,9,4,141
52661 DATA 0,221,173,13,221,41,16,201
52669 DATA 16,208,247,169,0,133,144,238
52677 DATA 89,206,173,89,206,201,72,48
52685 DATA 22,160,0,140,89,206,185,108
52693 DATA 207,201,0,240,7,32,167,205
52701 DATA 200,76,211,205,206,89,206,96
52709 DATA 162,37,221,90,206,240,6,202
52717 DATA 224,255,208,246,96,142,88,206
52725 DATA 160,1,96,162,255,232,189,128
52733 DATA 206,240,3,76,250,205,206,88
52741 DATA 206,173,88,206,201,255,208,237
52749 DATA 169,91,32,167,205,232,189,128
52757 DATA 206,201,0,240,6,32,167,205
52765 DATA 76,18,206,169,93,76,167,205
52773 DATA 32,15,243,240,3,76,1,247
52781 DATA 32,31,243,165,186,201,4,240
52789 DATA 3,76,80,242,76,117,242,32
52797 DATA 20,243,240,3,76,150,242,32
52805 DATA 31,243,138,72,165,186,201,4
52813 DATA 240,3,76,157,242,76,241,242
52821 DATA 0,0,0,0,0,5,8,9
52829 DATA 14,17,18,19,20,28,29,30
52837 DATA 31,142,144,145,146,147,148,156
52845 DATA 157,158,159,129,149,150,151,152
52853 DATA 153,154,155,133,137,134,138,135
52861 DATA 139,136,140,0,66,76,65,78
52869 DATA 67,0,83,85,80,80,46,83
52877 DATA 72,43,67,61,0,65,85,84
52885 DATA 46,83,72,43,67,61,0,77
52893 DATA 73,78,85,83,46,0,67,46
52901 DATA 66,65,83,0,82,86,83,32
52909 DATA 79,78,0,72,79,77,69,0
52917 DATA 68,69,76,0,82,79,85,71
52925 DATA 69,0,67,46,68,82,79,73
52933 DATA 84,69,0,86,69,82,84,0
52941 DATA 66,76,69,85,0,77,65,74
52949 DATA 85,83,46,0,78,79,73,82
52957 DATA 0,67,46,72,65,85,84,0
52965 DATA 82,86,83,32,79,70,70,0
52973 DATA 67,76,82,0,73,78,83,84
52981 DATA 0,80,79,85,82,80,82,69
52989 DATA 0,67,46,71,65,85,67,72
52997 DATA 69,0,74,65,85,78,69,0
53005 DATA 67,89,65,78,0,79,82,65
53013 DATA 78,71,69,0,79,82,65,78
53021 DATA 71,69,32,76,46,0,82,79
53029 DATA 83,69,0,67,89,65,78,32
53037 DATA 76,46,0,80,79,85,82,80
53045 DATA 82,69,32,76,46,0,86,69
53053 DATA 82,84,32,76,46,0,66,76
53061 DATA 69,85,32,76,46,0,74,65
53069 DATA 85,78,69,32,76,46,0,70
53077 DATA 49,0,70,50,0,70,51,0
53085 DATA 70,52,0,70,53,0,70,54
53093 DATA 0,70,55,0,70,56,0,13
53101 DATA 10,32,32,32,32,32,32,0
55555 SAVE"@0:LISTING-BAS",8

READY.
```

```
00001  0000                ;*******************************************************
00002  0000                ; GESTION D'IMPRIMANTE PARALLELE
00003  0000                ;
00004  0000                ; SUR PORT UTILISATEUR POUR LE
00005  0000                ;
00006  0000                ; CBM 64. INITIALISE PAR :
00007  0000                ;
00008  0000                ; SYS52525
00009  0000                ;
00010  0000                ; LISTER EN UTILISANT CECI :
00011  0000                ;
00012  0000                ; SYS 52525:OPEN 4,4:CMD 4:LIST
00013  0000                ;
00014  0000                ; CE PROGRAMME EST SITUE EN $CD2D (52525)
00015  0000                ;
00016  0000                ; A UTILISER POUR CREER DES LISTINGS
00017  0000                ;SUR UNE IMPRIMANTE CLASSIQUE.
00018  0000                ; LES CODES-ECRAN SPECIAUX DU 64 SONT LISTES EN CLAIR ENTRE CROCHETS
00019  0000                ;*******************************************************
00020  0000                ;ATTENTION, DANS CE LISTING :
00021  0000                ;'PLUS GRAND QUE' EST DONNE PAR: '~'
00022  0000                ;'PLUS PETIT QUE' EST DONNE PAR: '^'
00023  0000                ;*******************************************************
00024  0000                ;
00025  0000                      LF     = $0A    ;AVANCE PAPIER
00026  0000                      CR     = $0D    ;RETOUR CHARIOT
00027  0000                      CANAL  = $04    ;NUMERO DU CANAL UTILISE
00028  0000                      STATUS = $90    ;OCTET ST-STATUS
00029  0000                      PERI   = $9A    ;ADRESSE DU NUMERO DE PERIPH. DE SORTIE
00030  0000                      DEVICE = $BA    ;N.PERIPH DE SORTIE
00031  0000                      VECCLO = $031C  ;VECTEUR DE CLOSE
00032  0000                      VECSOT = $0320  ;VECTEUR INIT.SORTIE
00033  0000                      VECWRT = $0326  ;VECTEUR DE SORTIE D'UN CARACTERE
00034  0000                      WRT    = $F1CA  ;ROUTINE SORTIE D'UN CARACTERE
00035  0000                      SOT    = $F250  ;ROUTINE INIT. SORTIE
00036  0000                      CLOS   = $F291  ;ROUTINE DE CLOSE
00037  0000                      PORTA  = $DD00  ;PORT A DU CIA #2 POUR HANDSHAKE
00038  0000                      PORTB  = $DD01  ;PORT B DU CIA #2 :PORT UTILISATEUR
00039  0000                      DDRA   = $DD02  ;REGISTRE DE DIRECTION A
00040  0000                      DDRB   = $DD03  ;REGISTRE DE DIRECTION B
00041  0000                      ICR    = $DD0D  ;REGISTRE DE CONTROLE D'INTERRUPTION
00042  0000                      TABLEN = 37     ;NOMBRE DE CODES A CONVERTIR
00043  0000                      LINLEN = 72     ;NOMBRE DE CARACTERES PAR LIGNE + 2
00044  0000                ;
00045  0000                ;*******************************************************
00046  0000                ;
00047  0000                      DEBUT = $CD2D
00048  0000                ;
00049  0000                ;*******************************************************
00050  0000                ;
00051  0000                      * = DEBUT
00052  CD2D                ;
00053  CD2D  A9 64               LDA #<PRINT     ;INITIALISE LE VECTEUR
00054  CD2F  8D 26 03            STA VECWRT      ;DE SORTIE D'UN CARACTERE
00055  CD32  A9 CD               LDA #>PRINT     ;VERS LA ROUTINE CI-DESSOUS
00056  CD34  8D 27 03            STA VECWRT+1
00057  CD37  A9 25               LDA #<SO1       ;INIT.VECTEUR
00058  CD39  8D 20 03            STA VECSOT      ;D'INIT. SORTIE
00059  CD3C  A9 CE               LDA #>SO1
00060  CD3E  8D 21 03            STA VECSOT+1
00061  CD41  A9 3C               LDA #<CLO1      ;INIT.VECTEUR
00062  CD43  8D 1C 03            STA VECCLO      ;DE CLOSE
00063  CD46  A9 CE               LDA #>CLO1
00064  CD48  8D 1D 03            STA VECCLO+1
00065  CD4B  A9 FF               LDA #$FF        ;PROGRAMME LE PORT B EN SORTIE
00066  CD4D  8D 03 DD            STA DDRB
00067  CD50  A9 00               LDA #$00        ;REMET A ZERO LE PORT DE SORTIE
00068  CD52  8D 01 DD            STA PORTB
00069  CD55  8D 59 CE            STA NCAR
00070  CD58  AD 02 DD            LDA DDRA        ;PROGRAMME LE BIT 2 DU PORT A
00071  CD5B  09 04               ORA #$04        ;EN SORTIE
00072  CD5D  8D 02 DD            STA DDRA
```

```
00073  CD60  AD 0D DD        LDA ICR          ;ANNULE LE DRAPEAU SI NECESSAIRE
00074  CD63  60              RTS              ;C'EST TOUT
00075  CD64                  ;
00076  CD64                  ;*****************************************************
00077  CD64                  ;
00078  CD64  8D 55 CE  PRINT STA CAR          ;SAUVE LE CARACTERE,
00079  CD67  8E 56 CE        STX XPROV        ;LE REGISTRE X
00080  CD6A  8C 57 CE        STY YPROV        ;ET LE REGISTRE Y
00081  CD6D  A5 9A           LDA PERI         ;LE CANAL DE SORTIE
00082  CD6F  C9 04           CMP #CANAL       ;EST-IL LE BON ?
00083  CD71  D0 16           BNE FIN          ;SI NON,ON NE FAIT RIEN
00084  CD73  AD 55 CE        LDA CAR          ;SI OUI, PRENDRE LE CARACTERE
00085  CD76  A0 00           LDY #$00         ;SI Y RESTE A ZERO APRES CONVER, IL N'Y A
00086  CD78  20 E5 CD        JSR CONVER       ;PAS EU CONVERSION DES CODES-ECRAN
00087  CD7B  20 95 CD        JSR DIALOG       ;ENVOI DU (DES) CARACTERE(S) EN MODE 'CENTRONICS'
00088  CD7E  AD 55 CE        LDA CAR          ;RECUPERE LE CARACTERE
00089  CD81  AE 56 CE        LDX XPROV        ;RECUPERE LE REGISTRE X
00090  CD84  AC 57 CE        LDY YPROV        ;RECUPERE LE REGISTRE Y
00091  CD87  18              CLC              ;CARRY = 0 CAR PAS D'ERREUR
00092  CD88  60              RTS              ;ET C'EST TOUT
00093  CD89                  ;
00094  CD89                  ;*****************************************************
00095  CD89                  ;
00096  CD89  AD 55 CE  FIN   LDA CAR          ;SORTIE SI PAS POUR LE
00097  CD8C  AE 56 CE        LDX XPROV
00098  CD8F  AC 57 CE        LDY YPROV
00099  CD92  4C CA F1        JMP WRT          ;CANAL IMPRIMANTE
00100  CD95                  ;
00101  CD95                  ;*****************************************************
00102  CD95  C0 00     DIALOG CPY #$00        ;ENVOI UNE CHAINE PAR DIAL1 OU UN CARACTERE
00103  CD97  D0 5F           BNE DIAL1        ;SEUL AVEC DIAL
00104  CD99                                   ;SEUL DIAL TOTALISE LE N. DE CARACTERES/LIGNE.
00105  CD99  C9 0D           CMP #$0D
00106  CD9B  D0 0A           BNE DIAL         ;SI 'RETOUR CHARIOT'
00107  CD9D  20 A7 CD        JSR DIAL         ;L'ENVOYER
00108  CDA0  A0 00           LDY #$00         ;REMET A ZERO LE COMPTEUR DE CARACTERES
00109  CDA2  8C 59 CE        STY NCAR         ;QUAND ON VA A LIGNE
00110  CDA5  A9 0A           LDA #$0A         ;SUIVI DE 'AVANCE PAPIER'
00111  CDA7                  ;
00112  CDA7  8D 01 DD  DIAL  STA PORTB        ;ECRIRE LE CARACTERE A ENVOYER
00113  CDAA  AD 00 DD        LDA PORTA        ;METTRE A.ZERO LE BIT 2 DU PORT A
00114  CDAD  29 FB           AND #$FB
00115  CDAF  8D 00 DD        STA PORTA
00116  CDB2  09 04           ORA #$04         ;REMETTRE A UN LE BIT 2 DU PORT A
00117  CDB4  8D 00 DD        STA PORTA
00118  CDB7                  ;
00119  CDB7  AD 0D DD  LOOP  LDA ICR          ;LIRE LE BIT 4 DU REGISTRE
00120  CDBA  29 10           AND #$10         ;ET ATTENDRE QU'IL PASSE A UN
00121  CDBC  C9 10           CMP #$10         ;CE QUI SIGNIFIE 'ACK' RECU
00122  CDBE  D0 F7           BNE LOOP
00123  CDC0  A9 00           LDA #$00         ;REMISE A ZERO DU ST-STATUS
00124  CDC2  85 90           STA STATUS       ;SI TRANSMISSION SANS ERREUR
00125  CDC4  EE 59 CE        INC NCAR         ;COMPTE LE NOMBRE DE CARACTERES DANS LA LIGNE
00126  CDC7  AD 59 CE        LDA NCAR
00127  CDCA  C9 48           CMP #LINLEN
00128  CDCC  30 16           BMI ENDDI2       ;SI FIN DE LIGNE,
00129  CDCE  A0 00           LDY #$00         ;REMETTRE LE COMPTEUR DE CARACTERES
00130  CDD0  8C 59 CE        STY NCAR         ;A ZERO
00131  CDD3  B9 6C CF  DIAL2 LDA NLIN,Y
00132  CDD6  C9 00           CMP #$00         ;ENVOI DU DEBUT DE LA LIGNE SUITE
00133  CDD8  F0 07           BEQ ENDDI1
00134  CDDA  20 A7 CD        JSR DIAL
00135  CDDD  C8              INY
00136  CDDE  4C D3 CD        JMP DIAL2
00137  CDE1  CE 59 CE  ENDDI1 DEC NCAR        ;POUR COMPENSER LE 'RETOUR CHARIOT'
00138  CDE4  60        ENDDI2 RTS
00139  CDE5                  ;
00140  CDE5                  ;*****************************************************
00141  CDE5  A2 25     CONVER LDX #TABLEN
00142  CDE7  DD 5A CE  CHERCH CMP TABLE,X
00143  CDEA  F0 06           BEQ FOUND        ;CODE TROUVE : X=POSITION DU CODE DANS LA TABLE
00144  CDEC  CA              DEX
00145  CDED  E0 FF           CPX #$FF
```

```
00146  CDEF  D0 F6        BNE CHERCH      ;SI PAS TROUVE, RETOUR ET IMPRESSION
00147  CDF1               ;
00148  CDF1  60           RTS             ;ORDINAIRE (Y EST A ZERO)
00149  CDF2               ;
00150  CDF2  8E 58 CE     FOUND  STX CNT  ;CNT=POS. DU MESSAGE DANS LA TABLE
00151  CDF5  A0 01        LDY #$01
00152  CDF7  60           RTS             ;Y = 1 SIGNIFIE QUE LA CONVERSION A EU LIEU
00153  CDF8               ;
00154  CDF8               ;*****************************************************
00155  CDF8               ;
00156  CDF8  A2 FF        DIAL1  LDX #$FF ;IL Y A UN 00 DEVANT CHAQUE MESSAGE
00157  CDFA  E8           DI3    INX
00158  CDFB  BD 80 CE     LDA MSG,X
00159  CDFE  F0 03        BEQ ZERO        ;ON COMPTE LES MESSAGES
00160  CE00  4C FA CD     JMP DI3
00161  CE03               ;
00162  CE03  CE 58 CE     ZERO   DEC CNT  ;EN DECREMENTANT A CHAQUE FOIS CNT
00163  CE06  AD 58 CE     LDA CNT
00164  CE09  C9 FF        CMP #$FF        ;EST-CE LE BON MESSAGE ?
00165  CE0B  D0 ED        BNE DI3         ;SI NON, CONTINUER A DECOMPTER
00166  CE0D  A9 5B        LDA #'[         ;SI OUI, ENVOI DU CROCHET DE DEBUT DE MESSAGE
00167  CE0F  20 A7 CD     JSR DIAL        ;ENVOI DU CARACTERE
00168  CE12  E8           DI4    INX
00169  CE13               ;               ;X POINTE VERS LE DEBUT DU BON MESSAGE
00170  CE13  BD 80 CE     LDA MSG,X
00171  CE16  C9 00        CMP #$00        ;SI FIN DE MESSAGE,
00172  CE18  F0 06        BEQ DI5         ;ON SORT
00173  CE1A  20 A7 CD     JSR DIAL        ;SINON,ENVOI D'UN CARACTERE DU MESSAGE
00174  CE1D  4C 12 CE     JMP DI4         ;ET RECOMMENCER POUR LE CARACTERE SUIVANT
00175  CE20               ;
00176  CE20  A9 5D        DI5    LDA #']  ;ENVOI DU CROCHET DE FIN DE MESSAGE
00177  CE22  4C A7 CD     JMP DIAL        ;LE MESSAGE COMPLET EST ENVOYE
00178  CE25               ;
00179  CE25               ;*****************************************************
00180  CE25               ;ROUTINE INITIALISATION DU CANAL DE SORTIE
00181  CE25               ;
00182  CE25  20 0F F3     SO1    JSR $F30F ;FICHIER EXISTE?
00183  CE28  F0 03        BEQ SO2
00184  CE2A  4C 01 F7     JMP $F701       ;SINON, IL Y A ERREUR
00185  CE2D  20 1F F3     SO2    JSR $F31F ;LIT N.DE PERIPH.
00186  CE30  A5 BA        LDA DEVICE      ;EST-CE L'IMPRIMANTE?
00187  CE32  C9 04        CMP #CANAL
00188  CE34  F0 03        BEQ SO3         ;SINON, ON ENVOIE
00189  CE36  4C 50 F2     JMP SOT         ;AU KERNAL.
00190  CE39  4C 75 F2     SO3    JMP $F275 ;METTRE SON NUMERO EN $9A
00191  CE3C               ;
00192  CE3C               ;*****************************************************
00193  CE3C               ;ROUTINE DE CLOSE-FERME LE FICHIER LOGIQUE
00194  CE3C               ;
00195  CE3C  20 14 F3     CLO1   JSR $F314 ;FICHIER OUVERT?
00196  CE3F  F0 03        BEQ CLO2        ;SINON, FIN EN
00197  CE41  4C 96 F2     JMP $F296       ;ERREUR 'FILE NOT OPEN'
00198  CE44  20 1F F3     CLO2   JSR $F31F
00199  CE47  8A           TXA
00200  CE48  48           PHA
00201  CE49  A5 BA        LDA DEVICE      ;SI CE N'EST PAS
00202  CE4B  C9 04        CMP #CANAL      ;L'IMPRIMANTE
00203  CE4D  F0 03        BEQ CLO3
00204  CE4F  4C 9D F2     JMP $F29D       ;FIN DU CLOSE NORMAL
00205  CE52  4C F1 F2     CLO3   JMP $F2F1
00206  CE55               ;
00207  CE55               ;*****************************************************
00208  CE55               CAR    * = *+1
00209  CE56               XPROV  * = *+1
00210  CE57               YPROV  * = *+1
00211  CE58               CNT    * = *+1
00212  CE59               NCAR   * = *+1
00213  CE5A               ;*****************************************************
00214  CE5A               ;
00215  CE5A  05           TABLE  .BYTE   5,  8,  9, 14, 17, 18, 19, 20
00215  CE5B  08
```

```
00215  CE5C  09
00215  CE5D  0E
00215  CE5E  11
00215  CE5F  12
00215  CE60  13
00215  CE61  14
00216  CE62  1C                .BYTE  28, 29, 30, 31,142,144,145,146
00216  CE63  1D
00216  CE64  1E
00216  CE65  1F
00216  CE66  8E
00216  CE67  90
00216  CE68  91
00216  CE69  92
00217  CE6A  93                .BYTE 147,148,156,157,158,159,129,149
00217  CE6B  94
00217  CE6C  9C
00217  CE6D  9D
00217  CE6E  9E
00217  CE6F  9F
00217  CE70  81
00217  CE71  95
00218  CE72  96                .BYTE 150,151,152,153,154,155,133,137
00218  CE73  97
00218  CE74  98
00218  CE75  99
00218  CE76  9A
00218  CE77  9B
00218  CE78  85
00218  CE79  89
00219  CE7A  86                .BYTE 134,138,135,139,136,140
00219  CE7B  8A
00219  CE7C  87
00219  CE7D  8B
00219  CE7E  88
00219  CE7F  8C
00220  CE80                ;
00221  CE80                ;*******************************************************
00222  CE80                ;
00223  CE80  00          MSG   .BYTE $00        ;DEBUT DE LA TABLE DES MESSAGES
00224  CE81  42 4C             .BYTE 'BLANC',$00
00224  CE86  00
00225  CE87  53 55             .BYTE 'SUPP.SH+C=',$00
00225  CE91  00
00226  CE92  41 55             .BYTE 'AUT.SH+C=',$00
00226  CE9B  00
00227  CE9C  4D 49             .BYTE 'MINUS.',$00
00227  CEA2  00
00228  CEA3  43 2E             .BYTE 'C.BAS',$00
00228  CEA8  00
00229  CEA9  52 56             .BYTE 'RVS ON',$00
00229  CEAF  00
00230  CEB0  48 4F             .BYTE 'HOME',$00
00230  CEB4  00
00231  CEB5  44 45 4C          .BYTE 'DEL',$00
00231  CEB8  00
00232  CEB9  52 4F             .BYTE 'ROUGE',$00
00232  CEBE  00
00233  CEBF  43 2E             .BYTE 'C.DROITE',$00
00233  CEC7  00
00234  CEC8  56 45             .BYTE 'VERT',$00
00234  CECC  00
00235  CECD  42 4C             .BYTE 'BLEU',$00
00235  CED1  00
00236  CED2  4D 41             .BYTE 'MAJUS.',$00
00236  CED8  00
00237  CED9  4E 4F             .BYTE 'NOIR',$00
00237  CEDD  00
00238  CEDE  43 2E             .BYTE 'C.HAUT',$00
00238  CEE4  00
00239  CEE5  52 56             .BYTE 'RVS OFF',$00
00239  CEEC  00
00240  CEED  43 4C 52          .BYTE 'CLR',$00
```

```
00240  CEF0  00
00241  CEF1  49 4E             .BYTE 'INST',$00
00241  CEF5  00
00242  CEF6  50 4F             .BYTE 'POURPRE',$00
00242  CEFD  00
00243  CEFE  43 2E             .BYTE 'C.GAUCHE',$00
00243  CF06  00
00244  CF07  4A 41             .BYTE 'JAUNE',$00
00244  CF0C  00
00245  CF0D  43 59             .BYTE 'CYAN',$00
00245  CF11  00
00246  CF12  4F 52             .BYTE 'ORANGE',$00
00246  CF18  00
00247  CF19  4F 52             .BYTE 'ORANGE L.',$00
00247  CF22  00
00248  CF23  52 4F             .BYTE 'ROSE',$00
00248  CF27  00
00249  CF28  43 59             .BYTE 'CYAN L.',$00
00249  CF2F  00
00250  CF30  50 4F             .BYTE 'POURPRE L.',$00
00250  CF3A  00
00251  CF3B  56 45             .BYTE 'VERT L.',$00
00251  CF42  00
00252  CF43  42 4C             .BYTE 'BLEU L.',$00
00252  CF4A  00
00253  CF4B  4A 41             .BYTE 'JAUNE L.',$00
00253  CF53  00
00254  CF54  46 31             .BYTE 'F1',$00
00254  CF56  00
00255  CF57  46 32             .BYTE 'F2',$00
00255  CF59  00
00256  CF5A  46 33             .BYTE 'F3',$00
00256  CF5C  00
00257  CF5D  46 34             .BYTE 'F4',$00
00257  CF5F  00
00258  CF60  46 35             .BYTE 'F5',$00
00258  CF62  00
00259  CF63  46 36             .BYTE 'F6',$00
00259  CF65  00
00260  CF66  46 37             .BYTE 'F7',$00
00260  CF68  00

00261  CF69  46 38             .BYTE 'F8',$00
00261  CF6B  00
00262  CF6C              ;
00263  CF6C              ;*****************************************************
00264  CF6C              ;     DEBUT DE LIGNE-SUITE SI N.DE CARACTERES A IMPRIMER >LINLEN
00265  CF6C  0D          NLIN  .BYTE $0D,$0A,$20,$20,$20,$20,$20,$20,$00
00265  CF6D  0A
00265  CF6E  20
00265  CF6F  20
00265  CF70  20
00265  CF71  20
00265  CF72  20
00265  CF73  20
00265  CF74  00
00266  CF75              ;
00267  CF75              ;*****************************************************
00268  CF75                   .END
```

LISTVARIABLES

Ce programme peut être ajouté (avec APPENDISK) à un autre programme.

Il liste à chaque frappe de la barre d'espace une variable du BASIC avec son type (entier, flottant, chaîne). Pour les chaînes de caractères, il liste la chaîne, donne son adresse et sa longueur. Les fonctions FN sont listées : adresse du texte de la fonction, adresse de la variable locale. Les tableaux sont donnés avec d'abord leurs dimensions, puis les valeurs de chaque élément. Les tableaux de chaînes donnent les chaînes, les adresses et les longueurs.

```
1 A=123:B=-321
2 A%=123:B%=-321
3 A$="COUCOU":B$=A$+" A TOUS"
4 DEFFNAB(B)=2^B+1
5 DIMAB(2,3,4),A$(5),B%(5):A$(0)="#"
6 FORI=0TO2:FORJ=0TO3:FORK=0TO4:AB(I,J,K)=100*I+10*J+K:NEXTK,J,I
7 FORI=1TO5:A$(I)=A$(I-1)+"#":B%(I)=B%(I)+2*I:NEXTI
62990 REM*******************************
62991 REM*        LISTVARIABLES        *
62992 REM*                             *
62993 REM*   ECRIT PAR P. BRUNET       *
62994 REM*    COPYRIGHT BCM 1983       *
62995 REM*******************************
62996 :
62997 REM ZA:ADRESSE DEBUT VARIABLES
62998 :
63000 PRINT"[BLANC]":ZA=PEEK(46)*256+PEEK(45)
63001 :
63003 REM ZB:ADRESSE DEBUT TABLEAUX
63005 :
63010 ZB=PEEK(48)*256+PEEK(47)
63011 :
63013 REM*******************************
63014 REM*  DEBUT BOUCLE PRINCIPALE     *
63015 REM*        VARIABLES            *
63016 REM*******************************
63017 :
63018 REM ZC:INDICE DE BOUCLE
63019 :
63020 FORZC=ZATOZB-1STEP7
63021 :
63022 REM ZD:SAISIE DES BITS 7 DES DEUX
63023 REM    PREMIERS CARACTERES.
63024 REM ZD=0 : VAR. FLOTTANTES
63025 REM ZD=1 : CHAINE DE CARACTERES
63026 REM ZD=2 : FONCTIONS
63027 REM ZD=3 : VAR. ENTIERES
63028 :
63030 ZD=-((PEEK(ZC)AND128)=128)*2-((PEEK(ZC+1)AND128)=128)
63031 :
63033 REM CHOIX DES DIF. ROUTINES
63034 REM SUIVANT LA VALEUR DE ZD
63036 :
63040 ONZD+1GOSUB63080,63140,63190,63210
63041 :
63043 REM TOUCHE PRESSEE ?
63045 :
63050 GETZA$:IFZA$=""THEN63050
63051 :
63053 REM*******************************
63054 REM*   FIN BOUCLE PRINCIPALE      *
63055 REM*******************************
```

```
63057 :
63060 PRINT:NEXTZC:GOTO63260
63061 :
63063 REM*******************************
63064 REM*       SOUS PROGRAMME        *
63065 REM*  IMPRESSION NOM VARIABLE    *
63066 REM*******************************
63067 :
63070 PRINTCHR$(PEEK(ZC)AND127)CHR$(PEEK(ZC+1)AND127);:RETURN
63071 :
63073 REM*******************************
63074 REM*       SOUS PROGRAMME        *
63075 REM*   VARIABLES FLOTTANTES      *
63076 REM*******************************
63077 :
63078 REM CALCUL DE L'EXPOSANT
63079 :
63080 ZE=PEEK(ZC+2)-128
63081 :
63083 REM CALCUL DE LA MANTISSE
63085 :
63090 ZJ=.5+(PEEK(ZC+3)AND127)/256+PEEK(ZC+4)/256^2+PEEK(ZC+5)/256^3+PE
      EK(ZC+6)/256^4
63091 :
63093 REM RECHERCHE DU SIGNE
63094 REM ZK=-1 :SIGNE NEGATIF
63095 REM ZK= 1 :SIGNE POSITIF
63096 :
63100 IFPEEK(ZC+3)>127THENZK=-1
63110 IFPEEK(ZC+3)<128THENZK=1
63111 :
63113 REM CALCUL DU NOMBRE
63114 REM FIN DU S.P. SI TABLEAU
63115 :
63120 ZJ=ZK*ZJ*(2^ZE):IFZS>0THENRETURN
63121 :
63123 REM AFFICHE LE NOMBRE
63124 :
63130 GOSUB63070:PRINT"=";ZJ:ZJ=0:PRINT"[C.HAUT][C.HAUT]":RETURN
63131 :
63132 REM*******************************
63133 REM*       SOUS PROGRAMME        *
63134 REM*   CHAINES DE CARACTERES     *
63135 REM*******************************
63136 :
63137 REM CALCULE ADRESSE DE LA CHAINE
63138 :
63140 ZE=PEEK(ZC+4)*256+PEEK(ZC+3)
63141 :
63143 REM CALCULE LONGEUR DE LA CHAINE
63144 REM FIN DU S.P. SI TABLEAU
63145 :
63150 ZG=PEEK(ZC+2):IFZS>0THENRETURN
63151 :
63153 REM AFFICHE LE NOM DE LA CHAINE
63154 REM SON ADRESSE ET SA LONGUEUR
63155 :
63160 GOSUB63070:PRINT"$:AD=";ZE;"L=";ZG
63161 :
63163 REM AFFICHE LE NOM DE LA CHAINE
63164 :
63170 GOSUB63070:PRINT"$=";
63171 :
63173 REM AFFICHE LA CHAINE
63174 :
63180 FORZF=ZETOZE+ZG-1:PRINTCHR$(PEEK(ZF));:NEXT:RETURN
63181 :
63182 REM*******************************
63183 REM*       SOUS PROGRAMME        *
63184 REM*         FONCTIONS           *
63185 REM*******************************
63186 :
63187 REM AFFICHE ADR. DE LA DEFINITION
```

```
63188 REM  DE LA FONCTION
63189 :
63190 PRINT"FN";:GOSUB63070:PRINT" :DEF="(PEEK(ZC+2)+PEEK(ZC+3)*256)
63191 :
63193 REM AFFICHE ADR. DE LA VARIABLE   63194 REM EMPLOYEE DANS LA FONC
      TION
63195 :
63200 PRINT"FN";:GOSUB63070:PRINT" :VAR="(PEEK(ZC+4)+PEEK(ZC+5)*256):PR
      INT"[C.HAUT][C.HAUT]":RETURN
63201 :
63202 REM******************************
63203 REM*       SOUS PROGRAMME       *
63204 REM*    VARIABLES ENTIERES      *
63205 REM******************************
63206 :
63207 REM ZK= 1: NOMBRE NEGATIF
63208 REM ZK= 0: NOMBRE POSITIF
63209 :
63210 ZK=-((PEEK(ZC+2)AND128)=128)
63211 :
63213 REM CALCUL SI NOMBRE NEGATIF
63214 :
63220 IFZKTHENZJ=PEEK(ZC+2)*256+PEEK(ZC+3)-65536:GOTO63240
63221 :
63223 REM CALCUL SI NOMBRE POSITIF
63224 :
63230 ZJ=PEEK(ZC+2)*256+PEEK(ZC+3)
63231 :
63233 REM RETOUR SI TABLEAU
63234 :
63240 IFZS>0THENRETURN
63241 :
63243 REM AFFICHAGE DU NOMBRE
63244 :
63250 GOSUB63070:PRINT"%="ZJ:ZJ=0:PRINT"[C.HAUT][C.HAUT]":RETURN
63251 :
63252 REM******************************
63253 REM*   DEBUT DU PGM TABLEAUX    *
63254 REM******************************
63255 :
63256 REM DEFINITION DE TOUTES LES
63257 REM VARIABLES EMPLOYEES DANS LE
63258 REM PROGRAMME POUR EVITER LA
63259 REM MODIFICATION DES POINTEURS
63260 REM DURANT L'EXECUTION DU PGM
63261 :
63265 ZB$="":ZM=0:ZN=0:ZL=1:ZO=0:ZQ=0:ZS=0:DIMZP(4),ZQ(4)
63266 :
63267 REM ZB: ADRESSE DEBUT TABLEAUX
63268 :
63270 ZB=PEEK(48)*256+PEEK(47)
63271 :
63273 REM ZM: ADRESSE FIN DES TABLEAUX
63274 REM ZN: ADRESSE DEBUT DU TABLEAU
63275 :
63280 ZM=PEEK(50)*256+PEEK(49):ZN=ZB
63281 :
63283 REM TESTE SI FIN DES TABLEAUX
63284 :
63290 IFZN=ZMTHENEND
63291 :
63293 REM ZO: NOMBRE D'INDICES
63294 :
63300 ZO=PEEK(ZN+4)
63301 :
63303 REM CALCUL DES VALEURS MAXI
63304 REM DES INDICES DU TABLEAU
63305 :
63310 FORZF=1TOZO
63320 ZP(ZF)=PEEK(ZN+4+2*ZF)+PEEK(ZN+3+2*ZF)*256-1:ZQ(ZF)=ZP(ZF):NEXT
63321 :
63323 REM CALCUL DU TYPE DE TABLEAU
63324 REM ZD=0 :FLOTTANT
```

```
63325 REM ZD=1 :CHAINE
63326 REM ZD=3 :ENTIER
63327 :
63330 ZD=-((PEEK(ZN)AND128)=128)*2-((PEEK(ZN+1)AND128)=128)
63331 :
63333 REM ZB$: NOM DU TABLEAU
63334 :
63340 ZB$=CHR$(PEEK(ZN)AND127)+CHR$(PEEK(ZN+1)AND127)
63341 :
63343 REM FIN DE NOM SI CHAINE
63344 :
63350 IFZD=1THENZB$=ZB$+"$"
63351 :
63353 REM FIN DE NOM SI ENTIER
63354 :
63360 IFZD=3THENZB$=ZB$+"%"
63361 :
63363 REM AFFICHAGE DU NOM DU TABLEAU
63364 REM AVEC SES INDICES MAXIS
63365 :
63370 PRINT" DIM ";:GOSUB63580:PRINT" "
63371 :
63373 REM CALCUL DE ZL: NOMBRE
63374 REM D'ELEMENTS DU TABLEAU
63375 :
63380 FORZF=1TOZO:ZL=ZL*(ZP(ZF)+1):NEXT
63381 :
63383 REM INITIALISATION DES INDICES
63384 :
63390 FORZF=1TOZO:ZP(ZF)=0:NEXT
63391 :
63393 REM ZC:PARAMETRE D'ADRESSE POUR
63394 REM                          APPEL
63395 :
63400 ZC=ZN+(2*ZO)+3
63401 :
63402 REM*******************************
63403 REM*  DEBUT BOUCLE PRINCIPALE    *
63404 REM*          TABLEAUX           *
63405 REM*******************************
63406 :
63410 FORZS=1TOZL
63411 :
63413 REM AFFICHE LE NOM DE L'ELEMENT
63414 :
63420 GOSUB63590
63421 :
63423 REM CALCUL DE LA VARIABLE
63424 REM CALCUL DE ZC SI FLOTTANT
63425 :
63430 IFZD=0THENGOSUB63080:ZC=ZC+5
63431 :
63433 REM CALCUL ADRESSE ET LONGUEUR
63434 REM DE LA CHAINE, LES IMPRIME,
63435 REM ET CALCULE ZC
63436 :
63440 IFZD=1THENGOSUB63140:GOSUB63550:ZC=ZC+3
63441 :
63443 REM CALCUL DE LA VARIABLE
63444 REM CALCUL DE ZC SI ENTIER
63445 :
63450 IFZD=3THENGOSUB63210:ZC=ZC+2
63451 :
63453 REM AFFICHE LE NOMBRE
63454 :
63460 IFZD<>1THENPRINT"=";ZJ
63461 :
63463 REM*******************************
63464 REM*   DEBUT BOUCLE DE CALCUL    *
63465 REM*        DES INDICES          *
63466 REM*******************************
63467 :
63470 FORZF=ZOTO1STEP-1
```

```
63480 ZP(ZF)=ZP(ZF)+1
63490 IFZP(ZF)>ZQ(ZF)THENZP(ZF)=0:GOTO63510
63500 ZF=1
63501 :
63503 REM*******************************
63504 REM*  FIN DE BOUCLE DE CALCUL   *
63505 REM*******************************
63506 :
63510 NEXTZF
63511 :
63512 REM TOUCHE PRESSEE ?
63513 :
63520 GETZA$:IFZA$=""THEN63520
63521 :
63523 REM*******************************
63524 REM*  FIN DE BOUCLE PRINCIPALE  *
63525 REM*******************************
63526 :
63530 NEXTZS
63531 :
63533 REM CALCUL DE L'ADRESSE DU
63534 REM TABLEAU SUIVANT
63535 :
63540 ZN=ZC+2:ZL=1:GOTO63290
63541 :
63543 REM*******************************
63544 REM*       SOUS PROGRAMME        *
63545 REM*    AFFICHAGE DE CHAINE      *
63546 REM*******************************
63547 :
63550 PRINT":AD="ZE;"L="ZG
63560 GOSUB63590:PRINT"=";
63570 FORZF=ZETOZE+ZG-1:PRINTCHR$(PEEK(ZF));:NEXT:PRINT" ":RETURN
63571 :
63573 REM*******************************
63574 REM*       SOUS PROGRAMME        *
63575 REM*     AFFICHAGE DU NOM        *
63576 REM*       DE L'ELEMENT          *
63577 REM*******************************
63578 :
63580 ZB$=ZB$+"("
63590 PRINTZB$;
63600 FORZF=ZOTO1STEP-1:PRINTZP(ZF)"[C.GAUCHE],";:NEXT:PRINT"[C.GAUCHE]
      )";:RETURN

READY.
```

LUTIN.

Démonstration d'une façon simple de stocker des images de lutins en DATA et de programmer le VIC-II pour les faire apparaître.

```
10 PRINT"[CLR][BLANC]LUTIN":REM  PAR B.MICHEL LE LIVRE DU 64
20 X=210:Y=125:REM COORDONNEES
40 CL=1  :       REM COULEUR=BLANC
50 N=0   :       REM LUTIN NUMERO 0-7
60 BL=14:        REM LUTIN DANS BLOC 13-15
70 GOSUB 1000: REM PROGRAMME LE VIC-II
80 GOSUB 2000: REM DESSINNE LE LUTIN
99 END
100 REM*********************************
1000 REM PROGRAMME LE LUTIN DANS LE VIC-II
1140 POKE53277,PEEK(53277)OR(2^N):REM EXP.HORIZ.
1150 POKE53271,PEEK(53271)OR(2^N):REM EXP.VERTIC.
1160 POKE53276,PEEK(53276)AND(255-2^N):REM MONOCHROME
1190 POKE53287+N,CL:REM COULEUR
1200 POKE53264,PEEK(53264)AND255-(2^N)OR(2^N)*-(X>255):REM COORD.X 9EME
       BIT
```

```
1210 POKE53248+2*N,(XAND255):REM COORD X 8 BITS
1220 POKE53249+2*N,Y:REM COORD Y
1230 POKE2040+N,BL:REM POINTEUR DU LUTIN N=BL
1240 POKE53269,PEEK(53269)OR(2^N):REM AUTORISE LUTIN N
1999 RETURN
2000 REM DESSINNE LE LUTIN
2130 FOR I=0TO20:READ A$:FOR K=0TO2:T=0:FOR J=0TO7:B=0
2131 IF MID$(A$,J+K*8+1,1)="Q"THEN B=1
2135 T=T*2+B:NEXT J:POKE64*BL+I*3+K,T:NEXTK,I
2999 RETURN
10000 DATA"                QQ              "
10010 DATA"              QQQQQQQ           "
10020 DATA"              Q QQQ             "
10030 DATA"              QQQQQQ            "
10040 DATA"                QQQQQ           "
10050 DATA"                 QQ             "
10060 DATA"      Q        QQQQQQ           "
10070 DATA"    QQQ      QQQQQQQQ           "
10080 DATA"   QQQ     QQQQQQQQQQQ          "
10090 DATA"   QQQQ  QQQQQQQQQQQQ           "
10100 DATA"  Q QQQQQQQQQQQQQQQ             "
10110 DATA"      QQQQ QQQQQQQQQ           "
10120 DATA"       QQQ QQ Q Q QQ           "
10130 DATA"         QQQ QQQQQQQ           "
10140 DATA"         Q  QQQQQQ             "
10150 DATA"           QQQQQQ              "
10160 DATA"              QQQ              "
10170 DATA"              QQQ              "
10180 DATA"              QQQ              "
10190 DATA"             QQQQ              "
10200 DATA"            QQQQQQ             "
55555 SAVE"@0:LUTIN",8

READY.
```

MELODIE.

Mélodie paramétrable où les volumes, ADSR, formes d'onde, largeurs d'impulsion sont modifiables à volonté, ce qui permet de juger de l'effet de chacun de ces paramètres.

```
10 REM MELODIE
11 :
20 S=54272:GOSUB1000
30 VL=15:INPUT"VOLUME(0-15)";VL
40 AR=16:INPUT"FORME D'ONDE(16,32,64,80,96,128)";AR:DE=AR+1
50 IF(AR AND64)=64THENPH=8:INPUT"IMPULSION(0-15)";PH
60 T=3:INPUT"TEMPO(EN 1/32 DE SECONDES)";T:DU=T*30
65 AA=8:DD=6:SS=12:RR=6
70 INPUT"A,D,S,R(4 VALEURS ENTRE 0 ET 15)";AA,DD,SS,RR
80 AD=AA*16+DD:SR=SS*16+RR
110 RESTORE:POKE S+5,AD:POKE S+6,SR
115 POKE S+24,VL:POKE S+3,PH
120 READ N,D:D=DU*D:POKE S+1,N:POKES+4,DE
130 FOR TD=0TOD:NEXT:POKES+4,AR
140 FOR TD=0TO10*T:NEXT:IF N<>0 GOTO 120
150 PRINT:PRINT"VOL,FORME,TEMPO=";VL;AR;T
160 IF(AR AND64)=64THEN PRINT:PRINT"IMPULSION";PH
170 PRINT:PRINT"ADSR=";AA;DD;SS;RR
180 PRINT:INPUT"ENCORE(1),MODIFIER(2),FIN(3)";A
190 ON A GOTO 110,30,999
200 DATA25, 4,  28, 4,  25, 4 ,  25, 4
210 DATA25, 2,  28, 2,  32,12 ,  25, 4
220 DATA28, 4,  19, 4,  19, 2 ,  19, 2
```

```
230 DATA21, 1,  24, 1,  25, 4 ,  24, 2
240 DATA19,4 ,   0, 0
999 GOSUB1000:END
1000 REM SILENCE
1010 S=54272:FOR T=S TO S+24:POKE T,0:NEXT
1999 RETURN
55555 SAVE"@0:MELODIE",8

READY.
```

MODES MIXTES ET MODES MIXTES-SRC

Programme en langage machine et démonstration en BASIC des modes mixtes. Voir le chapître 3 à ce sujet.

```
10 REM MODES GRAPHIQUES MELANGES
11 REM MODE1 :CARACTERES
12 REM MODE7 :BITMAP MULTICOLORE
13 REM LIGNE-FRONTIERE MODIFIABLE
20 :
30 FOR I=49152TO49283:READ A:CS=CS+A:POKE I,A:NEXT
31 IF CS<>15955THEN STOP:REM MODES MIXTES
32 CS=0
33 FOR I=49312TO49344:READ A:CS=CS+A:POKE I,A:NEXT
34 IF CS<> 3356THEN STOP:REM USR-PEEK
35 CS=0
36 FOR I=49360TO49457:READ A:CS=CS+A:POKE I,A:NEXT
37 IF CS<>12821THEN STOP:REM EFFACEMENT BITMAP
38 BASE=57344:REM ADRESSE DU BITMAP
39 REM EMPLOYER USR ET NON PEEK !!!!!
40 POKE785,160:POKE786,192:REM VECTEUR D'USR
41 REM LES LIGNES 40 A 199 SERVENT UNIQUEMENT POUR L'EXEMPLE
42 PRINT"[CLR][NOIR][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C.BAS][C
      .BAS][C.BAS][C.BAS][C.BAS][RVS ON]F1[RVS OFF] POUR FAIRE MONTER L
      A LIMITE"
43 PRINT"[RVS ON]F3[RVS OFF] POUR LA FAIRE DESCENDRE"
44 PRINT"[RVS ON]SYS49152[RVS OFF] POUR RELANCER APRES STOP/RESTORE"
45 PRINT"A L'ARRET DU PROGRAMME, LE MODE MIXTE CONTINUE"
46 PRINT"STOP/RESTORE AVANT TOUT ACCES A UN PERIPHERIQUE"
49 SYS49152:REM DEMARRE LE MODE MIXTE
50 SYS 49363:REM EFFACE BITMAP,ECRAN ET RAM COULEUR
55 A=113:REM NUMERO DE LIGNE-FRONTIERE ENTRE MODES
57 FORI=0TO3:B(I)=4^I:NEXT
58 X=0
59 REM ********************** BOUCLE PRINCIPALE
91 POKE49268,  1 :REM  COULEUR FOND BITMAP
92 POKE49269,  5 :REM COULEUR FOND TEXTE
93 POKE49276, 14 :REM COULEUR BORD BITMAP
94 POKE49277,  7 :REM COULEUR BORD TEXTE
95 REM POUR PARTIE SUPERIEURE ECRAN:
96 REM POKE 49360,VALEUR DE REMPLISSAGE RAM COULEUR
97 REM POKE 49361,VALEUR DE REMPLISSAGE ECRAN TEXTE
98 REM POKE 49362,VALEUR DE REMPLISSAGE BITMAP
100 Y2=30:FORX=0TO159:REM SINUS
110 GOSUB300:REM TEST CLAVIER
120 Y1=Y2
130 Y2=14+SIN(X/2)*(1-ABS(80-X)/80)
140 Y2=INT(30*Y2)
145 FOR Y=Y1 TO Y2 STEP SGN(Y2-Y1)
150 W=3:GOSUB 200
160 NEXT Y,X
170 FORCH=15TO25
180 FORY=24TO29:W= 85:GOSUB400:NEXT Y
182 FORY=30TO35:W= 00:GOSUB400:NEXT Y
184 FORY=36TO41:W=170:GOSUB400:NEXT Y
186 FORY=42TO47:W=255:GOSUB400:NEXT Y
```

```
190 NEXT CH
198 GOSUB300:GOTO198:REM ****** FIN DE PROGRAMME
199 REM *********************************
200 REM ECRIT EN MODE 7 (VALEUR W=0,1,2 OU 3)
201 REM *********************************
210 CH=INT(X/4):RO=INT(Y/8):LN=Y AND7
220 BY=BASE+RO*320+8*CH+LN:BI=ABS(3-(X AND3))
298 POKE BY,USR(BY)OR(B(BI)*W):RETURN
299 REM *********************************
300 REM TEST DES TOUCHES F1 ET F3
301 REM *********************************
310 GETA$:IFA$=CHR$(134)THENA=A+8:IFA>255THENA=255
320 IFA$=CHR$(133)THENA=A-8:IFA<9THENA=9
330 POKE49266,A:RETURN
399 REM *********************************
400 REM TRACE UN OCTET MULTICOLORE (W=0 A 255) EN (CH,Y)
401 REM *********************************
402 REM CH VARIE DE 0 A 39
410 RO=INT(Y/8):LN=Y AND7
420 BY=BASE+RO*320+8*CH+LN
430 POKE BY,W:GOSUB300:RETURN
499 REM *********************************
30000 REM N. DE LIGNE = ADRESSE REELLE
49152 DATA120,169,127,141,013,220
49158 DATA169,001,141,026,208,169
49164 DATA001,133,251,169,027,141
49170 DATA020,003,169,192,141,021
49176 DATA003,088,096,173,025,208
49182 DATA141,025,208,041,001,240
49188 DATA071,198,251,016,004,169
49194 DATA001,133,251,166,251,189
49200 DATA116,192,141,033,208,189
49206 DATA126,192,141,000,221,189
49212 DATA118,192,141,017,208,189
49218 DATA120,192,141,022,208,189
49224 DATA124,192,141,032,208,189
49230 DATA128,192,141,034,208,189
49236 DATA130,192,141,035,208,189
49242 DATA122,192,141,024,208,189
49248 DATA114,192,141,018,208,173
49254 DATA013,220,041,001,240,003
49260 DATA076,188,254,076,049,234
49266 DATA033,000,015,005,059,027
49272 DATA216,200,056,021,015,007
49278 DATA148,151,001,002,002,001
49283 REM CHECKSUM= 15955 DE 49152 A 49283
49312 DATA165,021,072,165,020,072,032,247
49320 DATA183,160,000,120,169,053,133,001
49328 DATA177,020,168,169,055,133,001,088
49336 DATA104,133,020,104,133,021,076,162
49344 DATA179
49345 REM CHECKSUM= 3356 DE 49312 A 49344
49360 DATA000,038,000,169,000,133,098,169
49368 DATA216,133,099,169,000,133,100,169
49376 DATA220,133,101,173,208,192,160,000
49384 DATA032,030,193,208,251,169,000,133
49392 DATA098,169,224,133,099,169,000,133
49400 DATA100,169,000,133,101,173,210,192
49408 DATA032,030,193,208,251,169,000,133
49416 DATA098,169,204,133,099,169,000,133
49424 DATA100,169,208,133,101,173,209,192
49432 DATA032,030,193,208,251,096,145,098
49440 DATA230,098,208,002,230,099,166,099
49448 DATA228,101,208,004,166,098,228,100
49456 DATA096,169
49457 REM CHECKSUM= 12821 DE 49360 A 49457
55554 END
55555 SAVE"@0:MODES MIXTES",8

READY.
```

```
00001  0000             ;GRAPHIQUES MULTI-MODES
00002  0000             ;**************************************
00003  0000             ;ATTENTION , DANS CE LISTING:
00004  0000             ;'PLUS GRAND QUE' EST DONNE PAR :'"'
00005  0000             ;'PLUS PETIT QUE' EST DONNE PAR :'^'
00006  0000             ;**************************************
00007  0000             ;
00008  0000                   IRQVEC = $0314  ;VECTEUR D'INTERR. EN RAM
00009  0000                   CTR1   = $D011  ;VIC-II CONTROLE 1
00010  0000                   RASTER = $D012  ;REGISTRE DE BALAYAGE
00011  0000                   CTR2   = $D016  ;VIC-II CONTROLE 2
00012  0000                   ADRES  = $D018  ;ADRESSE ECRAN POUR VIC-II
00013  0000                   VICIRQ = $D019  ;VIC-II CONTROLE D'INTERRUPTION
00014  0000                   IRQMSK = $D01A  ;FLAGS D'INTERRUPTIONS
00015  0000                   BORD   = $D020  ;COULEUR DU BORD
00016  0000                   COFOND = $D021  ;VIC-II COULEUR DU FOND
00017  0000                   COL2   = $D022  ;VIC-II COULEUR AUX. NUMERO 2
00018  0000                   COL3   = $D023  ;VIC-II COULEUR AUX. NUMERO 3
00019  0000                   CIAICR = $DC0D  ;REG. D'INTERRUPTION DU CIA #2
00020  0000                   BANC   = $DD00  ;VIA#2 CONTROLE DU BANC MEMOIRE DU VIC-II
00021  0000                   CBMIRQ = $EA31  ;ROUTINE D'INTERRUPTION NORMALE
00022  0000                   FINIRQ = $FEBC  ;SORTIE D'INTERRUPTION NORMALE
00023  0000                                   ;
00024  0000                   NMODE  = $01    ;NOMBRE DE MODES DESIRES SIMULTANEMENT
00025  0000                                   ;01 POUR 2 MODES, 02 POUR 3 MODES, ETC...
00026  0000                   COMPTE = $FB    ;NUMERO DE LA PROCHAINE INTERRUPTION
00027  0000                                   ;
00028  0000                   * = $C000
00029  C000             ;
00030  C000  78         INIT  SEI              ;INTERDIT LES INTERRUPTIONS
00031  C001  A9 7F            LDA #$7F        ;ARRETE LES INT. DU CIA #2
00032  C003  8D 0D DC         STA CIAICR      ;
00033  C006  A9 01            LDA #$01        ;AUTORISE LES INT. DE BALAYAGE
00034  C008  8D 1A D0         STA IRQMSK      ;
00035  C00B  A9 01            LDA #NMODE      ;VALEUR DE DEPART DU NUMERO D'INTERR.
00036  C00D  85 FB            STA COMPTE      ;
00037  C00F  A9 1B            LDA #^MIX       ;MODIFIE LE VECTEUR
00038  C011  8D 14 03         STA IRQVEC      ;D'INTERRUPTION
00039  C014  A9 C0            LDA #"MIX       ;
00040  C016  8D 15 03         STA IRQVEC+1    ;
00041  C019  58               CLI             ;
00042  C01A  60               RTS             ;
00043  C01B             ;
00044  C01B             ;
00045  C01B  AD 19 D0  MIX    LDA VICIRQ      ;EFFACE LA DEM/E D'5ERR.
00046  C01E  8D 19 D0         STA VICIRQ      ;EN COURS
00047  C021  29 01            AND #$01
00048  C023  F0 47            BEQ INTRT       ;SI PAS BALAYAGE —> FIN
00049  C025  C6 FB            DEC COMPTE      ;SINON LE COMPTER
00050  C027  10 04            BPL RAS1        ;SI LE DERNIER, REDEMARRE COMPTAGE
00051  C029  A9 01            LDA #NMODE
00052  C02B  85 FB            STA COMPTE
00053  C02D             ;
00054  C02D  A6 FB      RAS1  LDX COMPTE      ;POSITION DANS LA TABLE
00055  C02F  BD 74 C0         LDA COLTBL,X    ;COULEUR DE FOND
00056  C032  8D 21 D0         STA COFOND      ;
00057  C035  BD 7E C0         LDA BANTBL,X    ;BANC MEMOIRE VIC-II
00058  C038  8D 00 DD         STA BANC        ;
00059  C03B  BD 76 C0         LDA CR1TBL,X    ;VIC-II CONTROLE 1
00060  C03E  8D 11 D0         STA CTR1        ;
00061  C041  BD 78 C0         LDA CR2TBL,X    ;VIC-II CONTROLE 2
00062  C044  8D 16 D0         STA CTR2        ;
00063  C047  BD 7C C0         LDA BORTBL,X    ;COULEUR DU BORD
00064  C04A  8D 20 D0         STA BORD        ;
00065  C04D  BD 80 C0         LDA CO2TBL,X    ;COULEUR AUX NUMERO 2
00066  C050  8D 22 D0         STA COL2        ;
00067  C053  BD 82 C0         LDA CO3TBL,X    ;COULEUR AUX NUMERO 3
00068  C056  8D 23 D0         STA COL3        ;
00069  C059  BD 7A C0         LDA ADRTBL,X    ;ADRESSES ECRAN ET GEN. DE CARACTERES
00070  C05C  8D 18 D0         STA ADRES       ;
00071  C05F  BD 72 C0         LDA RASTBL,X    ;N.LIGNE DE BALAYAGE
00072  C062                                   ;OU ON VA CHANGER DE MODE
00073  C062  8D 12 D0         STA RASTER      ;
```

```
00074  C065             ;
00075  C065             ; IL NE RESTE PLUS QU'A S'OCCUPER DE LA ROUTINE CLAVIER-HORLOGE
00076  C065  AD 0D DC        LDA CIAICR      ;Y A-T-IL UNE DEMANDE
00077  C068  29 01           AND #$01        ;
00078  C06A  F0 03           BEQ CLAHOR      ;SINON,
00079  C06C             ;
00080  C06C  4C BC FE   INTRT  JMP FINIRQ     ;SORTIE D'INTERR. EN ROM
00081  C06F             ;
00082  C06F  4C 31 EA   CLAHOR JMP OEMIRQ     ;SI OUI, GERE CLAVIER ET HORLOGE
00083  C072             ;
00084  C072             ; IL FAUT (NMODE+1) VALEURS PAR LIGNE DANS LA TABLE
00085  C072  1E         RASTBL .BYT 030,170   ;LIGNE DEBUT DE MODE
00085  C073  AA
00086  C074  0F         COLTBL .BYT 015,006   ;COULEUR DE FOND
00086  C075  06
00087  C076  3B         CR1TBL .BYT 059,027   ;MODES ETENDUS,BITMAP,SCROLL Y
00087  C077  1B
00088  C078  D8         CR2TBL .BYT 216,200   ;MODE MULTICOLORE, SCROLL X
00088  C079  C8
00089  C07A  78         ADRTBL .BYT 120,021   ;ADRESSE ECRAN+GEN. DE CARACTERES
00089  C07B  15
00090  C07C  0F         BORTBL .BYT 015,014   ;COULEUR DE BORD
00090  C07D  0E
00091  C07E  94         BANTBL .BYT 148,151   ;BANC MEMOIRE DU VIC-II
00091  C07F  97
00092  C080  01         CO2TBL .BYT 001,002   ;POUR LE MODE COULEUR ETENDU
00092  C081  02
00093  C082  02         CO3TBL .BYT 002,001   ;POUR LE MODE COULEUR ETENDU
00093  C083  01
00094  C084             .END
```

MON

MON est une version modifiée de EXTRAMON, un programme moniteur conçu à l'origine pour le PET par un groupe d'utilisateurs canadiens.. Pour entrer "MON" dans le 64, il faut un autre moniteur. Le programme ci-dessous, "PETITMON" est suffisant pour cela. De plus il calcule les sommes de vérification. Les sommes de tous les blocs de 256 octets de "MON" sont données ci-dessous:

Adresse de bloc	somme de vérification
2049 à 2304	19990
2305 à 2560	30047
2561 à 2816	30967
2817 à 3072	29706
3073 à 3328	30372
3329 à 3584	29406
3585 à 3840	29615
3841 à 4096	31031
4097 à 4352	25983
4353 à 4586	22141 *

* Le dernier bloc est plus court que 256 octets.

Mode d'emploi de MON :

LOAD puis RUN : installation du moniteur qui se reloge seul en haut de mémoire et se protège du BASIC. On l'appelle ensuite par SYS 2 ou SYS 8 (SYS 38893 après STOP-RESTORE ou RESET).

Instructions de MON :

.A assemble une instruction en langage machine avec la syntaxe COMMODORE. Exemple : A 5000 LDA $15

.D désassemble à partir d'une adresse et jusqu'à une autre adresse ou la fin de l'écran.

```
Exemple : .D 5000
          .D 5000 5100
```

.F remplissage de la mémoire avec une valeur donnée.

```
Exemple : .F 5000 5FFF 00
```

.G exécute un programme en langage machine.

```
Exemple : .G 5000
```

.H recherche une chaîne d'octets en mémoire d'une première à une dernière adresse.

```
Exemple :     .H 5000 5FFF 00          cherche 00
              .H A000 BFFF 'READ'      cherche 'READ'
              .H A000 FFFF 20D2   cherche $20, $D2
```

.L charge une zone mémoire de la cassette ou du disque.Ne modifie aucun pointeur du BASIC.

```
Exemple : .L"MON",08
          .L"MON",01
```

.M affiche le contenu d'une zone mémoire d'une première à une dernière adresse.

```
Exemple : .M A000 A080
```

.R affiche le contenu des registres du 6510.

```
Exemple : .R
```

.S sauve une zone mémoire sur cassette d'une première à une dernière adresse.

```
Exemple : .S"MON",08,0801,11EB
```

Il faut donner comme adresse de fin, l'adresse du dernier octet + 1.

.T transfère une zone mémoire: adresse de début - adresse de fin - adresse de destination.

Exemple : T A000 BFFF 5000

.X retour au BASIC.

Exemple : .X

Pour chaque affichage du 'MON', on peut utiliser l'éditeur d'écran pour modifier les valeurs affichées.

```
M: 0801 2A 08 64 00 99 22 93 05
M: 0809 12 4D 4F 4E 49 54 45 55
M: 0811 52 20 36 35 30 32 20 41
M: 0819 43 43 45 53 20 50 41 52
M: 0821 20 53 59 53 38 20 11 22
M: 0829 00 40 08 6E 00 97 35 33
M: 0831 32 38 30 2C 37 3A 97 35
M: 0839 33 32 38 31 2C 35 00 5E
M: 0841 08 82 00 9E 28 C2 28 34
M: 0849 33 29 AA 32 35 36 AC C2
M: 0851 28 34 34 29 AA 31 32 37
M: 0859 29 00 61 08 8C 00 A2 00
M: 0861 00 00 00 00 00 00 00 00
M: 0869 00 00 00 00 00 00 00 00
M: 0871 00 00 00 00 00 00 00 00
M: 0879 00 00 00 00 00 00 00 A5
M: 0881 2D 85 22 A5 2E 85 23 A5
M: 0889 37 85 24 A5 38 85 25 A0
M: 0891 00 A5 22 D0 02 C6 23 C6
M: 0899 22 B1 22 D0 3C A5 22 D0
M: 08A1 02 C6 23 C6 22 B1 22 F0
M: 08A9 21 85 26 A5 22 D0 02 C6
M: 08B1 23 C6 22 B1 22 18 65 24
M: 08B9 AA A5 26 65 25 48 A5 37
M: 08C1 D0 02 C6 38 C6 37 68 91
M: 08C9 37 8A 48 A5 37 D0 02 C6
M: 08D1 38 C6 37 68 91 37 18 90
M: 08D9 B6 C9 4F D0 ED A5 37 85
M: 08E1 33 A5 38 85 34 6C 37 00
M: 08E9 4F 4F 4F 4F AD E6 FF 00
M: 08F1 8D 16 03 AD E7 FF 00 8D
M: 08F9 17 03 A9 80 20 90 FF 00
M: 0901 00 D8 68 8D 3E 02 68 8D
M: 0909 3D 02 68 8D 3C 02 68 8D
M: 0911 3B 02 68 AA 68 A8 38 8A
M: 0919 E9 02 8D 3A 02 98 E9 00
M: 0921 00 8D 39 02 BA 8E 3F 02
M: 0929 20 57 FD 00 A2 42 A9 2A
M: 0931 20 57 FA 00 A9 52 D0 34
M: 0939 E6 C1 D0 06 E6 C2 D0 02
M: 0941 E6 26 60 20 CF FF C9 0D
M: 0949 D0 F8 68 68 A9 90 20 D2
M: 0951 FF A9 00 00 85 26 A2 0D
M: 0959 A9 2E 20 57 FA 00 A9 05
M: 0961 20 D2 FF 20 3E F8 00 C9
M: 0969 2E F0 F9 C9 20 F0 F5 A2
M: 0971 0E DD B7 FF 00 D0 0C 8A
M: 0979 0A AA BD C7 FF 00 48 BD
M: 0981 C6 FF 00 48 60 CA 10 EC
M: 0989 4C ED FA 00 A5 C1 8D 3A
M: 0991 02 A5 C2 8D 39 02 60 A9
M: 0999 08 85 1D A0 00 00 20 54
M: 09A1 FD 00 B1 C1 20 48 FA 00
```

```
M: 09A9 20 33 F8 00 C6 1D D0 F1
M: 09B1 60 20 88 FA 00 90 0B A2
M: 09B9 00 00 81 C1 C1 C1 F0 03
M: 09C1 4C ED FA 00 20 33 F8 00
M: 09C9 C6 1D 60 A9 3B 85 C1 A9
M: 09D1 02 85 C2 A9 05 60 98 48
M: 09D9 20 57 FD 00 68 A2 2E 4C
M: 09E1 57 FA 00 A9 90 20 D2 FF
M: 09E9 A2 00 00 BD EA FF 00 20
M: 09F1 D2 FF E8 E0 16 D0 F5 A0
M: 09F9 3B 20 C2 F8 00 AD 39 02
M: 0A01 20 48 FA 00 AD 3A 02 20
M: 0A09 48 FA 00 20 B7 F8 00 20
M: 0A11 8D F8 00 F0 5C 20 3E F8
M: 0A19 00 20 79 FA 00 90 33 20
M: 0A21 69 FA 00 20 3E F8 00 20
M: 0A29 79 FA 00 90 28 20 69 FA
M: 0A31 00 A9 90 20 D2 FF 20 E1
M: 0A39 FF F0 3C A6 26 D0 38 A5
M: 0A41 C3 C5 C1 A5 C4 E5 C2 90
M: 0A49 2E A0 3A 20 C2 F8 00 20
M: 0A51 41 FA 00 20 8B F8 00 F0
M: 0A59 E0 4C ED FA 00 20 79 FA
M: 0A61 00 90 03 20 80 F8 00 20
M: 0A69 B7 F8 00 D0 07 20 79 FA
M: 0A71 00 90 EB A9 08 85 1D 20
M: 0A79 3E F8 00 20 A1 F8 00 D0
M: 0A81 F8 4C 47 F8 00 20 CF FF
M: 0A89 C9 0D F0 0C C9 20 D0 D1
M: 0A91 20 79 FA 00 90 03 20 80
M: 0A99 F8 00 A9 90 20 D2 FF AE
M: 0AA1 3F 02 9A 78 AD 39 02 48
M: 0AA9 AD 3A 02 48 AD 3B 02 48
M: 0AB1 AD 3C 02 AE 3D 02 AC 3E
M: 0AB9 02 40 A9 90 20 D2 FF AE
M: 0AC1 3F 02 9A 6C 02 A0 A0 01
M: 0AC9 84 BA 84 B9 88 84 B7 84
M: 0AD1 90 84 93 A9 40 85 BB A9
M: 0AD9 02 85 BC 20 CF FF C9 20
M: 0AE1 F0 F9 C9 0D F0 38 C9 22
M: 0AE9 D0 14 20 CF FF C9 22 F0
M: 0AF1 10 C9 0D F0 29 91 BB E6
M: 0AF9 B7 C8 C0 10 D0 EC 4C ED
M: 0B01 FA 00 20 CF FF C9 0D F0
M: 0B09 16 C9 2C D0 DC 20 88 FA
M: 0B11 00 29 0F F0 E9 C9 03 F0
M: 0B19 E5 85 BA 20 CF FF C9 0D
M: 0B21 60 6C 30 03 6C 32 03 20
M: 0B29 96 F9 00 D0 D4 A9 90 20
M: 0B31 D2 FF A9 00 00 20 EF F9
M: 0B39 00 A5 90 29 10 D0 C4 4C
M: 0B41 47 F8 00 20 96 F9 00 C9
M: 0B49 2C D0 BA 20 79 FA 00 20
M: 0B51 69 FA 00 20 CF FF C9 2C
M: 0B59 D0 AD 20 79 FA 00 A5 C1
M: 0B61 85 AE A5 C2 85 AF 20 69
M: 0B69 FA 00 20 CF FF C9 0D D0
M: 0B71 98 A9 90 20 D2 FF 20 F2
M: 0B79 F9 00 4C 47 F8 00 A5 C2
M: 0B81 20 48 FA 00 A5 C1 48 4A
M: 0B89 4A 4A 4A 20 60 FA 00 AA
M: 0B91 68 29 0F 20 60 FA 00 48
M: 0B99 8A 20 D2 FF 68 4C D2 FF
M: 0BA1 09 30 C9 3A 90 02 69 06
M: 0BA9 60 A2 02 B5 C0 48 B5 C2
M: 0BB1 95 C0 68 95 C2 CA D0 F3
M: 0BB9 60 20 88 FA 00 90 02 85
M: 0BC1 C2 20 88 FA 00 90 02 85
M: 0BC9 C1 60 A9 00 00 85 2A 20
M: 0BD1 3E F8 00 C9 20 D0 09 20
M: 0BD9 3E F8 00 C9 20 D0 0E 18
M: 0BE1 60 20 AF FA 00 0A 0A 0A
M: 0BE9 0A 85 2A 20 3E F8 00 20
M: 0BF1 AF FA 00 05 2A 38 60 C9
```

```
M: 0BF9 3A 90 02 69 08 29 0F 60
M: 0C01 A2 02 2C A2 00 00 B4 C1
M: 0C09 D0 08 B4 C2 D0 02 E6 26
M: 0C11 D6 C2 D6 C1 60 20 3E F8
M: 0C19 00 C9 20 F0 F9 60 A9 00
M: 0C21 00 8D 00 00 01 20 CC FA
M: 0C29 00 20 8F FA 00 20 7C FA
M: 0C31 00 90 09 60 20 3E F8 00
M: 0C39 20 79 FA 00 B0 DE AE 3F
M: 0C41 02 9A A9 90 20 D2 FF A9
M: 0C49 3F 20 D2 FF 4C 47 F8 00
M: 0C51 20 54 FD 00 CA D0 FA 60
M: 0C59 E6 C3 D0 02 E6 C4 60 A2
M: 0C61 02 B5 C0 48 B5 27 95 C0
M: 0C69 68 95 27 CA D0 F3 60 A5
M: 0C71 C3 A4 C4 38 E9 02 B0 0E
M: 0C79 88 90 0B A5 28 A4 29 4C
M: 0C81 33 FB 00 A5 C3 A4 C4 38
M: 0C89 E5 C1 85 1E 98 E5 C2 A8
M: 0C91 05 1E 60 20 D4 FA 00 20
M: 0C99 69 FA 00 20 E5 FA 00 20
M: 0CA1 0C FB 00 20 E5 FA 00 20
M: 0CA9 2F FB 00 20 69 FA 00 90
M: 0CB1 15 A6 26 D0 64 20 28 FB
M: 0CB9 00 90 5F A1 C1 81 C3 20
M: 0CC1 05 FB 00 20 33 F8 00 D0
M: 0CC9 EB 20 28 FB 00 18 A5 1E
M: 0CD1 65 C3 85 C3 98 65 C4 85
M: 0CD9 C4 20 0C FB 00 A6 26 D0
M: 0CE1 3D A1 C1 81 C3 20 28 FB
M: 0CE9 00 B0 34 20 B8 FA 00 20
M: 0CF1 BB FA 00 4C 7D FB 00 20
M: 0CF9 D4 FA 00 20 69 FA 00 20
M: 0D01 E5 FA 00 20 69 FA 00 20
M: 0D09 3E F8 00 20 88 FA 00 90
M: 0D11 14 85 1D A6 26 D0 11 20
M: 0D19 2F FB 00 90 0C A5 1D 81
M: 0D21 C1 20 33 F8 00 D0 EE 4C
M: 0D29 ED FA 00 4C 47 F8 00 20
M: 0D31 D4 FA 00 20 69 FA 00 20
M: 0D39 E5 FA 00 20 69 FA 00 20
M: 0D41 3E F8 00 A2 00 00 20 3E
M: 0D49 F8 00 C9 27 D0 14 20 3E
M: 0D51 F8 00 9D 10 02 E8 20 CF
M: 0D59 FF C9 0D F0 22 E0 20 D0
M: 0D61 F1 F0 1C 8E 00 00 01 20
M: 0D69 8F FA 00 90 C6 9D 10 02
M: 0D71 E8 20 CF FF C9 0D F0 09
M: 0D79 20 88 FA 00 90 B6 E0 20
M: 0D81 D0 EC 86 1C A9 90 20 D2
M: 0D89 FF 20 57 FD 00 A2 00 00
M: 0D91 A0 00 00 B1 C1 DD 10 02
M: 0D99 D0 0C C8 E8 E4 1C D0 F3
M: 0DA1 20 41 FA 00 20 54 FD 00
M: 0DA9 20 33 F8 00 A6 26 D0 8D
M: 0DB1 20 2F FB 00 B0 DD 4C 47
M: 0DB9 F8 00 20 D4 FA 00 85 20
M: 0DC1 A5 C2 85 21 A2 00 00 86
M: 0DC9 28 A9 93 20 D2 FF A9 90
M: 0DD1 20 D2 FF A9 16 85 1D 20
M: 0DD9 6A FC 00 20 CA FC 00 85
M: 0DE1 C1 84 C2 C6 1D D0 F2 A9
M: 0DE9 91 20 D2 FF 4C 47 F8 00
M: 0DF1 A0 2C 20 C2 F8 00 20 54
M: 0DF9 FD 00 20 41 FA 00 20 54
M: 0E01 FD 00 A2 00 00 A1 C1 20
M: 0E09 D9 FC 00 48 20 1F FD 00
M: 0E11 68 20 35 FD 00 A2 06 E0
M: 0E19 03 D0 12 A4 1F F0 0E A5
M: 0E21 2A C9 E8 B1 C1 B0 1C 20
M: 0E29 C2 FC 00 88 D0 F2 06 2A
M: 0E31 90 0E BD 2A FF 00 20 A5
M: 0E39 FD 00 BD 30 FF 00 F0 03
```

```
M: 0E41 20 A5 FD 00 CA D0 D5 60
M: 0E49 20 CD FC 00 AA E8 D0 01
M: 0E51 C8 98 20 C2 FC 00 8A 86
M: 0E59 1C 20 48 FA 00 A6 1C 60
M: 0E61 A5 1F 38 A4 C2 AA 10 01
M: 0E69 88 65 C1 90 01 C8 60 A8
M: 0E71 4A 90 0B 4A B0 17 C9 22
M: 0E79 F0 13 29 07 09 80 4A AA
M: 0E81 BD D9 FE 00 B0 04 4A 4A
M: 0E89 4A 4A 29 0F D0 04 A0 80
M: 0E91 A9 00 00 AA BD 1D FF 00
M: 0E99 85 2A 29 03 85 1F 98 29
M: 0EA1 8F AA 98 A0 03 E0 8A F0
M: 0EA9 0B 4A 90 08 4A 4A 09 20
M: 0EB1 88 D0 FA C8 88 D0 F2 60
M: 0EB9 B1 C1 20 C2 FC 00 A2 01
M: 0EC1 20 FE FA 00 C4 1F C8 90
M: 0EC9 F1 A2 03 C0 04 90 F2 60
M: 0ED1 A8 B9 37 FF 00 85 28 B9
M: 0ED9 77 FF 00 85 29 A9 00 00
M: 0EE1 A0 05 06 29 26 28 2A 88
M: 0EE9 D0 F8 69 3F 20 D2 FF CA
M: 0EF1 D0 EC A9 20 2C A9 0D 4C
M: 0EF9 D2 FF 20 D4 FA 00 20 69
M: 0F01 FA 00 20 E5 FA 00 20 69
M: 0F09 FA 00 A2 00 00 86 28 A9
M: 0F11 90 20 D2 FF 20 57 FD 00
M: 0F19 20 72 FC 00 20 CA FC 00
M: 0F21 85 C1 84 C2 20 E1 FF F0
M: 0F29 05 20 2F FB 00 B0 E9 4C
M: 0F31 47 F8 00 20 D4 FA 00 A9
M: 0F39 03 85 1D 20 3E F8 00 20
M: 0F41 A1 F8 00 D0 F8 A5 20 85
M: 0F49 C1 A5 21 85 C2 4C 46 FC
M: 0F51 00 C5 28 F0 03 20 D2 FF
M: 0F59 60 20 D4 FA 00 20 69 FA
M: 0F61 00 8E 11 02 A2 03 20 CC
M: 0F69 FA 00 48 CA D0 F9 A2 03
M: 0F71 68 38 E9 3F A0 05 4A 6E
M: 0F79 11 02 6E 10 02 88 D0 F6
M: 0F81 CA D0 ED A2 02 20 CF FF
M: 0F89 C9 0D F0 1E C9 20 F0 F5
M: 0F91 20 D0 FE 00 B0 0F 20 9C
M: 0F99 FA 00 A4 C1 84 C2 85 C1
M: 0FA1 A9 30 9D 10 02 E8 9D 10
M: 0FA9 02 E8 D0 DB 86 28 A2 00
M: 0FB1 00 86 26 F0 04 E6 26 F0
M: 0FB9 75 A2 00 00 86 1D A5 26
M: 0FC1 20 D9 FC 00 A6 2A 86 29
M: 0FC9 AA BC 37 FF 00 BD 77 FF
M: 0FD1 00 20 B9 FE 00 D0 E3 A2
M: 0FD9 06 E0 03 D0 19 A4 1F F0
M: 0FE1 15 A5 2A C9 E8 A9 30 B0
M: 0FE9 21 20 BF FE 00 D0 CC 20
M: 0FF1 C1 FE 00 D0 C7 88 D0 EB
M: 0FF9 06 2A 90 0B BC 30 FF 00
M: 1001 BD 2A FF 00 20 B9 FE 00
M: 1009 D0 B5 CA D0 D1 F0 0A 20
M: 1011 B8 FE 00 D0 AB 20 B8 FE
M: 1019 00 D0 A6 A5 28 C5 1D D0
M: 1021 A0 20 69 FA 00 A4 1F F0
M: 1029 28 A5 29 C9 9D D0 1A 20
M: 1031 1C FB 00 90 0A 98 D0 04
M: 1039 A5 1E 10 0A 4C ED FA 00
M: 1041 C8 D0 FA A5 1E 10 F6 A4
M: 1049 1F D0 03 B9 C2 00 00 91
M: 1051 C1 88 D0 F8 A5 26 91 C1
M: 1059 20 CA FC 00 85 C1 84 C2
M: 1061 A9 90 20 D2 FF A0 41 20
M: 1069 C2 F8 00 20 54 FD 00 20
M: 1071 41 FA 00 20 54 FD 00 A9
M: 1079 05 20 D2 FF 4C B0 FD 00
M: 1081 A8 20 BF FE 00 D0 11 98
```

```
M: 1089 F0 0E 86 1C A6 1D DD 10
M: 1091 02 08 E8 86 1D A6 1C 28
M: 1099 60 C9 30 90 03 C9 47 60
M: 10A1 38 60 40 02 45 03 D0 08
M: 10A9 40 09 30 22 45 33 D0 08
M: 10B1 40 09 40 02 45 33 D0 08
M: 10B9 40 09 40 02 45 B3 D0 08
M: 10C1 40 09 00 00 22 44 33 D0
M: 10C9 8C 44 00 00 11 22 44 33
M: 10D1 D0 8C 44 9A 10 22 44 33
M: 10D9 D0 08 40 09 10 22 44 33
M: 10E1 D0 08 40 09 62 13 78 A9
M: 10E9 00 00 21 81 82 00 00 00
M: 10F1 00 59 4D 91 92 86 4A 85
M: 10F9 9D 2C 29 2C 23 28 24 59
M: 1101 00 00 58 24 24 00 00 1C
M: 1109 8A 1C 23 5D 8B 1B A1 9D
M: 1111 8A 1D 23 9D 8B 1D A1 00
M: 1119 00 29 19 AE 69 A8 19 23
M: 1121 24 53 1B 23 24 53 19 A1
M: 1129 00 00 1A 5B 5B A5 69 24
M: 1131 24 AE AE A8 AD 29 00 00
M: 1139 7C 00 00 15 9C 6D 9C A5
M: 1141 69 29 53 84 13 34 11 A5
M: 1149 69 23 A0 D8 62 5A 48 26
M: 1151 62 94 88 54 44 C8 54 68
M: 1159 44 E8 94 00 00 B4 08 84
M: 1161 74 B4 28 6E 74 F4 CC 4A
M: 1169 72 F2 A4 8A 00 00 AA A2
M: 1171 A2 74 74 74 72 44 68 B2
M: 1179 32 B2 00 00 22 00 00 1A
M: 1181 1A 26 26 72 72 88 C8 C4
M: 1189 CA 26 48 44 44 A2 C8 3A
M: 1191 3B 52 4D 47 58 4C 53 54
M: 1199 46 48 44 50 2C 41 42 F9
M: 11A1 00 35 F9 00 CC F8 00 F7
M: 11A9 F8 00 56 F9 00 89 F9 00
M: 11B1 F4 F9 00 0C FA 00 3E FB
M: 11B9 00 92 FB 00 C0 FB 00 38
M: 11C1 FC 00 5B FD 00 8A FD 00
M: 11C9 AC FD 00 46 F8 00 FF F7
M: 11D1 00 ED F7 00 0D 20 20 20
M: 11D9 50 43 20 20 53 52 20 41
M: 11E1 43 20 58 52 20 59 52 20
M: 11E9 53 50
```

PETITMON

PETITMON est une version BASIC du moniteur MON limitée à la commande 'M' qui permet de charger la mémoire en hexadécimal. Son seul usage prévu est l'entrée en mémoire du véritable 'MON' ci-avant. Il est obligatoire d'utiliser 'INIMEM' avant de charger PETITMON en mémoire pour libérer le bas de la zone BASIC qui sera occupée par le moniteur MON.

Au démarrage du programmme, la lettre M apparaît. Si on introduit au clavier une adresse hexadécimale sur 4 caractères (par exemple : 0801, début de MON) les 8 octets situés en mémoire à partir de cette adresse sont affichés. Chaque <RETURN> affiche les 8 suivants. Si avant de frapper <RETURN>, on introduit des chiffres hexa, ils remplacent les 8 octets affichés. Après chaque octet entré en deux caractères, on peut reculer avec la touche DEL.

Pour changer l'adresse de travail, pousser sur la touche STOP et taper 'RUN'.

En arrêtant le programme par RUN/STOP et en tapant 'GOTO 2000', on entre dans un programme de vérification qui calcule les sommes de contrôle du moniteur MON par bloc de 256 octets de 2049 à 4588, que vous pouvez comparer aux valeurs correctes.(Voir MON).

En arrêtant le programme par RUN/STOP et en tapant 'GOTO 3000', on peut sauver 'MON' sur cassette. Pour utiliser une disquette, remplacer "1,1" par "8,1" en ligne 3110.

Si vous réutilisez 'PETITMON' pour continuer à écrire un 'MON' déjà commencé, procédez comme suit :

1. allumer le 64
2. LOAD"INIMEM" et RUN
3. LOAD"MON",1,1 et NEW
4. LOAD"PETITMON" et RUN

```
3 REM
4 REM ´PETITMON´  PAR P.BRUNET -1984-
5 REM
9 IFPEEK(44)<>64THENPRINT"EMPLOYEZ INIMEM S.V.P.":NEW
10 GOTO 1000
100 POKE 198,0
110 GET A$: IF A$="" THEN 110
120 A=ASC(A$): N=A-48:RETURN
200 DA=PEEK(AD): D=INT(DA/16): R=DA-D*16
220 OC$=CHR$(D(D)+48) + CHR$(D(R)+48):RETURN
290 GOSUB 100
300 IF N<0 OR N>22 THEN 290
310 V=H(N): IF V=-1 THEN 290
320 PRINT A$;:RETURN
400 OC$="":FOR I=0 TO 3
420 V=INT(AL/16^(3-I))-INT(AL/16^(4-I))*16
430 OC$=OC$+CHR$(D(V)+48):NEXTI:PRINTOC$;:RETURN
999 REM ** 1000 ** AFFICHAGE/SAISIE                    ****
1000 DATA 0,1,2,3,4,5,6,7,8,9,17,18,19,20,21,22
1010 DATA 0,1,2,3,4,5,6,7,8,9,-1,-1,-1,-1,-1,-1,-1,-1,10,11,12,13,14,15
1020 DIM D(15),H(22),V(3)
1030 FOR I=0 TO 15: READ D(I): NEXT I
1040 FOR I=0 TO 22: READ H(I): NEXT I
1050 PRINT"[CLR][C.BAS][BLANC].M ";:FOR I=0TO3:GOSUB100:GOSUB300
1060 DD=DD+V*16^(3-I):NEXT I:AL=DD
1110 FF=AL+7
1120 FOR AD=AL TO FF: GOSUB 200: PRINT" "OC$;: NEXT AD
1130 PRINT"[C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE]
     [C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GA
     UCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE]
     [C.GAUCHE][C.GAUCHE][C.GAUCHE][C.GAUCHE]";
1140 FOR AD=AL TO FF:FOR I=0TO2
1150 IF I=2THEN PRINT" ";:GOTO1190
1155 GOSUB100
1160 IF A=20 THEN IF I=0THEN IF AD>AL THEN AD=AD-2:PRINT"[C.GAUCHE][C.G
     AUCHE][C.GAUCHE]    [C.GAUCHE][C.GAUCHE][C.GAUCHE]";:GOTO1210
1170 IF A=13THEN AD=FF:GOTO1210
1180 GOSUB 300: V(I)=V
1190 NEXT I:DA=V(0)*16+V(1):POKE AD,DA
1210 NEXT AD:AL=AL+8
1220 PRINT:PRINT".: ";:GOSUB400:GOTO1110
1998 REM ** 2000 ** SOMME DE VERIF. DU MONITEUR ****
2000 J=-1:N=0:AD=2049:REM DEBUT DU MONITEUR MON
2010 FOR K=AD TO4586 :REM FIN DU MONITEUR MON
```

```
2020 J=J+1: N=N + PEEK(K): IFJ<255GOTO2060
2050 GOSUB2100:AD=AD+256:J=-1:N=0
2060 NEXT K: IF J<>255THEN GOSUB2100
2070 END
2100 PRINT"SOMME BLOC"AD"[C.GAUCHE]($";
2110 AL=AD:GOSUB400:PRINT") ";
2140 PRINT"="N:RETURN
2998 REM ** 3000 ** SAUVER ´MON´ SUR CASSETTE   ****
3000 DD=2049:FF=4588:V(1)=INT(DD/256)
3010 V(0)=DD-V(1)*256:V(3)=INT(FF/256)
3030 V(2)=FF-V(3)*256
3040 PRINT"[CLR]POUR SAUVER ´MON´"
3050 PRINT"DEPUIS "DD"($";:AL=DD:GOSUB400:PRINT")";
3060 PRINT" JUSQUE "FF"($";:AL=FF:GOSUB400:PRINT")"
3070 PRINT:PRINT"VOUS DEVEZ :":PRINT
3080 FOR I=0TO3:PO$=STR$(43+I):VI$=STR$(V(I))
3090 PRINT"POKE"RIGHT$(PO$,LEN(PO$)-1)","RIGHT$(VI$,LEN(VI$)-1)":";:NEX
     T I
3110 PRINT"SAVE"CHR$(34)"MON"CHR$(34)",1,1"
3120 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS][C.BAS]POKE43,1:POKE44,64:POKE45
     ,0:POKE46,75:RUN"
3130 PRINT"[C.BAS]POUR OBTENIR DE NOUVEAU LE PETITMON,"
3140 PRINT"APPUYEZ SUR <RETURN>.":END
55555 SAVE"@0:PETITMON",8

READY.
```

POIGNEES.

Programme BASIC chargeant en mémoire une routine en langage machine indispensable pour effectuer une lecture correcte des entrées analogiques POT X et POT Y des 2 ports de contrôle. Cette routine se charge dans les octets inoccupés en page 2 et est longue de 63 octets. Cette routine est employée dans "SVP"

```
 10 REM LECTURE DES POIGNEES DE JEU
 11 :
 15 FORI=679TO742:READA:POKEI,A:CS=CS+A:NEXT
 16 IFCS<>8274THEN STOP
 20 SYS679:PRINT"[CLR]POT X PORT1  ";PEEK(744)
 25 PRINT"POT Y PORT1  ";PEEK(746)
 30 PRINT"BOUTON PORT1 ";255-PEEK(749)
 35 PRINT"[C.BAS][C.BAS]POT X PORT2  ";PEEK(745)
 40 PRINT"POT Y PORT2  ";PEEK(747)
 45 PRINT"BOUTON PORT2 ";255-PEEK(748)
 50 FORI=0TO200:NEXT:GOTO20
 55 :
679 DATA162,001,120,173,002,220,141,231
687 DATA002,169,192,141,002,220,169,128
695 DATA141,000,220,160,128,234,136,016
703 DATA252,173,025,212,157,232,002,173
711 DATA026,212,157,234,002,173,000,220
719 DATA009,128,141,236,002,169,064,202
727 DATA016,222,173,231,002,141,002,220
735 DATA173,001,220,141,237,002,088,096
736 REM CHECKSUM= 8274 DE 679 A 742
55555 SAVE"@0:POIGNEES",8

READY.
```

PRESS RETURN 1 ET PRESS RETURN 2.

Deux petites routines en langage machine qui attendent la frappe de la touche RETURN avec un changement des couleurs de fond et de bord très rapide (plus de 10.000 fois par seconde) et moins rapide (environ 1000 fois par seconde).

Dans 'PRESS RETURN 1' on peut remplacer le code ASCII 013 du RETURN par une autre valeur en 51223 (fin de la ligne 51216). Par exemple, employer 48 pour tester la touche "0". Modifier en conséquence la valeur CS en ligne 20.

La même manoeuvre est possible dans 'PRESS RETURN 2' en 51228 (au lieu de 51223).

```
1 REM ATTEND FRAPPE DE RETURN
2 REM AVEC EFFETS SPECIAUX ECRANS
3 :
10 CS=0:FORI=51200TO51238:READ A:POKEI,A:CS=CS+A:NEXT
20 IFCS<>6268THENSTOP
30 PRINT"[CLR][C.BAS][C.BAS]TAPEZ RETURN "
40 SYS 51200
50 END
80 REM CETTE ROUTINE MARCHE N'IMPORTE OU EN MEMOIRE
90 REM FAIRE : SYS(DEBUT ROUTINE)
51200 DATA173,032,208,141,254,199,173,033
51208 DATA208,141,255,199,200,140,032,208
51216 DATA140,033,208,032,228,255,201,013
51224 DATA208,242,173,254,199,141,032,208
51232 DATA173,255,199,141,033,208,096
51233 REM CHECKSUM= 6268 DE 51200 A 51238
55555 SAVE"@0:PRESS RETURN1",8

READY.
```

```
1 REM ATTEND FRAPPE DE RETURN
2 REM AVEC AUTRES EFFETS SPECIAUX ECRANS
3 :
10 CS=0:FORI=51200TO51249:READ A:POKEI,A:CS=CS+A:NEXT
20 IFCS<>7658THENSTOP
30 PRINT"[CLR][C.BAS][C.BAS]TAPEZ RETURN "
40 SYS 51200
50 LIST
60 REM L'ADRESSE 51245 CONTIENT LA VALEUR DELAI
70 REM QUI DONNE LA LARGEUR DES BANDES DE COULEUR
80 REM CETTE ROUTINE MARCHE N'IMPORTE OU EN MEMOIRE
90 REM FAIRE : SYS(DEBUT ROUTINE)
51200 DATA173,032,208,141,254,199,173,033
51208 DATA208,141,255,199,172,018,208,140
51216 DATA032,208,140,033,208,032,044,200
51224 DATA032,228,255,201,013,208,237,173
51232 DATA254,199,141,032,208,173,255,199
51240 DATA141,033,208,096,162,000,202,208
51248 DATA253,096
51249 REM CHECKSUM= 7658 DE 51200 A 51249
55555 SAVE"@0:PRESS RETURN2",8

READY.
```

PROTECTEUR.

Programme auto-explicatif qui rend un programme BASIC stocké sur disque inlistable, incopiable et qui démarre seul lorsqu'on le charge par LOAD"PROG",8,1. Le 'LOAD' normal ne donne rien, bien sûr ! Le nouveau nom du programme peut être modifié à volonté. Ne convient pas pour les cassettes. Pour être protégé efficacement, un programme doit empêcher LIST par le truc suivant:

1. PRINT"O REM"CHR$(219) suivi de <RETURN>
2. mettre le curseur sur la ligne imprimée par 1. et taper <RETURN>.

et inhiber STOP/RESTORE par POKE 808,234
ou STOP par POKE 808,239
ou SAVE par POKE 808,225:POKE 818,32

3. Une protection supplémentaire contre 'LIST': POKE 775,167

```
10 PRINT"[CLR][C.BAS]   CE PROGRAMME REND UN  PROGRAMME
20 PRINT"    BASIC INLISTABLE. EXECUTER D´ABORD
30 PRINT"    ´PROTECTEUR´ , PUIS   CHARGER  LE
40 PRINT"    PROGRAMME  A  PROTEGER  EN MEMOIRE
50 PRINT"    Y AJOUTER LA LIGNE 999 CI-DESSOUS.
60 PRINT"    ET  TAPER :  ´RUN 999´.LE PROGRAMME
70 PRINT"    DEVIENT INLISTABLE ET ´INSAUVABLE´
90 PRINT"    LE  PROGRAMME  PROTEGE  NE PEUT SE
91 PRINT"    CHARGER QUE PAR LOAD ´NOM´,8,1  ET
92 PRINT"    IL  DEMARRE  DES QU´ IL EST CHARGE
93 PRINT"    SANS  DEVOIR  FAIRE  ´RUN´.
100 PRINT:PRINT:PRINT
110 FORI=828TO897:READA:POKEI,A:CS=CS+A:NEXT
120 IFCS<>6627THENSTOP
828 DATA169,175,141,007,003,141,051,003
836 DATA169,008,141,006,003,141,050,003
844 DATA169,254,141,023,003,141,025,003
852 DATA169,105,141,022,003,141,024,003
860 DATA169,082,141,119,002,169,213,141
868 DATA120,002,169,013,141,121,002,169
876 DATA003,133,198,169,202,141,038,003
884 DATA169,241,141,039,003,169,001,133
892 DATA043,169,008,133,044,096
893 REM CHECKSUM= 6627 DE 828 A 897
998 LIST 999
999 PRINT"[CLR]";:POKE43,38:POKE44,3:POKE806,60:POKE807,3:SAVE"@0:PROG.
    PROTECT",8,1
1000 END
55554 END
55555 SAVE"@0:PROTECTEUR",8

READY.
```

RECOPIE.

Ce programme de recopie d'écran en mode 1 n'accepte ni les minuscules, ni les inverses mais voyez sa longueur!

```
10 REM RECOPIE D'ECRAN NORMAL
20 :
30 OPEN3,3:OPEN4,4:PRINT"[HOME]";:FOR I=0TO999
40 GET#3,A$:PRINT#4,A$;:NEXT I:CLOSE 3:CLOSE 4:END
50 :
55555 SAVE"@0:RECOPIE",8

READY.
```

REORG.

Ce programme trie le catalogue d'une disquette insérée dans le lecteur 1541 par ordre alphabétiqque.

N'essayez ce programme que sur une copie de test avant de vous y fier. Une faute de frappe pourrait être fatale.

la méthode de tri utilisée est le tri à bulle (BUBBLE SORT).

```
20 REM PAR THIERRY SALOME
21 :
98 IFPEEK(2)=0THENPOKE2,8
99 DIMT$(224),TT$(28),TS$(28):Q$=CHR$(34)
110 PRINT"[CLR]TRIE LE CATALOGUE DU DISQUE"
130 NP=8:DR=0
300 PRINT"[C.BAS]ENTREZ LA DISQUETTE"
320 POKE198,0:PRINT"ET POUSSER ENTER"
330 WAIT198,1:GETA$:CH=4:T=18:S=1:CP=0
1000 OPEN15,8,15,"IO":OPEN4,8,4,"#"
1015 PRINT"[CLR]CATALOGUE"
1020 PRINT#15,"U1"4;0;T;S:PRINT#15,"B-P"4;0:GET#4,C1$,C2$
1050 NT=ASC(C1$+CHR$(0)):NS=ASC(C2$+CHR$(0))
1055 T=NT:S=NS:TT$(CP/8)=CHR$(NT):TS$(CP/8)=CHR$(NS)
1060 FORI=1TO8:Z$="":FORJ=1TO30
1080 GET#4,C$
1090 Z$=Z$+CHR$(ASC(C$+CHR$(0))):NEXTJ
1100 IFASC(MID$(Z$,1,1))<>0THENT$(I+CP)=Z$:PRINT"[C.DROITE]"Q$MID$(Z$,4
     ,16)Q$:GOTO1120
1110 T$(I+CP)=""
1120 GET#4,C0$,C1$:NEXTI
1130 IFNT<>0THENCP=CP+8:GOTO1020
1131 MX=CP+I-1:IFMX=0THENPRINT"[CLR]REPERTOIRE VIDE.":GOTO1290
1132 J=1:FORI=1TOMX:IFT$(I)<>""THENT$(J)=T$(I):J=J+1
1134 NEXTI:FORI=JTOMX:T$(I)="":NEXTI
1135 MX=J-1:CP=INT(MX/8):IFMX/8=INT(MX/8)THENCP=CP-1
1136 TT$(CP)=CHR$(0):TS$(CP)=CHR$(255)
1145 PRINT"[CLR]TRI"
1150 FORI=1TOMX-1
1160 FORJ=I+1TOMX:PRINT"[HOME][C.BAS][C.BAS][C.BAS][C.BAS]            [HO
     ME][C.BAS][C.BAS][C.BAS][C.BAS]"I;J
1170 IFMID$(T$(J),4,16)<MID$(T$(I),4,16)THENZ$=T$(J):T$(J)=T$(I):T$(I)=
     Z$
1180 NEXTJ,I:T=18:S=1:PRINT"[CLR]ECRITURE"
1200 FORI=0TOCP:Z0$="":FORJ=1TO4
1215 IFT$(I*8+J)=""THENFORK=1TO30:Z0$=Z0$+CHR$(0):NEXTK:GOTO1221
```

```
1220 ZO$=ZO$+T$(I*8+J)
1221 ZO$=ZO$+CHR$(0):NEXTJ
1230 Z1$="":FORJ=5TO8
1231 IFT$(I*8+J)=""THENFORK=1TO30:Z1$=Z1$+CHR$(0):NEXTK:GOTO1236
1235 Z1$=Z1$+T$(I*8+J)
1236 IFJ<8THENZ1$=Z1$+CHR$(0)+CHR$(0)
1237 NEXTJ
1240 PRINT#15,"B-P"4,0:PRINT#4,TT$(I);TS$(I);ZO$;Z1$;:PRINT#15,"U2:"4;0
     ;T;S
1250 T=ASC(TT$(I)+CHR$(0)):S=ASC(TS$(I)+CHR$(0)):NEXTI
1290 CLOSE4:CLOSE15:END
55555 SAVE"@0:REORG",8

READY.
```

REVEIL-MATIN.

Ce programme charge et démarre une routine d'interruption qui affiche l'heure exacte, celle de l'horloge temps-réel du 6526 et non l'horloge TI$ peu fiable.

De plus, la fonction réveil est programmable sur 24 heures. La "sonnerie" audio-visuelle dure une minute et ne gênee nullement les programmes BASIC qui tournent à ce moment, même pendant un INPUT.

- Pour arrêter la sonnerie, pousser sur "F1".
- La couleur de l'affichage est celle du curseur, à tout moment.
- En cas de perte d'affichage par STOP-RESTORE, relancer l'affichage par SYS 839. (RESET, lui, déprogramme l'horloge).
- Avant un accès cassette, arrêter le réveil par SYS 995 et le relancer ensuite en rechargeant le programme et en n'exécutant que le SYS 839. L'horloge n'aura pas perdu une seconde.

```
100 REM REVEIL-MATIN POUR CBM-64
110 REM
120 REM
130 REM SYS839 :REM RANIME L'AFFICHAGE
140 REM SYS995 :REM STOPPE L'AFFICHAGE
150 REM POKE982,CL:REM COULEUR HORLOGE
160 REM POUSSER F1 ARRETE LE REVEIL
170 REM
180 REM
190 GOSUB2000:REM HORLOGE
200 GOSUB3000:REM REVEIL-MATIN
210 END
220 :
679 DATA173,013,220,041,004,240,003,141
687 DATA235,002,173,235,002,240,050,173
695 DATA162,000,106,106,106,041,012,141
703 DATA032,208,041,004,141,024,212,240
711 DATA017,162,006,189,235,002,157,032
719 DATA004,173,134,002,157,032,216,202
727 DATA208,241,173,197,000,201,004,208
735 DATA008,142,235,002,142,024,212,134
743 DATA198,076,049,234,000,018,005,022
751 DATA005,009,012
752 REM CHECKSUM= 7842 DE 679 A 753
753 :
```

```
839 DATA120,162,089,142,020,003,162,003
847 DATA142,021,003,169,129,141,014,220
855 DATA088,096,173,011,220,170,041,015
863 DATA024,105,048,141,067,004,138,016
871 DATA004,162,016,016,002,162,001,142
879 DATA077,004,162,032,041,016,240,002
887 DATA162,049,142,066,004,173,010,220
895 DATA170,041,015,105,048,141,070,004
903 DATA138,074,074,074,074,024,105,048
911 DATA141,069,004,173,009,220,170,041
919 DATA015,105,048,141,073,004,138,074
927 DATA074,074,074,024,105,048,141,072
935 DATA004,173,008,220,105,048,141,075
943 DATA004,169,032,141,065,004,141,076
951 DATA004,141,079,004,162,014,157,024
959 DATA004,202,208,250,169,058,141,068
967 DATA004,141,071,004,169,046,141,074
975 DATA004,169,013,141,078,004,173,134
983 DATA002,162,013,157,065,216,202,208
991 DATA250,076,167,002,120,169,049,141
999 DATA020,003,169,234,141,021,003,088
1007 DATA096
1008 REM CHECKSUM= 15275 DE 839 A 1007
1999 :
2000 REM INIT. ET REGLAGE HORLOGE
2010 CS=0:FOR I=679TO753
2020 READ A:POKEI,A:CS=CS+A:NEXT
2030 IF CS<>8000 THEN STOP
2040 CS=0:FOR I=839TO1007
2050 READ A:POKE I,A:CS=CS+A:NEXT
2060 IF CS<>15275THEN STOP
2070 INPUT"[CLR][C.BAS][C.BAS][C.BAS]MATIN (A) OU APRES-MIDI (P)";A$
2080 PRINT:INPUT"QUELLE HEURE EST-IL (0-11)";H
2090 PRINT:INPUT"QUELLE MINUTE (0-59)";M
2100 IFH>9THENH=H+6
2110 IFLEFT$(A$,1)="P"THENH=H+128
2120 C=56328:POKEC+3,H:POKEC+1,0
2130 M=M+INT(M/10)*6
2140 POKEC+2,M:POKEC,0:SYS839
2998 RETURN
2999 :
3000 REM REGLAGE DU REVEIL-MATIN
3010 PRINT:PRINT"HREGLAGE DU REVEIL-MATIN"
3020 PRINT:INPUT"MATIN (A) OU APRES-MIDI (P)";A$
3030 PRINT:INPUT"HEURE (0-11)";H
3040 A$=LEFT$(A$,1):H=H-6*(H>9)-128*(A$="P")
3050 PRINT:INPUT"MINUTE (0-59)";M
3060 M=M+INT(M/10)*6
3070 POKEC+7,136:POKEC+3,H:POKEC+2,M:POKEC,1:POKEC+7,8
3080 POKE54273,99:POKE54278,240:POKE54276,21
3090 POKE54287,2:POKE54290,17
3998 RETURN
3999 :
55555 SAVE"@0:REVEIL-MATIN",8

READY.
```

SAUVEUR. (par A.SURNY)

Ce programme en langage machine est long de quelques six cents octets. Il a pour utilité de transformer un programme en langage machine qui se charge normalement à une adresse différente de BASIC (même derrière les ROM BASIC et KERNAL) en un programme qui se charge normalement.

Exemple : le programme PRINTER se charge en mémoire de $CF5A à $CFF9 et est démarré par SYS 53082.

Sans SAUVEUR :

```
LOAD"PRINTER",8,1
NEW
SYS 53082
```

Après avoir utilisé SAUVEUR :

```
LOAD"PRINTER",8
RUN
```

Mode d'emploi :

1. LOAD"SAUVEUR" et RUN.
2. A la question "DISK(Y/N)", répondre Y si on utilise la disquette
   N si on utilise la cassette.
3. LOAD"PRINTER",8,1 charge le programme à modifier
4. NEW remet à 0 les pointeurs du BASIC.
5. PRINT USR(53082) communique à 'SAUVEUR' l'adresse de départ du programme à modifier.
6. (SAUVEUR redescend le programme en mémoire basse et y ajoute le SYS de démarrage qui le retransférera à sa place et le lancera).
7. SAVE"PRINTERMOD",8 sauve le programme modifié sur disque.

- Pour la cassette, remplacer 8 par 1 au point 7.
- Attention, SAUVEUR occupe en mémoire les adresses $CE00 à $CFD1. Il ne convient donc pas pour les programmes occupant cette zone.
- On peut charger SAUVEUR avec 'MON' ou éventuellement avec 'INIMEM'+'PETIT MON'.
- Pour sauver 'SAUVEUR' avec MON, taper:
  .S"SAUVEUR",08,0801,0A22

```
M: 0801 0C 08 0A 00 9E 20 32 30
M: 0809 36 34 00 00 00 00 00 A5
M: 0811 2D 85 FB A5 2E 85 FC A9
M: 0819 D0 85 FE A0 00 84 FD B1
M: 0821 FB 91 FD A5 FE C9 CE D0
M: 0829 06 A5 FD C9 00 F0 13 A5
M: 0831 FB D0 02 C6 FC A5 FD D0
M: 0839 02 C6 FE C6 FB C6 FD 38
M: 0841 B0 DD 4C 00 CE 00 00 00
M: 0849 00 00 00 00 00 00 00 A9
M: 0851 F0 8D 11 03 A9 CE 8D 12
```

```
M: 0859 03 A0 0C B9 C5 CF 20 CA
M: 0861 F1 88 D0 F7 20 CF FF C9
M: 0869 59 F0 0F C9 4E D0 EA A9
M: 0871 CA 8D 26 03 A9 F1 8D 27
M: 0879 03 60 A9 23 8D 26 03 A9
M: 0881 CF 8D 27 03 60 A5 C2 C9
M: 0889 09 B0 09 A5 C1 C9 50 B0
M: 0891 03 4C 97 CE A5 AE F0 02
M: 0899 E6 AF A5 C1 8D AC CF A5
M: 08A1 C2 8D A6 CF A5 AF 8D 98
M: 08A9 CF A9 08 85 2E A9 50 85
M: 08B1 2D A0 00 B1 C1 91 2D E6
M: 08B9 2D D0 02 E6 2E E6 C1 D0
M: 08C1 F2 E6 C2 A5 C2 C5 AF D0
M: 08C9 EA A9 CF 85 C2 A9 7F 85
M: 08D1 C1 A9 08 85 FC 84 FB B1
M: 08D9 C1 91 FB E6 FB E6 C1 A5
M: 08E1 FB C9 46 D0 F2 60 A5 AE
M: 08E9 F0 02 E6 AF A5 AF 85 2E
M: 08F1 8D 78 CF 8D 55 CF A9 3C
M: 08F9 85 2D 85 AE A9 CF 85 FC
M: 0901 A9 7E 85 FB A0 00 B1 FB
M: 0909 91 AE C6 FB C6 AE A5 AE
M: 0911 C9 14 D0 F2 A5 AF 85 FE
M: 0919 84 FD B1 FD 91 AE A5 FD
M: 0921 D0 02 C6 FE A5 AE D0 02
M: 0929 C6 AF C6 FD C6 AE A5 FE
M: 0931 C9 07 D0 E6 B1 FB 91 AE
M: 0939 C6 FB C6 AE D0 F6 60 A5
M: 0941 15 48 A5 14 48 20 F7 B7
M: 0949 A5 01 29 FE 85 01 A5 14
M: 0951 8D C3 CF 8D 7C CF A5 15
M: 0959 8D C4 CF 8D 7D CF 68 85
M: 0961 14 68 85 15 20 36 CE A0
M: 0969 00 A5 01 09 01 85 01 4C
M: 0971 A2 B3 48 8A 48 BA BD 06
M: 0979 01 C9 F2 D0 0F BD 07 01
M: 0981 C9 F4 D0 08 A5 AE 85 C1
M: 0989 A5 AF 85 C2 68 AA 68 4C
M: 0991 CA F1 00 0C 08 0A 00 9E
M: 0999 20 32 30 36 34 00 00 00
M: 09A1 00 00 4C 15 01 00 A9 08
M: 09A9 85 FC 85 FE A9 14 85 FB
M: 09B1 A0 00 84 FD B1 FB 91 FD
M: 09B9 E6 FB D0 02 E6 FC E6 FD
M: 09C1 D0 F2 E6 FE A5 FE C9 00
M: 09C9 D0 EA 4C 7B CF 00 00 0C
M: 09D1 08 0A 00 9E 20 32 30 36
M: 09D9 34 00 00 00 00 00 A5 2D
M: 09E1 85 FB A5 2E 85 FC A9 00
M: 09E9 85 FE A0 00 84 FD B1 FB
M: 09F1 91 FD A5 FE C9 00 D0 06
M: 09F9 A5 FD C9 00 F0 13 A5 FB
M: 0A01 D0 02 C6 FC A5 FD D0 02
M: 0A09 C6 FE C6 FB C6 FD 38 B0
M: 0A11 DD 4C C2 CF 20 3A 29 4E
M: 0A19 2F 59 28 20 4B 53 49 44
M: 0A21 0D 00 00 00 00 00 00 00
```

SD

Programme partiellement en langage machine et partiellement en BASIC qui affiche le répertoire complet d'un disque et vous permet de choisir un programme par son numéro d'ordre. Il sera chargé et exécuté directement.

Il faut mettre ce programme sur chacune de vos disquettes : elles deviennent tellement plus faciles à utiliser !

```
10 PRINT"[CLR][NOIR]"
100 DIMF$(224)
120 FORX=880TO950:READZ:POKEX,Z:CS=CS+Z:NEXT
130 IFCS<>10432THENPRINT"[CLR][RVS ON]ERREUR DE DATA":END
160 PRINTTAB(12)"[RVS ON]TABLE DES MATIERES[RVS OFF][C.BAS][C.BAS]"
190 OPEN15,8,15
200 PRINT#15,"I0":GOSUB290
210 OPEN1,8,0,"$0":GOSUB290
220 H$="                "+"":SYS880
230 IF ST THEN CLOSE1: CLOSE15: GOTO 340
240 F$="                "+"":SYS880
250 IF RIGHT$(F$,1)="P"THENC=C+1:F$(C)=LEFT$(F$,16)
260 GOTO 230
280 REM ERREUR DISQUE
290 INPUT#15,EN,EM$,ET,ES:IFEN=0THENRETURN
300 PRINTEN;EM$;ET;ES
310 CLOSE1:CLOSE15:END
330 REM MENU
340 IFF=0ORF=CTHENS=1:F=C:IFC>40THENF=40:GOTO360
350 IFC>FTHENS=S+20:F=C:IFC>S+20THENF=S+19
360 L=0
370 PRINT"[CLR][RVS ON]NOM DU DISQUE:[RVS OFF] ";H$"[C.BAS]"
380 FORX=STOS+19
390 L=L+1
395 IFLEN(F$(X))=0THEN420
400 IFASC(F$(X))=32THEN420
410 PRINT"[RVS ON]"RIGHT$(STR$(X),2)"[RVS OFF]";CHR$(95);" ";F$(X)
420 NEXT
430 PRINT
440 PRINT"[RVS ON] ESPACE=SUITE OU VALEUR OU * POUR FIN[RVS OFF]"
450 PRINTTAB(9)"[RVS ON] VOTRE CHOIX [RVS OFF]";:INPUTG$
500 IFG$="*"THENPRINT"[CLR]":END
510 G=VAL(G$):IFG=0THEN340
550 REM — CHAIN
560 F$=F$(G)
570 IFRIGHT$(F$,1)=" "THENF$=LEFT$(F$,LEN(F$)-1):GOTO 570
580 PRINT"[CLR]LOAD"+CHR$(34)+D$+":"+F$+CHR$(34)+",8"
590 PRINT"[C.BAS][C.BAS][C.BAS][C.BAS]?"+CHR$(34)+"[CLR]"+CHR$(34)+":RU
    N"
600 POKE631,19:POKE632,13:POKE633,13:POKE198,3
610 END
880 DATA162,1,32,198,255,160,1,177,71,133
890 DATA99,200,177,71,133,100,162,32,32,207
900 DATA255,202,224,27,208,248,32,207,255,201
910 DATA34,240,5,202,208,246,240,29,160,0
920 DATA32,207,255,201,34,240,5,145,99,200
930 DATA208,244,32,207,255,240,10,201,80,208
940 DATA247,160,16,145,99,208,241,32,204,255
950 DATA96
55555 SAVE"@0:SD",8

READY.
```

SIBOFF

Ce tout petit programme en langage machine met hors service une cartouche d'extension BASIC (SIMON'S BASIC par exemple) en rétablissant les pointeurs BASIC normaux. Cette procédure permet de sortir du 'SIMON'S BASIC' sans couper le courant. Toutefois l'effet est assez semblable au RESET. Pour récupérer un éventuel programme en mémoire, il faut utiliser une version de DE-NEW

correspondant à la bonne adresse de départ de BASIC avant l'exécution de 'SIBOFF'. Cette adresse dépend du programme en cartouche que l'on veut inhiber.

```
10 REM  INHIBE SIMON'S BASIC
20 REM
30 REM  PAR B.MICHEL
40 REM
50 REM  ON REAUTORISE SIMON'S BASIC PAR
60 REM
70 REM  SYS 33106
80 REM
100 FOR I=900 TO 907:READ A:POKE I,A:NEXT I
200 SYS 900
300 DATA 162,255,120,154,216,76,239,252
999 END
55555 SAVE"@0:SIBOFF",8

READY.
```

SILENCE.

Ce tout petit programme en langage machine coupe le son généré par le SID proprement, c'est à dire sans générer de parasites. Il peut être redéplacé à n'importe quelle adresse et occupe 14 octets.

```
10 REM SILENCE
20 :
30 REM ARRETE LE SON DU CBM-64
40 REM L'ADRESSE DU PROGRAMME PEUT ETRE
50 REM MODIFIEE SANS PROBLEME
60 :
70 SI=49152
80 CS=0:FORI=SI TO SI+13:READA:POKEI,A:CS=CS+A:NEXT
90 IFCS<>1949THENSTOP
100 :
110 SYS SI
120 END
49152 DATA120,169,000,170,157,000,212,232
49160 DATA224,025,208,248,088,096
49161 REM CHECKSUM= 1949 DE 49152 A 49165
55555 SAVE"@0:SILENCE",8

READY.
```

SUPERLIST 64.

Ce programme commence à la ligne 50000. Il est prévu pour être ajouté (voir APPENDISK) à un autre programme et il le liste comme le ferait le LIST de la ROM BASIC. Le programme hybride étant en mémoire, il suffit de lancer 'SUPERLIST 64' par GOTO 63500.

Son intérêt pratique est faible mais il permet de comprendre comment est structuré le BASIC en mémoire (voir le chapître 2).

Pour avoir le résultat à l'imprimante, répondre "O" à la question. Toutefois, sur l'imprimante COMMODORE, il liste les codes de contrôle tels que 'CLEAR' ou 'HOME' en clair et non sous forme de graphiques, ce qui rend les liste de programmes bien plus lisibles.

On préfère lire : 3*(CRS BAS) que de voir des "Q" sur fond inversé !

```
50000 END
55555 SAVE"@0:SUPERLIST64",8
55556 END
63500 REM
63501 REM*******************************
63502 REM*         SUPERLIST           *
63503 REM*                             *
63504 REM*       PAR P. BRUNET         *
63505 REM*    COPYRIGHT 1983 BCM       *
63506 REM*******************************
63507 :
63508 REM OPTION IMPRIMANTE
63509 :
63510 PRINT"[BLANC]VOULEZ-VOUS LISTER SUR IMPRIMANTE?"
63511 :
63513 REM MISE A ZERO BUFFER CLAVIER
63515 :
63520 POKE 198,0
63521 :
63523 REM TOUCHE PRESSEE ?
63525 :
63530 GET A$:IF A$="" THEN 63530
63531 :
63533 REM IMP=1 SI OPTION IMPRIMANTE
63535 :
63540 IF A$="O" THEN IMP=1:GOTO 63570
63541 :
63543 REM IMP=0 SI OPTION ECRAN
63545 :
63550 IF A$="N" THEN IMP=0:GOTO 63570
63560 GOTO 63530
63561 :
63563 REM TB:ADRESSE MOTS-CLEF BASIC
63565 REM TA$:LISTE DES CONTROLES
63567 :
63570 TB=41118:DIM TA$(62)
63571 :
63573 REM CREATION DES TABLEAUX
63575 :
63590 FOR I=1 TO 62:READ TA$(I):NEXT I
63591 :
63593 REM SI OPTION IMPRIMANTE
63594 REM OUVERTURE DU CANAL DE
63596 REM SORTIE IMPRIMANTRE
63598 :
63600 IF IMP=1 THEN OPEN4,4:CMD4
63607 :
63608 REM RETOUR DEBUT DE PAGE
63609 :
63610 PRINT"[CLR]"
63611 REM*******************************
63612 REM*     DEBUT DU PROGRAMME      *
63613 REM*******************************
63614 REM CAD:ADRESSE DEBUT BASIC OU
63615 REM NL :NUMERO DE LIGNE
63616 REM         ADRESSE DEBUT LIGNE
```

```
63617 REM SAD:ADRESSE LIGNE SUIVANTE
63618 REM NL :NUMERO DE LIGNE
63619 :
63620 CAD=PEEK(43)   +PEEK(44)*256 -1
63630 SAD=PEEK(CAD+1)+PEEK(CAD+2)*256
63640 NL =PEEK(CAD+3)+PEEK(CAD+4)*256
63641 :
63643 REM EMPECHE DE LISTER LE PGM
63645 :
63650 IF NL<63500 THEN 63680
63651 :
63653 REM FIN DU PROGRAMME ET FERMETURE
63654 REM DU CANAL IMPRIMANTE
63656 :
63660 IF IMP=1 THEN CLOSE4
63670 END
63671 :
63673 REM IMPRIME NUMERO DE LIGNE
63675 :
63680 PRINT NL;
63681 :
63682 REM*******************************
63683 REM* DEBUT DE BOUCLE PRINCIPALE *
63684 REM*******************************
63685 :
63690 FORI=CAD+5 TO SAD-2
63691 :
63693 REM SAISIE DU CARACTERE DANS LA
63694 REM LIGNE BASIC
63696 :
63700 CHAR=PEEK(I)
63701 :
63703 REM TEST DE GUILLEMETS OUVRANTS
63704 REM ASC=34  :CODE DES GUILLEMETS
63705 REM QUOTE=0 :ENTRE GUILLEMETS
63706 REM QUOTE=1 :HORS GUILLEMETS
63708 :
63710 IF CHAR=34 AND QUOTE=0 THEN QUOTE=1:GOTO 63730
63711 :
63713 REM TEST DE GUILLEMETS FERMANTS
63715 :
63720 IF CHAR=34 AND QUOTE=1 THEN QUOTE=0
63721 :
63723 REM SI HORS GUILLEMETS ET
63724 REM 127 < CODE ASCII < 204
63725 REM ALORS MOT CLEF BASIC
63727 :
63730 IF QUOTE=0 AND CHAR>127 AND CHAR<204 THEN GOTO 63800
63731 :
63733 REM SI HORS GUILLEMETS
63734 REM IMPRIME CARACTERE TEL-QUEL
63736 :
63740 IF QUOTE=0 THEN PRINTCHR$(CHAR);
63741 :
63743 REM SI ENTRE GUILLEMETS APPEL
63744 REM AU SOUS-PROGRAMME GUILLEMETS
63746 :
63750 IF QUOTE=1 THEN GOSUB 63900
63751 :
63753 REM*******************************
63754 REM* FIN DE LA BOUCLE PRINCIPALE*
63755 REM*******************************
63757 :
63760 NEXT I
63761 :
63763 REM APRES SORTIE DE LA BOUCLE
63764 REM PRINCIPALE, RETOUR A LA LIGNE
63765 REM POUR LA LIGNE SUIVANTE
63767 :
63770 PRINT
63771 :
63773 REM REMISE A JOUR DE L'ADRESSE
63774 REM DEBUT DE LIGNE
```

```
63776 :
63780 CAD=SAD-1
63781 :
63782 REM*******************************
63783 REM*      FIN DU PROGRAMME       *
63784 REM*******************************
63786 :
63790 GOTO 63630
63791 :
63793 REM*******************************
63794 REM*       SOUS-PROGRAMME        *
63795 REM*      MOTS CLEFS BASIC       *
63796 REM*******************************
63797 REM T: INDICE DE RECHERCHE
63798 REM D: NUMERO DU MOT CLEF BASIC
63799 :
63800 T=TB:D=CHAR-128:Q=0
63801 :
63803 REM SI NUMERO DU MOT CLEF EST
63804 REM NUL ALLER A L'IMPRESSION
63806 :
63810 IF D=0 THEN 63850
63811 :
63813 REM SAISIE D'UN CARACTERE DANS
63814 REM LA TABLE DES MOTS-CLEFS
63816 :
63820 P=PEEK(T)
63821 :
63823 REM SORTIE DE CETTE LIGNE SI
63824 REM FIN DE MOT CLEF
63826 :
63830 IF P<128 THEN T=T+1:GOTO 63820
63831 :
63833 REM REMISE A JOUR DES INDICES
63834 REM ET RETOUR AU MOT CLEF SUIVANT
63836 :
63840 T=T+1:D=D-1:GOTO 63810
63841 :
63843 REM SAISIE DU BON MOT CLEF. TEST
63844 REM DE FIN DE MOT CLEF. SI OUI
63845 REM ALORS Q=1
63847 :
63850 P=PEEK(T):IF P>127 THEN P=P-128 :Q=1
63851 :
63853 REM IMPRESSION DU MOT CLEF BASIC
63855 :
63860 PRINT CHR$(P);:T=T+1
63861 :
63863 REM SI FIN DE MOT CLEF, RETOUR AU
63864 REM PROGRAMME PRINCIPAL
63866 :
63870 IF Q=1 THEN 63750
63871 :
63873 REM*******************************
63874 REM*   FIN DU SOUS PROGRAMME     *
63875 REM*******************************
63877 :
63880 GOTO 63850
63881 :
63882 REM*******************************
63883 REM*       SOUS PROGRAMME        *
63884 REM*         GUILLEMETS          *
63885 REM*******************************
63887 :
63891 :
63893 REM TEST SI CARACTERE DE CONTROLE
63895 :
63900 IF(CHAR>31 AND CHAR<128)OR(CHAR>159 AND CHAR<256)THEN PRINTCHR$(C
      HAR);:RETURN
63901 :
63903 REM SP: NUMERO DU CARACTERE
63904 REM NB: NOMBRE DE CARACTERES
63905 REM                IDENTIQUES
```

```
63907 :
63910 SP=CHAR:NB=1
63911 :
63913 REM BOUCLE DE CALCUL DU NOMBRE
63914 REM D'ELEMENTS IDENTIQUES
63916 :
63920 I=I+1:IF SP=PEEK(I) THEN NB=NB+1:GOTO 63920
63921 :
63923 REM RECALCUL DES INDICES > 128
63925 :
63930 I=I-1:IF SP>128 THEN SP=SP-97
63931 :
63933 REM IMPRESSION DU NBRE D'ELEMENTS
63935 :
63940 IF NB>1 THEN PRINTSTR$(NB)"*";
63941 :
63943 REM IMPRESSION DU CONTROLE
63950 PRINT"["TA$(SP)"]";
63951 :
63953 REM*******************************
63954 REM*   FIN DU SOUS PROGRAMME     *
63955 REM*******************************
63957 :
63960 RETURN
63961 :
63963 REM*******************************
63964 REM* ELEMENTS DE LA TABLE DES    *
63965 REM* CARACTERES DE CONTROLE      *
63966 REM*******************************
63968 :
63980 DATACTRL A,CTRL B,CTRL C,CTRL D,BLANC,CTRL F,CTRL G,CTRL H
63981 DATACTRL I,CTRL J,CTRL K,CTRL L,CTRL M,CTRL N,CTRL O,CTRL P
63982 DATACRS BAS,RVS ON,HOME,DELETE,CTRL U,CTRL V,CTRL W,CTRL X
63983 DATACTRL Y,CTRL Z,CTRL [,ROUGE,CRS DROITE,VERT,BLEU,C= 1
63984 DATA,,F 1,F 3,F 5,F 7,F 2
63985 DATAF 4,F 6,F 8,,,NOIR,CRS HAUT
63986 DATARVS OFF,CLEAR,INSERT,C= 2,C= 3,C= 4,C= 5,C= 6
63987 DATAC= 7,C= 8,POURPRE,CRS GAUCHE,JAUNE,CYAN

READY.
```

S.V.P.

* Après réponse à la question "JOYSTICK ou POIGNEES", le message "UN MOMENT,S'IL VOUS-PLAIT JE TRAVAILLE" apparaît. Les 10 niveaux de gris possibles sur T.V. noir et blanc et 30 parmi les 256 couleurs possibles sont visibles dans un encadré démonstratif. Ensuite, l'écran de travail apparaît.

* Le joystick doit être connecté au port2, les poignées au port1 (ou une souris, ou une tablette graphique...).

* La main-curseur sert au tracé. Appuyer sur le bouton de tir dépose un caractère ou choisit une option.

* Les caractères viennent des 5 lignes inférieures. Décrivons-les:

* La rangée supérieure contient 4 gros pots de peinture et 16 petits. Ces derniers sélectionnent la couleur de tracé. Les 4 gros pots fixent la couleur de fond des caractéres (mode étendu): Le blanc (invariable) et 3 de couleurs programmables. Pour chan-

ger la couleur d'un gros pot, prendre la couleur d'un petit et la déposer sur le goulot du gros pot à programmer. La sélection de la couleur de fond s'effectue en mettant l'index de la main sur le corps d'un des gros pots. La couleur du bord donne à tout moment la couleur de fond en cours.

* Les caractères peuvent être pris dans les lignes inférieures.

* La poubelle sert à effacer le dessin.

* Le graphisme "feuille de papier" donne accès aux options entrée/sortie.

* Le graphisme "imprimante" donne accès à la recopie d'écran sur imprimante 1526 ou MPS-801/802

* Les graphismes "cassette supérieure" (SAVE) et "cassette inférieure" (LOAD) donnent accés à la sauvegarde sur cassette ou disquette, au choix.

* Le graphisme "flèche en haut" ramène au mode normal

* Le graphisme "flèche en bas" n'est accessible qu'en mode normal et signifie: passer à la programmation des caractères programmables. Ces derniers sont les 16 caractères du bas de l'écran.

* Pour programmer un caractère, le saisir avec la main et le poser sur le damier de programmation. Ensuite, à chaque appui sur le bouton de tir, le point du damier sous le curseur change de couleur. Pour revenir au mode normal, mettre le curseur au bord du damier et appuyer.

* La fonction "OUPPS!!" se produit à chaque enfoncement de la touche "CTRL" du clavier. Cette fonction annule la dernière modification effectuée au dessin.

* NOTE: Ne soyez pas effrayés par la longueur du programme! Son but étant essentiellement didactique, il est truffé de lignes "REM" et même ":". Aucune de ces lignes n'est nécessaire au programme: Ne les entrez donc pas!!

```
0 IF FL GOTO 20250: REM EXECUTE APRES LOAD
1 REM ************************************
2 REM *           S. V. P.               *
3 REM *    GENERATEUR DE  DESSINS        *
4 REM *         PAR P. BRUNET            *
5 REM *      COPYRIGHT 1984 BCM          *
6 REM ************************************
9 :
10 REM PROTECTION DU SOMMET DE BASIC EN 38912 ($9800)
12 POKE 51,0: POKE 52,152: POKE 55,0: POKE 56,152: CLR
13 REM SAUVE LES POINTEURS DE BASIC
14 FOR I=0 TO 5: POKE 51456+I,PEEK(43+I): NEXT I
15 :
```

```
16 REM JOYSTICK OU PADDLES
18 PRINT"[CLR][C.BAS][C.BAS]JOYSTICK (0) OU POIGNEES (1)":INPUT PL
19 IF PL<0 OR PL>1 THEN 18
21 :
22 REM INITIALISATION DES VARIABLES
23 POKE 1022, PEEK(45): POKE 1023, PEEK(46): FL=1
24 PG=0:I=0:J=0:K=0:L=0:CC=11:FO=0:DE=15:CP=46
25 B1=0          :REM CHOIX BANK#3
26 EC=55176:E1=16 :REM ECRAN    EN 50176
27 CO=55296       :REM COULEUR EN 55296
28 CA=49152:C1= 0 :REM CARACT. EN 49152
29 DIMA$(19):VIC=53248:OG=51019
30 DIMXA(5),XD(5),YA(5),YD(5),PX(10),PY(10),BI(15),Z(159)
31 :
32 REM INITIALISATION DE A$(N)
34 A$(0)="[NOIR]L[VERT L.]^[JAUNE L.]3[VERT L.]^[POURPRE L.]3[VERT L.]^
     [CYAN L.]3[VERT L.]^[NOIR]?[VERT L.]^[NOIR]>[VERT L.]^[ROUGE]?[VE
     RT L.]^[CYAN]?[VERT L.]^[POURPRE]?[VERT L.]^[VERT]?[VERT L.]^"
35 A$(1)="[BLEU]?[VERT L.]^[JAUNE]?[VERT L.]^[ORANGE]?[VERT L.]^[ORANGE
     L.]?[VERT L.]^[ROSE]?[VERT L.]^[CYAN L.]?[VERT L.]^[POURPRE L.]?
     [VERT L.]^[VERT L.]?[VERT L.]^[BLEU L.]?[VERT L.]^[JAUNE L.]?[VER
     T L.]^"
36 A$(2)="[BLANC]^[VERT L.]^[JAUNE L.]^[VERT L.]^[POURPRE L.]^[VERT L.]
     ^[CYAN L.]^[VERT L.]^^[BLEU]@[VERT L.]^[BLEU]A[VERT L.]^[BLEU]B[V
     ERT L.]^[BLEU]C[VERT L.]^[BLEU]D[VERT L.]^[BLEU]E"
37 A$(3)="[VERT L.]^[BLEU]F[VERT L.]^[BLEU]G[VERT L.]^[BLEU]H[VERT L.]^
     [BLEU]I[VERT L.]^[BLEU]J[VERT L.]^[BLEU]K[VERT L.]^[BLEU]L[VERT L
     .]^[BLEU]M[VERT L.]^[BLEU]N[VERT L.]^^"
38 A$(4)="[VERT L.]^^^^^^^^O^[BLEU]P[VERT L.]^[BLEU]Q[VERT L.]^[BLEU]R[
     VERT L.]^[BLEU]S[VERT L.]^[BLEU]T[VERT L.]^"
39 A$(5)="[BLEU]U[VERT L.]^[BLEU]V[VERT L.]^[BLEU]W[VERT L.]^[BLEU]X[VE
     RT L.]^[BLEU]Y[VERT L.]^[BLEU]Z[VERT L.]^[BLEU][[VERT L.]^[BLEU]\
     [VERT L.]^[BLEU]][VERT L.]^[BLEU]^[VERT L.]^"
40 A$(6)="[VERT L.]^^^^^^^^^[BLEU] [VERT L.]^[BLEU] [VERT L.]^[BLEU]![V
     ERT L.]^[BLEU]"+CHR$(34)+CHR$(34)+"[C.GAUCHE][VERT L.]^[BLEU]#[VE
     RT L.]^[BLEU]$"
41 A$(7)="[VERT L.]^[BLEU]%[VERT L.]^[BLEU]&[VERT L.]^[BLEU]'[VERT L.]^
     [BLEU]([VERT L.]^[BLEU])[VERT L.]^[BLEU]*[VERT L.]^[BLEU]+[VERT L
     .]^[BLEU],[VERT L.]^[BLEU]-[VERT L.]^^"
42 A$(8)="[VERT L.]^^^^^^^^[ROUGE].[VERT L.]^[ROUGE]/[VERT L.]^[ROUGE]0
     [VERT L.]^[ROUGE]1[VERT L.]^[ROUGE]2[VERT L.]^[ROUGE]3[VERT L.]^"

43 A$(9)="[ROUGE]4[VERT L.]^[ROUGE]5[VERT L.]^[ROUGE]6[VERT L.]^[ROUGE]
     7[VERT L.]^[ROUGE]8[VERT L.]^[ROUGE]9[VERT L.]^[ROUGE]:[VERT L.]^
     [ROUGE];[VERT L.]^[ROUGE]<[VERT L.]^[ROUGE]="
44 A$(10)="[JAUNE]^^^^^^^^^^^^^^^^^^^^"
45 A$(11)="[JAUNE]^^^^^^^^^^^^^^^^^^^^"
46 A$(12)="[JAUNE]^^^[ROUGE]UUUU[JAUNE]^^^^^^^^^^^^^^"
47 A$(13)="[JAUNE]^^^^^^^^^^^^^^^^^^^^"
48 A$(14)="[JAUNE]^^^[ROUGE]UUUU[JAUNE]^[ROUGE].[JAUNE]^[ROUGE]/[JAUNE]
     ^[ROUGE]0[JAUNE]^[ROUGE]1[JAUNE]^[ROUGE]2[JAUNE]^[ROUGE]3[JAUNE]^
     "
49 A$(15)="[ROUGE]4[JAUNE]^[ROUGE]5[JAUNE]^[ROUGE]6[JAUNE]^[ROUGE]7[JAU
     NE]^[ROUGE]8[JAUNE]^[ROUGE]9[JAUNE]^[ROUGE]:[JAUNE]^[ROUGE];[JAUN
     E]^[ROUGE]<[JAUNE]^[ROUGE]=[JAUNE]^"
50 A$(16)="[JAUNE]^^^[ROUGE]UUUU[JAUNE]^^^^^^^^^^^^^^"
51 A$(17)="[JAUNE]^^^^^^^^^^^^^^^^^^^^"
52 A$(18)="[JAUNE]^^^[ROUGE]UUUU[JAUNE]^^^^^^^^^^^^^^"
53 A$(19)="[JAUNE]^^^^^^^^^^^^^^^^^^^^"
54 :
55 REM CREATION DES ROUTINE EN L.M.
56 GOSUB 30010
57 :
58 REM INITIALISATION DU PROGRAMME
59 GOSUB 1010
60 :
61 REM DEBUT DU PROGRAMME
62 GOTO 2030
63 :
64 REM DATA POUR ROUTINE POIGNEES
65 DATA162,001,120,173,002,220,141,231
66 DATA002,169,192,141,002,220,169,128
67 DATA141,000,220,160,128,234,136,016
68 DATA252,173,025,212,157,232,002,173
```

```
 69 DATA026,212,157,234,002,173,000,220
 70 DATA009,128,141,236,002,169,064,202
 71 DATA016,222,173,231,002,141,002,220
 72 DATA173,001,220,141,237,002,088,096
 74 REM DATA POUR ROUTINE SAUVE ECRAN
 75 DATA076,006,200,076,015,200,169,076
 76 DATA133,106,169,200,076,021,200,169
 77 DATA100,133,106,169,200,120,133,107
 78 DATA162,000,160,000,177,106,133,095
 79 DATA200,177,106,133,096,200,177,106
 80 DATA133,090,200,177,106,133,091,200
 81 DATA177,106,133,088,200,177,106,133
 82 DATA089,200,138,072,152,072,032,191
 83 DATA163,104,168,104,170,232,224,004
 84 DATA208,210,088,096,000,196,032,199
 85 DATA032,155,000,216,032,219,064,158
 86 DATA032,208,037,208,112,158,112,193
 87 DATA000,194,000,159,000,152,032,155
 88 DATA032,199,032,155,064,158,032,219
 89 DATA107,158,112,158,037,208,112,158
 90 DATA000,159,000,194
 91 REM DATA POUR ROUTINE EFFACE ECRAN
 92 DATA001,169,060,133,098,169,204,133
 93 DATA099,169,000,133,100,169,208,133
 94 DATA101,173,124,200,160,000,032,175
 95 DATA200,208,251,169,000,133,098,169
 96 DATA196,133,099,169,232,133,100,169
 97 DATA199,133,101,169,015,032,175,200
 98 DATA208,251,096,145,098,230,098,208
 99 DATA002,230,099,166,099,228,101,208
100 DATA004,166,098,228,100,096
120 REM* JEU DE CARACTERES NUMERO 1   *
121 REM*********************************
122 DATA255,000,000,000,000,000,000,000
123 DATA128,128,128,128,128,128,128,128
124 DATA128,064,032,016,008,004,002,001
125 DATA001,002,004,008,016,032,064,128
126 DATA255,128,128,128,128,128,128,128
127 DATA255,064,032,016,008,004,002,001
128 DATA255,002,004,008,016,032,064,128
129 DATA128,192,160,144,136,132,130,129
130 DATA129,130,132,136,144,160,192,128
131 DATA129,066,036,024,024,036,066,129
132 DATA255,192,160,144,136,132,130,129
133 DATA255,130,132,136,144,160,192,128
134 DATA255,066,036,024,024,036,066,129
135 DATA129,194,164,152,152,164,194,129
136 DATA255,194,164,152,152,164,194,129
199 REM* JEU DE CARACTERES NUMERO 2   *
200 REM*********************************
201 DATA000,000,000,000,000,000,000,000
202 DATA240,240,240,240,000,000,000,000
203 DATA015,015,015,015,000,000,000,000
204 DATA000,000,000,000,015,015,015,015
205 DATA000,000,000,000,240,240,240,240
206 DATA255,255,255,255,000,000,000,000
207 DATA240,240,240,240,015,015,015,015
208 DATA240,240,240,240,240,240,240,240
209 DATA015,015,015,015,015,015,015,015
210 DATA015,015,015,015,240,240,240,240
211 DATA000,000,000,000,255,255,255,255
212 DATA255,255,255,255,015,015,015,015
213 DATA255,255,255,255,240,240,240,240
214 DATA240,240,240,240,255,255,255,255
215 DATA015,015,015,015,255,255,255,255
216 DATA255,255,255,255,255,255,255,255
299 REM* JEU DE CARACTERES NUMERO 3   *
300 REM*********************************
301 DATA160,080,160,080,000,000,000,000
302 DATA010,005,010,005,000,000,000,000
303 DATA000,000,000,000,010,005,010,005
304 DATA000,000,000,000,160,080,160,080
305 DATA170,085,170,085,000,000,000,000
```

```
306 DATA160,080,160,080,010,005,010,005
307 DATA160,080,160,080,160,080,160,080
308 DATA010,005,010,005,010,005,010,005
309 DATA010,005,010,005,160,080,160,080
310 DATA000,000,000,000,170,085,170,085
311 DATA170,085,170,085,010,005,010,005
312 DATA170,085,170,085,160,080,160,080
313 DATA160,080,160,080,170,085,170,085
314 DATA010,005,010,005,170,085,170,085
315 DATA170,085,170,085,170,085,170,085
399 REM* JEU DE CARACTERES PROGAM.    *
400 REM*********************************
401 DATA255,126,060,024,000,000,000,000
402 DATA001,003,007,015,015,007,003,001
403 DATA000,000,000,000,024,060,126,255
404 DATA128,192,224,240,240,224,192,128
405 DATA255,127,063,031,015,007,003,001
406 DATA255,126,060,024,024,060,126,255
407 DATA255,254,252,248,240,224,192,128
408 DATA001,003,007,015,031,063,127,255
409 DATA129,195,231,255,255,231,195,129
410 DATA128,192,224,240,248,252,254,255
411 DATA255,127,063,031,031,063,127,255
412 DATA255,255,255,255,255,231,195,129
413 DATA255,254,252,248,248,252,254,255
414 DATA129,195,231,255,255,255,255,255
415 DATA000,060,112,066,066,112,060,000
416 DATA000,060,126,126,126,126,060,000
417 DATA000,126,036,036,066,066,126,000
418 DATA000,126,060,060,126,126,126,000
499 REM* DEFINITION LUTIN CARACTERE   *
500 REM*********************************
501 DATA000,000,000,000,000,000,000,000
502 DATA000,000,000,000,000,000,000,000
503 DATA000,000,000,000,000,000,000,000
504 DATA000,000,000,000,000,000,000,000
505 DATA000,000,000,000,000,000,000,000
506 DATA000,000,000,000,000,000,000,000
507 DATA000,000,000,000,000,000,000,000
508 DATA000,000,000,000,000,000,000,000
599 REM* DEFINITION LUTIN MENU       *
600 REM*********************************
601 DATA024,000,000,060,000,000,126,000
602 DATA000,255,000,000,024,000,000,024
603 DATA000,000,024,000,000,024,063,240
604 DATA024,080,008,000,072,004,000,056
605 DATA004,000,008,004,024,016,008,024
606 DATA016,008,024,016,008,024,008,004
607 DATA024,007,254,255,000,000,126,000
608 DATA000,060,000,000,024,000,000,000
699 REM* DEFINITION LUTIN CASSETTES   *
700 REM*********************************
701 DATA000,000,032,000,000,096,028,056
702 DATA224,034,068,252,034,068,252,028
703 DATA056,224,008,016,096,007,224,032
704 DATA000,000,000,000,000,000,000,000
705 DATA000,000,000,000,000,000,000,000
706 DATA000,032,000,000,048,028,056,056
707 DATA034,068,252,034,068,252,028,056
708 DATA056,008,016,048,007,224,032,000
799 REM* DEFINITION LUTIN IMPRIMANTE  *
800 REM*********************************
801 DATA000,000,000,000,000,000,000,000
802 DATA000,000,000,000,003,248,000,005
803 DATA004,000,008,130,000,008,130,000
804 DATA063,131,000,049,004,128,041,252
805 DATA064,036,000,032,034,000,016,017
806 DATA255,248,009,000,008,004,128,004
807 DATA002,128,004,001,064,002,000,255
808 DATA254,000,000,000,000,000,000,000
899 REM* DEFINITION LUTIN POUBELLE    *
900 REM*********************************
901 DATA000,127,000,000,255,128,001,255
```

```
902 DATA128,001,255,000,000,254,000,000
903 DATA254,000,001,129,128,002,000,064
904 DATA002,000,064,003,129,192,003,255
905 DATA192,003,255,192,003,255,192,003
906 DATA255,192,003,255,192,003,255,192
907 DATA003,255,192,003,255,192,003,255
908 DATA192,001,255,240,000,126,048,000
913 REM* DEFINITION LUTIN MAIN        *
914 REM*********************************
915 DATA000,000,000,000,000,000,031,255
916 DATA192,039,255,224,039,255,248,031
917 DATA255,252,000,031,255,000,063,255
918 DATA000,063,255,000,031,255,000,063
919 DATA255,000,063,255,000,031,255,000
920 DATA063,254,000,063,252,000,031,248
921 DATA000,007,224,000,000,000,000,000
922 DATA000,000,000,000,000,000,000,000
923 :
951 REM   DEFINITION DU TABLEAU DE DEPLACEMENT DE
952 REM   LA MAIN DANS UNE MAILLE CARACTERE.
954 DATA8,8,8,-8,8,0,0,0,-8,8,-8,-8,-8,0,0,0,0,8,0,-8,0,0
955 :
956 REM DEFINITION DES CARACTERES PROGRAMMABLES
958 DATA15,17,16,20,18,23,21,26,19,24,22,27,25,29,28,30
1000 REM*********************************
1001 REM* SOUS-ROUTINE INITIALISATION *
1002 REM*********************************
1008 REM SELECTION DU BANC NUMERO 3 POUR VIC II
1010 POKE 56578, PEEK(56578) OR 3
1020 POKE 56576, (PEEK(56576) AND 252) OR B1
1027 :
1028 REM SELECTION DE L'ECRAN EN BANC 3 + 1024
1030 POKE VIC+24, (PEEK(VIC+24) AND 15) OR E1
1037 :
1038 REM PASSAGE EN MODE ETENDU
1040 POKE VIC+17, PEEK(VIC+17) OR 64
1047 :
1048 REM DEFINITION COULEUR DE BORD
1050 POKE VIC+32, 1 :REM BLANC
1057 :
1058 REM DEFINITION COULEURS DE FOND
1060 POKE VIC+33,  1:REM #3=BLANC
1070 POKE VIC+34, 15:REM #0=GRIS 3 CLAIR
1080 POKE VIC+35, 12:REM #1=GRIS 2 MOYEN
1090 POKE VIC+36, 11:REM #2=GRIS 1 FONCE
1097 :
1098 REM MISE A JOUR ADRESSE ECRAN = 50176
1100 POKE 648,196
1102 :
1107 :
1108 REM ECRITURE PAGE DEMO
1110 GOSUB 10010
1117 :
1118 REM SELECTION DE LA RAM CARACTERES
1120 POKE VIC+24,(PEEK(VIC+24) AND 240) OR C1
1127 :
1128 REM CREATION DES CARACTERES
1130 FOR I=0 TO 511
1140 READ A: POKE CA+I,A
1150 NEXT I
1157 :
1158 REM CREATION DES POINTEURS DE LUTINS
1160 FOR I=0 TO 5
1170 POKE EC+1016+I,8+I
1180 NEXT I
1187 :
1188 REM ALLUMAGE DES LUTINS
1190 POKE VIC+21,63 :REM 00111111
1197 :
1198 REM CHOIX COULEURS LUTINS
1200 POKE VIC+39,11: POKE VIC+40,6: POKE VIC+41,6
1210 POKE VIC+42,6: POKE VIC+43,2: POKE VIC+44,10
```

```
1217 :
1218 REM MISE EN PLACE DES LUTINS
1220 GOSUB 12010
1227 :
1228 REM  EFFACE LA PAGE DEMO
1230 GOSUB 11010: RETURN
1500 REM*******************************
1501 REM* SOUS-ROUTINE ENTREE MOUVEMT *
1502 REM*******************************
1504 REM VARIABLES EMPLOYEES DANS LA SOUS-ROUTINE
1505 REM PL    :TEST PADDLES OU JOYSTICK
1506 REM DIR   :SAISIE DU PORT II JOYSTICK
1507 REM XA,YA :COORDONNEES POUR APPEL DEPL. JOYSTICK
1508 REM PX,PY :INCREMENT AUX COORDONNEES
1509 REM XP,YP :VALEURS DES POIGNEES DU PORT I
1510 REM EX,EY :COORDONNEES DU CARACTERE DANS L'ECRAN
1511 :
1512 REM TEST SI POIGNEES OU JOYSTICK
1514 IF PL=1 THEN 1630
1515 :
1516 REM DEBUT DE LA BOUCLE JOYSTICK
1518 REM LECTURE PORT II DU JOYSTICK
1520 DIR=PEEK(56320)
1527 :
1528 REM TEST SI TIR
1530 IF DIR=111 THEN 1690
1537 :
1538 REM TEST DE MVT SANS TIR
1540 DIR=DIR-117: IF DIR<0 THEN 1520
1547 :
1548 REM CALCUL DE LA POSITION DU LUTIN
1550 XA=XD(SP)+PX(DIR)
1560 YA=YD(SP)+PY(DIR)
1567 :
1568 REM TEST DE LIMITES ECRAN
1570 IF XA<24  THEN XA=24
1580 IF YA<50  THEN YA=50
1590 IF XA>336 THEN XA=336
1600 IF YA>242 THEN YA=242
1607 :
1608 REM APPEL DEPLACEMENT JOYSTICK ET FIN DE BOUCLE
1610 GOSUB 12214: GOTO 1520
1625 :
1626 REM DEBUT DE LA BOUCLE POIGNEES
1628 REM APPEL ENTREE PADDLES
1630 GOSUB 1810
1637 :
1638 REM CALCUL POSITION LUTIN
1640 XA=INT(XP/5)*8+24
1650 YA=INT(YP/8)*8+50
1657 :
1658 REM TEST SI TIR
1660 IF PEEK(56321)<255 THEN 1690
1667 :
1668 REM APPEL DEPLACEMENT PADDLES ET FIN DE BOUCLE
1670 GOSUB 12510: GOTO 1630
1687 :
1688 REM CALCUL POSITION CARACTERE
1690 EX=(XA-24)/8
1700 EY=(YA-50)/8
1710 RETURN
1800 REM*******************************
1801 REM* SOUS-ROUTINE ENTREE POIGNEES*
1802 REM*******************************
1804 REM VARIABLES EMPLOYEES DANS LA SOUS-ROUTINE
1805 REM XP,YP :VALEURS DES PADDLES DU PORT I
1807 :
1808 REM LECTURE POIGNEE X PORT 1
1810 SYS 679: XP=PEEK(744)
1817 :
1818 REM TEST DES LIMITES DE LECTURE
1820 IF XP<25  THEN XP=25
```

```
1830 IF XP>224 THEN XP=224
1837 :
1838 REM VALEUR DE XP ENTRE 0 ET 199
1840 XP=XP-25
1847 :
1848 REM LECTURE POIGNEE Y PORT 1
1850 SYS 679: YP=PEEK(746)
1857 :
1858 REM TEST DES LIMITES DE LECTURE
1860 IF YP<25  THEN YP=25
1870 IF YP>224 THEN YP=224
1877 :
1878 REM VALEUR DE YP ENTRE 0 ET 199
1880 YP=YP-25
1890 RETURN
2000 REM*******************************
2001 REM* DEBUT DU PROGRAMME PRINCIPAL *
2002 REM*******************************
2004 REM LISTE DES VARIABLES EMPLOYEES DANS LE PROGRAMME
2005 REM SP    :NUMERO DU LUTIN
2006 REM PG    :PERMET DE SAVOIR SI L'ON SE TROUVE DANS LA BOUCLE PRINC
     IPALE
2008 REM PE,DE :VALEUR DU CARACTERE SOUS CURSEUR
2009 REM XA,YA :POSITION DU LUTIN
2010 REM DS    :PERMET DE SAVOIR SI L'ON SE TROUVE DANS
2011 REM        LA PARTIE DESSIN OU DANS LE MENU
2012 :
2028 REM MISE A JOUR DES VARIABLES
2030 FOR I=0 TO 10 :READ PX(I),PY(I)
2040 NEXT I
2050 FOR I=0 TO 15 :READ BI(I): NEXT I
2060 SP=5:PG=1
2065 :
2066 REM DEBUT DE LA BOUCLE PRINCIPALE
2068 REM APPEL ENTREE MOUVEMENT
2070 IF LO=1 THEN 20210: REM LOAD
2071 GOSUB 1514
2072 :
2073 REM TEST CTRL POUR OOPS
2074 IF PEEK(653)=4 THEN SYS 51203
2077 :
2078 REM VALEUR DU CARACTERE SOUS CURSEUR
2080 PE=PEEK(EC+EY*40+EX)
2081 :
2087 REM TEST SI DESSIN OU OPTION
2088 REM SI OUI APPEL DEFINITION LUTIN
2089 :
2090 IF YA<210 THEN DS=1: GOSUB 15020: GOTO 2070
2091 :
2092 REM SAUVETAGE DE L'ECRAN
2093 SYS 51200
2096 :
2097 REM TRAITEMENT DU CAS PARTICULIER
2098 REM CARACTERE PLEIN <> DU FOND
2100 IF XA=328 AND YA=226 THEN 2110
2101 :
2102 REM TEST SI FOND
2105 IF PE=30 AND XA>80 THEN 2070
2107 :
2108 REM SI OPTION APPEL OPTIONS
2110 IF XA<88 THEN GOSUB 14012: GOTO 2070
2117 :
2118 REM TEST DE CHOIX DE COULEUR
2120 IF PE>61 THEN GOSUB 16013: GOTO 2070
2127 :
2128 REM APPEL DEFINITION LUTIN
2130 DE=PE: DS=0: GOSUB 15020
2137 :
2138 REM FIN DE LA BOUCLE PRINCIPALE
2140 GOTO 2070
10000 REM*****************************
10001 REM* SOUS-ROUTINE PAGE DEMO     *
10002 REM*****************************
```

```
10010 PRINT"[CLR][NOIR]";
10020 PRINT"OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO";
10030 PRINT"[NOIR]OCDACDACDBCDAOO/[ROUGE]OOOOOOOOOO[NOIR]1OOOOOOO[NOIR]
      ^OOOOO";
10040 PRINT"[NOIR]OAAAAAAAAOBAAOO/[ORANGE]OOOOOOOOOO[NOIR]1OOOOOOOO[CYA
      N L.]^OOOO";
10050 PRINT"[NOIR]OAAAAAAAAGOOAOO/[JAUNE]OOOOOOOOOO[NOIR]1OOOOOOOOO[POU
      RPRE L.]^OOO";
10060 PRINT"[NOIR]OAAAAAAAAABOAOO/[VERT]OOOOOOOOOO[NOIR]1OOOOOOOOOO[JAU
      NE L.]^OO";
10070 PRINT"[NOIR]OAA@@OAAAGCAAOO/[BLEU]OOOOOOOOOO[NOIR]1OOOOOOOOOO[BL
      ANC]^O";
10080 PRINT"[NOIR]OH@@@FOHFOHFOOO/[POURPRE]OOOOOOOOOO[NOIR]1OOOOOOOOOO
      OO";
10090 PRINT"[NOIR]O@@@@QOO@OO@QOOOQ.....OOOOOOOOOOOOO";
10100 PRINT"OCDBC@ACD@@@ACDBC@ACD@@@ACDBCDACD@@AOOOO";
10110 PRINT"OAAOOOAAADFAAAAOOOAAAD@FOAAOBAAHFAKDOOOO";
10120 PRINT"OAAGCAAAADAAAAAGCAAAA@@AOAAGOOA@AAAOOOOO";
10130 PRINT"OAAHAAAAAAAAAAAHAAAAADFOOAAABOAOAAAOOOOO";
10140 PRINT"OAAAAAAAA@@OAAAAAAAAA@@@AAAGCAAOAAAOOOOO";
10150 PRINT"OHFOHFOH@@@FOHFOHFOH@@@FOHFOHFOOHFOOOOOO";
10160 PRINT"O@OO@OO@@@@OO@OO@OO@@@@OO@OO@OOO@OOOOOO";
10170 PRINT"OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO";
10180 PRINT"O[PXQ[OVOOVV[VVV[POWZWORUQ[Q[OOOOOOOOOO";
10190 PRINT"OTVOOVOVOOVVVVVVTVTWTWOW]OVOVOOOOOO[CYAN]57OOO";
10200 PRINT"[NOIR]OTPOQTOTPOQOTPTPTPOQOQTQQQTOPOOOOO[CYAN]5^^7OOO";
10210 PRINT"[NOIR]OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO[CYAN]5^^7OOO";
10220 PRINT"[NOIR]O#%*_O)_*%´"CHR$(34)"%%´"CHR$(34)")_%O%O*_OOOOOOO[CY
      AN]5^42^OOO";
10230 PRINT"[NOIR]O"CHR$(34)"%*OO&O- +%%%+%&O%O%O*OOOOOOOOOOOOOOOO";
10240 PRINT"O# # O O     O  # # # # OOOOOOOOOOOOOOOO";
10250 PRINT"OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO";
10260 FOR I=EC+56 TO EC+256 STEP 40
10270 POKEI,30:POKEI+1,30:POKEI+2,237:POKEI+3,237:POKEI+4,173:POKEI+5,1
      73
10280 POKEI+6,109:POKEI+7,109:POKEI+8,45:POKEI+9,45
10290 NEXT I
10300 POKEEC+74,237:POKEEC+75,173:POKEEC+76,109:POKEEC+77,45
10310 POKEEC+115,173:POKEEC+116,109:POKEEC+117,45
10320 POKEEC+156,109:POKEEC+157,45:POKEEC+197,45
10330 RETURN
11000 REM*******************************
11001 REM* SOUS-ROUTINE EFFACE ECRAN  *
11002 REM*******************************
11009 REM RETABLIT LA TABLE DES ADRESSES LIGNES
11010 FOR I=0 TO 25: PRINT: NEXT I
11018 :
11019 REM EFFACE L´ECRAN
11020 SYS 51324
11028 :
11030 REM AFFICHE LE MENU
11040 GOSUB 13010
11050 RETURN
12000 REM*******************************
12001 REM* SOUS-ROUTINE DEMO LUTINS   *
12002 REM*******************************
12008 REM MISE EN PLACE LUTIN 5
12010 SP=5:XA=24:YA=50: GOSUB 12214
12017 :
12018 REM MISE EN PLACE LUTIN 1
12020 SP=1:XA=48:YA=50: GOSUB 12214
12027 :
12028 REM MISE EN PLACE LUTIN 2
12030 SP=2:XA=72:YA=50: GOSUB 12214
12037 :
12038 REM MISE EN PLACE LUTIN 3
12040 SP=3:XA=96:YA=50: GOSUB 12214
12047 :
12048 REM MISE EN PLACE LUTIN 4
12050 SP=4:XA=120:YA=50: GOSUB 12214
12057 :
12058 REM DEFINITION DES LUTINS
```

```
12060 FOR I=CA+512 TO CA+895
12070 READ A: POKE I,A
12080 NEXT I
12087 :
12088 REM DEPLACEMENT LUTIN 5
12090 SP=5:XA=266:YA=178: GOSUB 12510
12097 :
12098 REM DEPLACEMENT LUTIN 1
12100 SP=1:XA=24:YA=228: GOSUB 12510
12101 GOSUB 12510
12107 :
12108 REM DEPLACEMENT LUTIN 2
12110 SP=2:XA=24:YA=250: GOSUB 12510
12117 :
12118 REM DEPLACEMENT LUTIN 3
12120 SP=3:XA=24:YA=250: GOSUB 12510
12127 :
12128 REM DEPLACEMENT LUTIN 4
12130 SP=4:XA=48:YA=228: GOSUB 12510
12140 RETURN
12200 REM******************************
12201 REM* SOUS-ROUTINE DEPL. JOYSTICK*
12202 REM******************************
12204 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
12205 REM SX,SY :INCREMENT DE DEPLACEMENT DU LUTIN
12206 REM XA,YA :COORDONNEES D'ARRIVEE
12207 REM XD,YD :COORDONNEES DE DEPART
12208 REM X ,Y  :COORDONNEES DE TRAVAIL POUR
12209 REM        APPEL DEPLACEMENT LUTIN
12210 REM SP    :NUMERO DU LUTIN
12211 :
12213 REM INITIALISATION DES VARIABLES
12214 SY=2:SX=2:XA(SP)=XA:YA(SP)=YA
12217 :
12218 REM TEST SI DEPLACEMENT NEGATIF
12220 IF XA(SP) < XD(SP) THEN SX=-2
12230 IF YA(SP) < YD(SP) THEN SY=-2
12237 :
12238 REM INITIALISE VARIABLES DE TRAVAIL
12240 X=XD(SP) : Y=YD(SP)
12245 :
12246 REM DEBUT DE LA BOUCLE DE MOUVEMENT
12247 :
12248 REM TEST DE FIN DE MOUVEMENT HORIZONTAL
12250 IF X=XA(SP) THEN SX=0
12257 :
12258 REM TEST DE FIN DE MOUVEMENT VERTICAL
12260 IF Y=YA(SP) THEN SY=0
12267 :
12268 REM CALCUL DES NOUVELLES COORDONEES
12270 X=X+SX: Y=Y+SY
12277 :
12278 REM APPEL DEPLACEMENT LUTIN
12280 GOSUB 12713
12287 :
12288 REM TEST ET FIN DE BOUCLE MOUVEMENT
12290 IF SX=0 AND SY=0 THEN 12310
12300 GOTO 12250
12307 :
12308 REM NOUVELLES COORDONNEES DE DEPART
12310 XD(SP)=XA(SP)
12320 YD(SP)=YA(SP)
12330 RETURN
12500 REM******************************
12501 REM* SOUS-ROUTINE DEPL. PADDLES *
12502 REM******************************
12507 REM CALCUL DES INDICES POUR
12508 REM APPEL DEPLACEMENT LUTIN
12510 X=XA: Y=YA
12520 GOSUB 12713
12530 RETURN
12700 REM******************************
```

```
12701 REM* SOUS-ROUTINE DEPL. LUTINS  *
12702 REM*******************************
12704 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
12705 REM X1,X2 :PARTIES ENTIERE ET DECIMALE DE X/256
12706 REM X ,Y  :COORDONNEES DU LUTIN
12707 REM SP    :NUMERO DU LUTIN
12708 REM PG    :PERMET DE SAVOIR SI L'ON SE TROUVE DANS
12709 REM        LA BOUCLE PRINCIPALE
12710 :
12711 REM CALCUL DE X SUR DEUX OCTETS
12713 X1=INT(X/256)
12720 X2=X-X1*256
12726 :
12727 REM SI DANS DEMO : UN LUTIN
12728 REM SINON DEUX LUTINS (0 ET SP)
12730 IF PG=0 THEN 12780
12736 :
12737 REM PROGRAMMATION DU BIT DE POIDS FORT
12738 REM DU LUTIN 0 DANS LE REGISTRE VIC+16
12740 IF X1=0 THEN POKE VIC+16, PEEK(VIC+16) AND 254
12750 IF X1=1 THEN POKE VIC+16, PEEK(VIC+16) OR 1
12756 :
12757 REM PROGRAMMATION DES REGISTRES DE POSITION
12758 REM POUR LE LUTIN 0
12760 POKE VIC, X2
12770 POKE VIC+1, Y
12776 :
12777 REM PROGRAMMATION DU BIT DE POIDS FORT
12778 REM DU LUTIN SP DANS LE REGISTRE VIC+16
12780 IF X1=0 THEN POKE VIC+16, PEEK(VIC+16) AND (255-2^SP)
12790 IF X1=1 THEN POKE VIC+16, PEEK(VIC+16) OR (2^SP)
12796 :
12797 REM PROGRAMMATION DES REGISTRES DE POSITION
12798 REM POUR LE LUTIN SP
12800 POKE VIC+2*SP,X2
12810 POKE VIC+2*SP+1,Y
12820 RETURN
13000 REM*******************************
13001 REM* SOUS-ROUTINE MENU          *
13002 REM*******************************
13004 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
13005 REM PAGE  :-0- MENU NORMAL
13006 REM        -1- PROGRAMMATION DES CARACTERES
13007 REM A$(19):TABLEAU CONTENANT LES MENUS
13008 REM MISE EN PLACE DU CURSEUR
13010 POKE 211,0: POKE 214,20:SYS 58732
13017 :
13018 REM BOUCLE D'AFFICHAGE
13020 FOR I=0 TO 9
13030 PRINTA$(I+PAGE*10);
13040 NEXT I
13046 :
13047 REM AFFICHAGE DU DERNIER CARACTERE ET DE LA POUBELLE
13050 POKE EC+999,30
13060 IF PAGE=0 THEN POKE CO+999,13: POKE VIC+9,228
13070 IF PAGE=1 THEN POKE CO+999,7: POKE VIC+9,250
13076 :
13077 REM DANS LA PAGE 0 MISE A JOUR DES COULEURS DE FOND
13080 IF PAGE=0 THEN FOR I=0 TO 2: POKE CO+802+2*I, PEEK(VIC+34+I): NEX
      T I
13081 IF PAGE=0 THEN FOR I=0 TO 2: POKE CO+842+2*I, PEEK(VIC+34+I): NEX
      T I
13087 :
13088 REM SI PAGE=1 APPEL PROGRAMMATION DES CARACTERES
13090 IF PAGE=1 THEN GOSUB 19016
13100 RETURN
14000 REM*******************************
14001 REM* SOUS-ROUTINE OPTIONS       *
14002 REM*******************************
14004 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
14005 REM XA,YA :POSITION DU LUTIN
14006 REM PM    :FAIT LA DIFFERENCE ENTRE LA PROGRAMMATION
```

```
14007 REM         OU LE CHOIX DE LA COULEUR DE FOND
14008 REM FO     :NUMERO DU FOND CONCERNE
14009 :
14010 REM APPEL MENU PAGE 0
14012 IF XA=24 AND YA=226 THEN PAGE=0: GOSUB 13010: GOTO 14150
14017 :
14018 REM APPEL MENU PAGE 1
14020 IF XA=24 AND YA=242 THEN PAGE=1: GOSUB 13010: GOTO 14150
14027 :
14028 REM APPEL EFFACE ECRAN
14030 IF XA=56 AND YA=226 THEN GOSUB 11010: GOTO 14150
14040 IF XA=56 AND YA=234 THEN GOSUB 11010: GOTO 14150
14050 IF XA=56 AND YA=242 THEN GOSUB 11010: GOTO 14150
14057 :
14058 REM APPEL OPTION ENTREES/SORTIES
14060 IF XA=40 AND (YA=234 OR YA=242) THEN GOSUB 17020: GOTO 14150
14070 IF XA=32 AND (YA=234 OR YA=242) THEN GOSUB 17020: GOTO 14150
14077 :
14078 REM TEST CHOIX DES FONDS
14080 IF YA>218 THEN 14150
14087 :
14088 REM TEST SI PROGRAMATION OU CHOIX DU FOND
14090 IF YA=210 THEN PM=1
14100 IF YA=218 THEN PM=0
14107 :
14108 REM APPEL FOND AVEC FO=0
14110 IF XA=24 THEN FO=0: GOSUB 18013: GOTO 14150
14117 :
14118 REM APPEL FOND AVEC FO=1
14120 IF XA=40 THEN FO=1: GOSUB 18013: GOTO 14150
14127 :
14128 REM APPEL FOND AVEC FO=2
14130 IF XA=56 THEN FO=2: GOSUB 18013: GOTO 14150
14137 :
14138 REM APPEL FOND AVEC FO=3
14140 IF XA=72 THEN FO=3: GOSUB 18013
14150 RETURN
15000 REM*******************************
15001 REM* SOUS-ROUTINE DEF. LUTIN     *
15002 REM*******************************
15004 REM VARIABLES UTILISEES DANS LA SOUS ROUTINE
15005 REM EX,EY :COORDONNEES DU CARACTERE SOUS CURSEUR
15006 REM DS     :PERMET DE SAVOIR SI L'ON EST DANS
15007 REM         LE MENU OU DANS LE DESSIN
15008 REM DE     :CARACTERE SOUS CURSEUR
15009 REM FO     :FOND UTILISE
15010 REM CC     :COULEUR COURANTE
15011 REM CA     :ADRESSE RAM CARACTERES
15012 REM EC     :ADRESSE RAM ECRAN
15013 REM CO     :ADRESSE RAM COULEUR
15014 :
15016 REM SI DANS DESSIN ,ON POKE DANS
15017 REM L'ECRAN .SINON ,ON PROGRAMME LE
15018 REM LUTIN NUMERO 0.
15020 IF DS=1 THEN 15080
15025 :
15026 REM PROGRAMMATION DU LUTIN 0
15027 REM LE CARACTERE EST DECALE D'UN
15028 REM BIT VERS LA DROITE
15030 FOR I=0 TO 7
15040 POKE 49664+3*I,(PEEK(CA+8*DE+I)AND254)/2
15050 POKE 49664+3*I+1,(PEEK(CA+8*DE+I)AND1)*128
15060 NEXT I
15067 :
15068 REM FIN PROGRAMATION LUTIN 0
15070 GOTO 15110
15077 :
15078 REM POKE LE CARACTERE DANS L'ECRAN
15080 POKE EC+EY*40+EX, DE
15087 :
15088 REM POKE LA COULEUR DE FOND
15090 POKE EC+EY*40+EX,(PEEK(EC+EY*40+EX) AND 63) OR (FO*64)
```

```
15097 :
15098 REM POKE LA COULEUR DU CARACTERE
15100 POKE CO+EY*40+EX, CC
15110 RETURN
16000 REM*******************************
16001 REM* SOUS-ROUTINE COULEUR        *
16002 REM*******************************
16004 REM VARIABLES UTILISEES DANS LA SOUS ROUTINE
16005 REM PE     :CARACTERE SOUS CURSEUR
16007 REM CC     :COULEUR COURANTE
16008 REM EX,EY :COORDONNEES DU CARACTERE SOUS CURSEUR
16009 REM CO     :ADRESSE RAM COULEUR
16010 :
16011 REM TEST POUR COULEUR BLANCHE
16013 IF PE=62 THEN CC=1: GOTO 16030
16017 :
16018 REM TEST POUR AUTRES COULEURS
16020 CC=PEEK(CO+EY*40+EX)
16027 :
16028 REM CHANGE LA COULEUR DU LUTIN 0
16030 POKE VIC+39,CC
16040 RETURN
17000 REM*******************************
17001 REM* SOUS-ROUTINE ENTREE-SORTIE *
17002 REM*******************************
17004 REM VARIABLES UTILISEES DANS LA SOUS ROUTINE
17005 REM PG     :PERMET DE SAVOIR SI L'ON SE TROUVE DANS
17006 REM          LA BOUCLE PRINCIPALE
17007 REM SP     :NUMERO DU LUTIN
17008 REM XA,YA :COORDONNEES D'ARRIVEE DU LUTIN
17009 REM PAGE   :CHOIX DU MENU
17010 :
17011 REM MISE EN PLACE DES LUTINS
17020 SP=1: XA=24: YA=186: GOSUB 12510
17030 SP=2: XA=48: YA=186: GOSUB 12510
17040 SP=3: XA=72: YA=186: GOSUB 12510
17050 SP=5: XA=56: YA=194: GOSUB 12510
17055 :
17056 REM DEBUT DE LA BOUCLE DE TEST
17058 REM APPEL ENTREE MOUVEMENT
17060 GOSUB 1514
17067 :
17068 REM APPEL MENU PAGE 0
17070 IF XA=24 AND YA=186 THEN PAGE=0 : GOSUB 13010: GOTO 17130
17087 :
17088 REM APPEL SAVE IMAGE
17090 IF (XA<64 OR XA>48) AND YA=186 THEN GOSUB 20010: GOTO 17130
17097 :
17098 REM APPEL LOAD IMAGE
17100 IF (XA<64 OR XA>48) AND YA=202 THEN LO=1: GOTO 17130
17107 :
17108 REM APPEL PRINT IMAGE ET FIN DE BOUCLE DE TEST
17110 IF XA=80 AND YA=194 THEN GOSUB 20610: GOTO 17130
17120 GOTO 17060
17127 :
17128 REM REMISE DES LUTINS EN PLACE
17130 SP=1: XA=24: YA=226: GOSUB 12510
17140 SP=2: XA=24: YA=250: GOSUB 12510
17150 SP=3: XA=24: YA=250: GOSUB 12510
17160 SP=5: XA=40: YA=234: GOSUB 12510
17170 PG=1
17180 RETURN
18000 REM*******************************
18001 REM* SOUS-ROUTINE FOND          *
18002 REM*******************************
18004 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
18005 REM PM :-0- SELECTIONNE LE CHOIX DU FOND
18006 REM     -1- SELECTIONNE LA PROGRAMMATION DU FOND
18007 REM FO :NUMERO DU FOND CONCERNE
18008 REM CC :COULEUR COURANTE
18009 REM CO :ADRESSE RAM COULEUR
18010 :
```

```
18011 REM INTERDIT DE PROGRAMMER LE FOND BLANC
18013 IF PM=1 AND FO=0 THEN 18070
18017 :
18018 REM OPTION CHOIX DE FOND
18020 IF PM=0 THEN 18060
18027 :
18028 REM PROGRAMATION COULEUR FOND
18030 POKE VIC+33+FO, CC
18037 :
18038 REM PROG. COULEUR BOUTEILLE
18040 POKE CO+800+2*FO, CC
18050 POKE CO+840+2*FO, CC
18057 :
18058 REM PROG. COULEUR DU BORD
18060 POKE VIC+32, PEEK(VIC+33+FO)
18070 RETURN
19000 REM*******************************
19001 REM* SOUS-ROUTINE PROG. CARA     *
19002 REM*******************************
19004 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
19005 REM EX,EY :COORDONNEES DU CARACTERE SOUS CURSEUR
19006 REM CC    :COULEUR COURANTE
19007 REM DE    :CARACTERE SOUS CURSEUR
19008 REM CP    :CARACTERE A PROGRAMMER
19009 REM DS    :TESTE SI CURSEUR EST DANS MENU OU DESSIN
19011 REM XA,YA :COORDONNEES DU LUTIN
19012 REM EC    :ADRESSE RAM ECRAN
19013 :
19014 REM INITIALISATION VARIABLES
19016 CC=0:DS=0:DE=15
19017 :
19018 REM APPEL DEFINITION LUTINS
19020 GOSUB 15020
19027 :
19028 REM COULEUR LUTIN 0 = NOIR
19030 POKE VIC+39,CC
19035 :
19036 REM DEBUT DE LA BOUCLE DE MOUVEMENT
19037 :
19038 REM APPEL ENTREE MOUVEMENT
19040 GOSUB 1514
19047 :
19048 REM OBLIGE LUTIN DANS MENU
19050 IF YA<218 THEN 19040
19057 :
19058 REM TEST SI GRILLE OU MENU 0
19060 IF XA<=72 THEN 19110
19067 :
19068 REM VALEUR DU CARACTERE A PROGRAMMER
19070 CP=PEEK(EC+40*EY+EX): DE=CP
19077 :
19078 REM REFUSE LE CARACTERE PLEIN
19080 IF CP=30  THEN 19040
19087 :
19088 REM APPEL DEFINITION LUTIN ET FIN DE BOUCLE MOUVEMENT
19090 GOSUB 15020: GOTO19040
19108 REM SORTIE DE LA ROUTINE
19110 IF XA=24 AND YA=226 THEN 19140
19116 :
19117 REM APPEL DEPLACEMENT GRILLE
19118 REM SI LUTIN DANS GRILLE
19120 IF XA>40 THEN GOSUB 19215
19127 :
19128 REM FIN DE LA BOUCLE DE MOUVEMENT AVEC PROGRAMMATION
19130 GOTO 19040
19137 :
19138 REM APPEL MENU PAGE 0
19140 PAGE=0
19150 GOSUB 13010
19160 RETURN
19200 REM*******************************
19201 REM* SOUS-ROUTINE DEPL. GRILLE  *
19202 REM*******************************
```

```
19204 REM VARIABLES UTILISEES DANS LA SOUS-ROUTINE
19205 REM XA,YA :NOUVELLES COORDONNEES DU LUTIN
19206 REM XD,YD :ANCIENNES COORDONNEES DU LUTIN
19207 REM PX,PY :VALEURS DE DEPLACEMENT DU LUTIN
19208 REM DE    :CARACTERE SOUS CURSEUR
19209 REM PL    :TEST SI PADDLES OU JOYSTICK
19210 REM DIR   :SAISIE DU PORT II DU JOYSTICK
19211 REM XP,YP :VALEUR DES PADDLES
19212 :
19213 REM ENVOI VERS DEFINITION LUTIN
19215 DE=15: GOSUB 15020
19217 :
19218 REM APPEL PROGRAMMATION GRILLE
19220 GOSUB 19621
19227 :
19228 REM COORDONNEES DU HOME DANS LA GRILLE
19230 XA=46: YA=216
19237 :
19238 REM APPEL DEPLACEMENT POIGNEES
19240 GOSUB 12510
19247 :
19248 REM TEST SI POIGNEES
19250 IF PL=1 THEN 19400
19256 :
19257 REM MISE A JOUR DES INDICES DE DEPLACEMENT
19258 REM DU LUTIN PAR 4 PIXELS
19259 :
19260 FOR I=0 TO 10
19270 PX(I)=INT(PX(I)/2): PY(I)=INT(PY(I)/2)
19280 NEXT I
19287 :
19288 REM SAISIE DU JOYSTICK  PORT II
19290 DIR=PEEK(56320)
19297 :
19298 REM TEST SI TIR
19300 IF DIR=111 THEN 19460
19307 :
19308 REM N'AIMET QUE LES POSITIONS SANS TIR
19310 DIR=DIR-117: IF DIR<0 THEN 19290
19317 :
19318 REM CALCULE LES NOUVELLES COORDONNEES
19320 XA=XD(SP)+PX(DIR)
19330 YA=YD(SP)+PY(DIR)
19337 :
19338 REM LIMITE LE DEPLACEMENT A LA GRILLE
19340 IF XA<42  THEN XA=42
19350 IF XA>78  THEN XA=78
19360 IF YA<216 THEN YA=216
19370 IF YA>244 THEN YA=244
19377 :
19378 REM APPEL DEPLACEMENT JOYSTICK ET FIN DE BOUCLE
19380 GOSUB 12214: GOTO 19290
19397 :
19398 REM APPEL ENTREE POIGNEES
19400 GOSUB 1810
19407 :
19408 REM CALCUL DES COORDONEES
19410 XA=INT(XP/20)*4+42
19420 YA=INT(YP/25)*4+216
19427 :
19428 REM APPEL DEPLACEMENT POIGNEES
19430 GOSUB 12510
19437 :
19438 REM TEST SI TIR ET BOUCLAGE POIGNEES
19440 IF PEEK(56321)<255 THEN 19460
19450 GOTO 19400
19457 :
19458 REM TEST SI SORTIE A DROITE
19460 IF XA=78 THEN XA=88: YA=226: GOTO 19500
19467 :
19468 REM TEST SI SORTIE A GAUCHE
19470 IF XA=42 THEN XA=40: YA=226: GOTO 19500
```

```
19477 :
19478 REM APPEL PROGRAMMATION MEMOIRE
19480 GOSUB 19814
19487 :
19488 REM BOUCLAGE SOUS-ROUTINE
19490 GOTO 19250
19497 :
19498 REM TEST SI POIGNEES
19500 IF PL=1 THEN 19540
19507 :
19508 REM INDICES DE DEPLACEMENT * 2
19510 FOR I=0 TO 10
19520 PX(I)=INT(PX(I)*2): PY=INT(PY(I)*2)
19530 NEXT I
19540 RETURN
19600 REM*******************************
19601 REM* SOUS-ROUTINE PROG. GRILLE  *
19602 REM*******************************
19604 REM VARIABLES EMPLOYEES DANS LA SOUS-ROUTINE
19605 REM I ,J  :COORDONNEES DANS LA GRILLE
19606 REM PK    :POSITION DU CARACTERE DANS LA GRILLE
19607 REM OG    :ADRESSE DE L'ORIGINE DE LA GRILLE (51019)
19608 REM PU    :DONNE LA VALEUR DES DEUX BITS DANS UN
19609 REM        OCTET DE LA RAM CARACTERE, CORRESPONDANTS
19610 REM        A UN CARACTERE DANS LA GRILLE
19611 REM        EXEMPLE: 00XX0000= 48
19612 REM D1,D2 :ADRESSE DES DEUX OCTETS CONCERNES DANS LA RAM CARACTER
      ES
19613 REM E1,E2 :VALEUR DES DEUX BITS CONCERNES DANS
19614 REM        CHAQUE OCTETS DU CARACTERE GRACE A PU
19615 REM BI(15):TABLEAU PERMETTANT LA CONVERSION DES
19616 REM        BITS SELECTONNES EN UN CARACTERE QUART
19617 REM        DE MAILLE
19618 :
19619 REM DEBUT DE LA BOUCLE DE PROGRAMMATION
19621 FOR I=0 TO 3: FOR J=0 TO 3
19622 :
19623 REM POSITION DU CARACTERE
19625 PK=OG+I+40*J
19626 :
19627 REM CALCUL POUR ISOLER PAR <AND> LES 2 BITS CONCERNES
19630 PU=3*(2^(6-2*I))
19636 :
19637 REM RECHERCHE DES DEUX OCTETS
19638 REM CONCERNES DANS LA RAM CARACTERE
19640 D1=PEEK(CA+8*CP+J*2  )
19650 D2=PEEK(CA+8*CP+J*2+1)
19656 :
19657 REM ISOLE LES DEUX BITS DANS
19658 REM CHAQUE OCTET ET CALCULE BI(N)
19660 E1=(D1 AND PU) * 3/PU
19670 E2=(D2 AND PU) * 3/PU * 4
19677 :
19678 REM POKE LE CARACTERE DANS LA GRILLE
19680 POKE PK, BI(E1+E2)
19687 :
19688 REM FIN DE LA BOUCLE DE PROGRAMMATION
19690 NEXT J,I: RETURN
19800 REM*******************************
19801 REM* SOUS-ROUTINE PROG. MEMOIRE *
19802 REM*******************************
19804 REM VARIABLES EMPLOYEES DANS LA SOUS-ROUTINE
19805 REM GX,GY  :COORDONNEES DANS LA GRILLE
19806 REM XA,YA :COORDONNEES DU LUTIN
19807 REM CP    :CARACTERE A PROGRAMMER
19808 REM PX    :OCTET A PROGRAMMER
19809 REM CA    :ADRESSE DE LA RAM CARACTERE
19810 REM TEST  :BIT A PROGRAMMER
19811 :
19812 REM INDICE HORIZONTAL
19814 GX=7-(XA-46)/4
19815 :
```

```
19818 REM INDICE VERTICAL
19820 GY=(YA-216)/4
19827 :
19828 REM CALCUL DE L'OCTET A PROGRAMMER
19830 PK=CA+8*CP+GY
19837 :
19838 REM TEST= BIT A PROGRAMMER
19840 TEST=PEEK(PK) AND (2^GX)
19847 :
19848 REM SI BIT=0 ALORS BIT=1
19850 IF TEST=0 THEN POKE PK, PEEK(PK)  OR (2^GX)
19857 :
19858 REM SI BIT=1 ALORS BIT=0
19860 IF TEST THEN POKE PK, PEEK(PK) AND (255-2^GX)
19867 :
19868 REM APPEL PROGRAMMATION GRILLE
19870 GOSUB 19621
19880 RETURN
20000 REM************************************
20001 REM* SOUS-ROUTINE SAVE IMAGE EN $9800 *
20002 REM************************************
20009 REM APPEL ECRAN NORMAL
20010 GOSUB 20510
20011 :
20012 REM OPTION DISQUE OU CASSETTE
20013 PRINT"[C.BAS]DISQUE OU CASSETTE ?": POKE 198,0
20014 GET A$:IF A$="" THEN 20014
20015 IF A$="C" THEN DV=1
20016 DV=8: IF A$<>"D" THEN20013
20017 POKE 1020,DV
20018 :
20019 REM SAUVE LES POINTEURS DE BASIC
20020 FOR I=0 TO 7: POKE 51456+I, PEEK(43+I): NEXT I
20028 :
20029 REM AJUSTE LES POINTEURS POUR SAUVER LE DESSIN
20030 POKE 43,0: POKE 44,152: POKE 45,128: POKE 46,159: POKE 47,128
20040 POKE 48,159: POKE 49,128: POKE 50,159: POKE 51,0: POKE 52,160
20048 :
20049 REM SAISIE DU NOM DU DESSIN
20050 PRINT"[C.BAS][C.BAS][C.BAS]  QUEL EST LE NOM DU DESSIN A SAUVER ?
      "
20051 PRINT"     (MAX. 16 CARACTERES)"
20060 POKE198,0:INPUT NM$: IF LEN(NM$) > 16 THEN 20020
20078 :
20079 REM SAUVE LE DESSIN SUR DISQUE
20080 DV=PEEK(1020): NM$="@0:"+NM$: SAVE NM$,DV,1
20088 :
20089 REM REMET LES POINTEURS DE BASIC
20090 POKE 43, PEEK(51456): POKE 44, PEEK(51457)
20091 POKE 45, PEEK(51458): POKE 46, PEEK(51459)
20092 POKE 47, PEEK(51460): POKE 48, PEEK(51461)
20093 POKE 49, PEEK(51462): POKE 50, PEEK(51463)
20098 :
20099 REM APPEL ECRAN DESSIN
20100 GOSUB 20410: RETURN
20200 REM*******************************
20201 REM* SOUS-ROUTINE LOAD IMAGE     *
20202 REM*******************************
20209 REM APPEL ECRAN NORMAL
20210 GOSUB 20510
20211 :
20212 REM OPTION DISQUE OU CASSETTE
20213 PRINT"[C.BAS]DISQUE OU CASSETTE ?": POKE 198,0
20214 GET A$:IF A$="" THEN 20214
20215 IF A$="C" THEN DV=1
20216 DV=8: IF A$<>"D" THEN20213
20217 POKE 1020,DV
20218 :
20219 REM SAISIE DU NOM DU DESSIN
20220 PRINT"[C.BAS][C.BAS][C.BAS]  QUEL EST LE NOM DU DESSIN A CHARGER?
      "
20230 POKE 198,0: INPUT NM$: IF LEN(NM$) > 16 THEN 20220
```

```
20238 :
20239 REM CHARGE LE DESSIN DU DISQUE
20240 DV=PEEK(1020): LO=0: LOAD NM$,DV,1
20247 :
20248 REM CONNECTION DE LA LIGNE 0 (GOTO 20250 APRES LOAD)
20249 REM RETABLIT LES POINTEURS DE BASIC
20250 POKE 45,PEEK(51458): POKE 46,PEEK(51459)
20258 :
20259 REM APPEL ECRAN DESSIN
20260 GOSUB 20410: GOTO 2070
20400 REM*******************************
20401 REM* SOUS-ROUTINE ECRAN DESSIN  *
20402 REM*******************************
20409 REM SELECTION BANC 3 POUR VIC II
20410 POKE 56578, PEEK(56578) OR 3
20411 POKE 56576, (PEEK(56576) AND 252)
20418 :
20419 REM SELECTION ECRAN EN BANC 3 + 1024
20420 POKE VIC+24, (PEEK(VIC+24) AND 15) OR 16
20428 :
20429 REM MISE A JOUR ADRESSE ECRAN=50176
20430 POKE 648,196
20438 :
20439 REM PASSAGE EN MODE ETENDU
20440 POKE VIC+17, PEEK(VIC+17) OR 64
20448 :
20449 REM SELECTION DE LA RAM CARACTERE
20450 POKE VIC+24, (PEEK(VIC+24) AND 240) OR 0
20458 :
20459 REM ALLUMAGE DES LUTINS
20460 POKE VIC+21,63: REM 00111111
20468 :
20469 REM EFFACE L'ECRAN
20470 GOSUB 11010
20478 :
20479 REM RETABLIT LE DESSIN
20480 SYS 51203: RETURN
20500 REM*******************************
20501 REM* SOUS-ROUTINE ECRAN NORMAL  *
20502 REM*******************************
20509 REM SAUVE LE DESSIN
20510 SYS51200
20518 :
20519 REM SELECTION BANC 0 POUR VIC II
20520 POKE 56578, PEEK(56578) OR 3
20521 POKE 56576, (PEEK(56576) AND 252) OR 3
20528 :
20529 REM SELECTION ECRAN + CARACTERES NORMAUX
20530 POKE VIC+24, 21
20538 :
20539 REM MISE A JOUR ADRESSE ECRAN=1024
20540 POKE 648,4
20548 :
20549 REM PASSAGE EN MODE NORMAL
20550 POKE VIC+17, PEEK(VIC+17) AND 191
20558 :
20559 REM SELECTION COULEUR
20560 POKE VIC+32,7: POKE VIC+33,5
20568 :
20569 REM EXTINCTION DES LUTINS
20570 POKE VIC+21,0
20578 :
20579 REM EFFACE L'ECRAN
20580 PRINT"[CLR][NOIR]";: RETURN
20600 REM*******************************
20601 REM* SOUS-ROUTINE PRINT IMAGE   *
20602 REM*******************************
20610 OPEN4,4:C8$=CHR$(8):E=50176:G=49152:REM ECRAN,GENERATEUR
20620 C0=0:C1=1:C2=2:C7=7:C8=8
20630 FOR I=0 TO 7:P(I)=2^I:NEXT
20640 FOR I=39 TO C0 STEP-1: PRINT#4,C8$;:FOR J=C0 TO 19:FOR K=C0 TO C7
```

```
20650 Q=J*C8+K:M=PEEK(G+(PEEK(E+I+40*J)AND63)*C8+K)
20660 Z(Q)=Z(Q)+M*P(YY)
20670 PRINT#4,CHR$((Z(Q)AND127)+128);
20680 Z(Q)=Z(Q)/128:NEXT:NEXT:PRINT#4
20690 YY=YY+C1:IF YY=C7 THEN YY=C0:GOTO 20710
20700 NEXT:R=C1
20710 PRINT#4,C8$;:FOR L=C0 TO 159:PRINT#4,CHR$((Z(L)AND127)+128);
20720 Z(L)=Z(L)/128:NEXT:PRINT#4
20730 IF R=C0 GOTO 20700
20740 CLOSE4:RETURN
30000 REM*******************************
30001 REM*CREATION DES ROUTINES EN LM *
30002 REM*******************************
30004 REM LECTURE POIGNEES SYS 679
30005 REM RESULTATS PORT1: X=744, Y=746, TIR=749
30010 CS=0: FOR I=679 TO 742: READ A: POKE I,A: CS=CS+A: NEXT I
30020 IF CS<>8274 THEN STOP
30101 :
30102 REM SAUVE L'ECRAN SYS 51200
30103 REM RECUPERE L'ECRAN SYS 51203
30110 CS=0: FOR I=51200 TO 51323: READ A: POKE I,A: CS=CS+A: NEXT I
30120 IF CS<>15304 THEN STOP
30201 :
30202 REM EFFACE L'ECRAN SYS 51324
30210 CS=0: FOR I=51324 TO 51393: READ A: POKE I,A: CS=CS+A: NEXT I
30220 IF CS<>9585 THEN STOP
30999 RETURN
55555 SAVE"@0:SVP",8

READY.
```

SYNCHRO

Démonstration des effets de synchronisation sonore du S.I.D. Les effets de trémolo, xylophone et harmonica sont quelques-unes des possibilités de cet aspect du S.I.D.

```
10 REM SYNCHRO ET TREMOLO
11 :
20 PRINT:PRINT
30 PRINT"NORMAL          -FLUTE      (1)"
40 PRINT"TREMOLO                     (2)"
45 PRINT:PRINT"SYNCHRONISATION":PRINT
50 PRINT"AIGUE           -XYLOPHONE  (3)"
55 PRINT"PROPORTIONELLE-HARMONICA (4)"
60 PRINT:INPUT "(1,2,3,4 OU 0=FIN) ";B
70 IF B=0 THEN END
80 S=54272:GOSUB500:VL=15:AR=16:DE=AR+1
90 IF B>2 THEN AR=AR+2
100 DE=AR+1:AD=8*16+6:SR=12*16+6:POKE S+24,VL
110 RESTORE:POKE S+5,AD:POKE S+6,SR
120 READ N,D:IF N=0THEN GOSUB500:RUN
123 IF B>2 THEN POKE S+15,N
125 POKE S+1,N:POKES+4,DE
130 ON B GOSUB 600,700,800,900
140 POKES+4,AR
150 ON B GOSUB 600,700,800,900
160 GOTO120
170 :
200 DATA25,4,28,4,25,4,25,4,25,2,28,2
210 DATA32,12,25,4,28,4,19,4,19,2,19,2
230 DATA21,1,24,1,25,4,24,2,19,4,0,0
500 FOR T=S TO S+24:POKE T,0:NEXT:RETURN
600 FOR TD=0TO50*D:NEXT:RETURN:REM NORMAL
```

```
699 :
700 REM                                TREMOLO
710 FOR V=0TO D:FORTD=15TO11STEP-2
720 POKES+24,TD:NEXT TD
725 FORTD=11TO15STEP2
730 POKES+24,TD:NEXT TD,V:RETURN
799 :
800 REM                          SYNCHRO AIGUE
810 POKE S+1,N*4:GOTO600
899 :
900 REM              SYNCHRO PROPORTIONELLE
910 POKE S+1,N*1.4:GOTO600
999 :
55555 SAVE"@0:SYNCHRO",8

READY.
```

TERMINAL.

Ce programme transforme le 64 en terminal RS-232. Les caractères programmables sont utilisés pour obtenir un terminal avec un clavier et un écran aussi proches que possible dans leur fonctionnement de ceux d'un vrai terminal. (Par exemple, la barre oblique inverse remplace la livre sterling). Il a été testé avec succès à la vitesse de 300 Bauds sur un MODEM connecté à un gros ordinateur PRIME !

Il nécessite bien entendu un coupleur RS-232 et (accessoirement) un gros ordinateur pour s'y connecter !

```
2 REM TERMINAL CARACTERES ASCII STANDARD POUR CBM 64
3 :
4 :
5 REM BY V.LABAYE & B.MICHEL COPYRIGHT
6 :
7 REM PROTEGE LE SOMMET DE MEMOIRE EN 32768
8 POKE643,0:POKE644,128
9 POKE55,0:POKE56,128:CLR
11 REM RECOPIE ROM CARACTERE EN RAM
15 RAM=32768:ROM=53248+2048:MAX=256*8-1
16 PRINT"[CLR]UN PETIT INSTANT, S.V.P..."
20 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
30 FORI=0TO7:POKERAM+I,PEEK(ROM+I):NEXTI
31 FORI=8TO215:POKERAM+I,PEEK(ROM+I+512):NEXTI
32 FORI=216TO511:POKERAM+I,PEEK(ROM+I):NEXTI
33 FORI=512TO727:POKERAM+I,PEEK(ROM+I-512):NEXTI
34 FORI=728TOMAX:POKERAM+I,PEEK(ROM+I):NEXTI
40 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
60 FORI=1TO7:READ AD:FORJ=0TO7:READA:POKEAD+J,A:NEXTJ,I:REM SPECIAUX
70 POKE56576,197:REM VIC-II DANS LE BANC 2
90 POKE53272,32:REM ECRAN 34816,GEN.CAR 32768
110 REM DIT AU KERNAL OU EST L'ÉCRAN
120 POKE648,(34816/256)
130 PRINTCHR$(147);CHR$(8)"TERMINAL-64  EFFACEMENT ECRAN= [RVS ON]F1[RV
    S OFF]":PRINT" ";
299 :
300 REM TERMINAL RS232
301 OPEN5,2,3,CHR$(38)+CHR$(160)
303 S=54272:POKE S+24,15
304 POKE S+5,15:POKE S+6,249
306 F1$=CHR$(133):BE$=CHR$(7):LO$=CHR$(14)
```

```
307 :
310 POKE212,0:GET#5,A$:GETB$
315 IFA$=BE$GOTO500
320 IFB$=F1$THEN PRINT "[JAUNE L.][CLR]";A$;:A=FRE(0):GOTO310
330 IFB$<>""THEN PRINT#5,B$;
335 IFA$=LO$ GOTO310
340 PRINT "[C.GAUCHE] [C.GAUCHE]";A$;"_";:GOTO310
399 :
500 REM MUSIQUE POUR BELL
510 POKE S+1,44 :POKES+4,17
520 POKES+4,16:FORI=0TO100:NEXT
530 POKE S+1,10:GOTO310
999 :
3000 DATA32992,048,048,024,12,06,003,003,0:REM BACKSLASH
3100 DATA33016,000,000,000,00,00,127,127,0:REM UNDERSCORE
3200 DATA33280,012,012,006,00,00,000,000,0:REM BACK-QUOTE
3300 DATA33496,006,012,012,24,12,012,006,0:REM ACCOLLADE
3400 DATA33504,012,012,012,00,12,012,012,0:REM VERTICAL BAR
3500 DATA33512,024,012,012,06,12,012,024,0:REM ACCOLLADE
3600 DATA33520,059,110,000,00,00,000,000,0:REM TILDA
55555 SAVE"@0:TERMINAL",8

READY.
```

TOUCHE-PROG.

Programmation des touches fonctions : 4 fonctions par touche : normal, shift, logo et CTRL.

Les chaînes de caractères affectées aux touches ne peuvent excéder 8 caractères. La £ remplace le RETURN dans les chaînes. Celles-ci sont accessibles dans les lignes DATA en 1000 à 1030. L'ordre F5, F1, F3, F7 des touches est nécessaire pour simplifier la routine d'interruption qui gère ceci. Cette routine occupe les adresses $CC00 à $CDA0 et peut être relancée par :
SYS 52224 après un STOP-RESTORE ou RESET.

```
10 REM TOUCHES PROGRAMMABLES POUR CBM-64
11 REM
12 REM  4 FONCTIONS PAR TOUCHE:
13 REM
14 REM  NORMAL,SHIFT,LOGO ET CTRL
15 REM
20 BU=52480:FOR I=0TO15:READ A$
30 L=LEN(A$):IF L>10THEN STOP
40 FORJ=0TO7:IF J>=LTHEN P=0:GOTO 60
50 P=ASC(MID$(A$,J+1,1)):IFP=92 THEN P=13
60 POKE BU+J+I*8,P:NEXT J,I
70 :
75 REM ***** CHARGE TOUCHE-PROG/LANG.MACHINE
80 CS=0:FOR I=52224TO52336:READ A:POKEI,A:CS=CS+A:NEXT I
90 IFCS<>14456THEN STOP
100 SYS 52224
990 :
999 REM **** DONNEES POUR TOUCHES PROGRAMMABLES ****
1000 DATA"F5\",,"F3\",,"RU55555\","LIST\" :REM F1 A F7
1010 DATA"F6\",,"F4\",,"F2\",,"F8\",,     :REM F2 A F8
1020 DATA"C=F5\",,"C=F3\",,"C=F1\",,"C=F7\":REM AVEC LOGO
1030 DATA"CTF5\",,"CTF3\",,"CTF1\",,"CTF7\":REM AVEC CTRL
1040 REM   TOUJOURS DANS L'ORDRE F5,F1,F3,F7
1050 REM   LA LIVRE SIGNIFIE ^RETURN^ OU CHR$(13)
```

```
1060 REM   EX : ON A PROGRAMME F7 AVEC LIST
1070 REM   ET F1 AVEC RUN 55555.
1998 :
1999 REM **** PARTIE LANGAGE MACHINE EN $CC00-$CDA0
52224 DATA 120,169,13,141,20,3,169,204
52232 DATA 141,21,3,88,96,72,138,72
52240 DATA 152,72,165,197,197,251,208,8
52248 DATA 104,168,104,170,104,76,49,234
52256 DATA 133,251,173,141,2,201,4,240
52264 DATA 23,201,2,240,14,201,1,240
52272 DATA 5,162,0,76,66,204,162,32
52280 DATA 76,66,204,162,64,76,66,204
52288 DATA 162,96,160,6,132,254,165,197
52296 DATA 197,254,240,13,138,24,105,8
52304 DATA 170,136,192,2,208,238,76,24
52312 DATA 204,138,133,252,169,205,133,253
52320 DATA 160,0,177,252,153,119,2,200
52328 DATA 192,8,208,246,132,198,76,24
52336 DATA 204
55554 END
55555 SAVE"@0:TOUCHE-PROG",8

READY.
```

USR-PEEK ET USR-PEEK SRC

Cette routine en langage machine se substitue à PEEK pour aller lire la mémoire derrière les ROMS KERNAL et BASIC. Ne pas utiliser pour lire le reste de la mémoire. Se charge en mémoire en $C0A0.

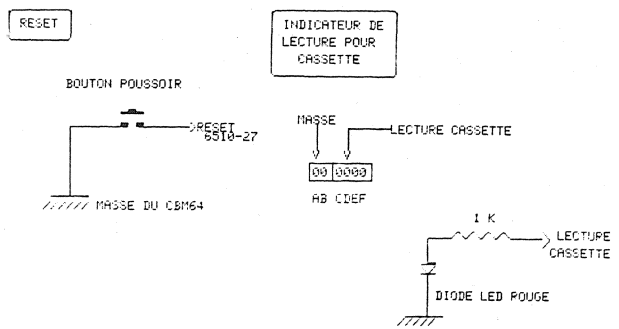
```
30 REM USR-PEEK
40 REM
50 REM REMPLACE PEEK POUR MEMOIRE RAM
60 REM
70 REM ´DERRIERE´ LES ROM .
80 REM
100 FORI=49312TO49344:READA:CS=CS+A:POKEI,A:NEXT:IFCS<>3356THENSTOP
105 POKE785,160:POKE786,192:REM MODIFIE LE VECTEUR D´USR
110 PRINT"[CLR]POKE       PEEK      USR [C.BAS][C.BAS]"
120 FORI=64000TO64200STEP20
130 A=INT(255*RND(0)):POKEI,A
140 PRINTA,PEEK(I),USR(I)
150 NEXT I
160 PRINT:PRINT"ON VOIT QUE PEEK RELIT LA ROM ET QUE
170 PRINT:PRINT"USR RELIT CE QU´ON A ECRIT EN RAM"
49000 :
49312 DATA165,021,072,165,020,072,032,247
49320 DATA183,160,000,120,169,053,133,001
49328 DATA177,020,168,169,055,133,001,088
49336 DATA104,133,020,104,133,021,076,162
49344 DATA179
49345 REM CHECKSUM= 3356 DE 49312 A 49344
55554 END
55555 SAVE"@0:USR-PEEK",8

READY.
```

```
00001  0000                ;ROUTINE USR POUR REMPLACER PEEK
00002  0000                ;        LIT LA RAM 'EN DESSOUS' DES ROM KERNAL ET BASIC
00003  0000                ;
00004  0000                ;IL FAUT INITIALISER LE VECTEUR USR EN 785,786
00005  0000                      * = $C0A0
00006  C0A0  A5 15         USRPIK LDA $15          ;PROTEGE LES VALEURS
00007  C0A2  48                   PHA              ;EN $14 ET $15
00008  C0A3  A5 14                LDA $14
00009  C0A5  48                   PHA              ;EN $14 ET $15
00010  C0A6  20 F7 B7             JSR $B7F7        ;CONVERSION FLOTTANT ->ENTIER EN $14
00011  C0A9  A0 00                LDY #$00
00012  C0AB  78                   SEI
00013  C0AC  A9 35                LDA #53
00014  C0AE  85 01                STA $01          ;BANK ROM HORS SERVICE
00015  C0B0  B1 14                LDA ($14),Y      ;PEEK
00016  C0B2  A8                   TAY              ;SAUVE LA VALEUR LUE
00017  C0B3  A9 37                LDA #55
00018  C0B5  85 01                STA $01          ;BANK ROM EN SERVICE
00019  C0B7  58                   CLI
00020  C0B8  68                   PLA
00021  C0B9  85 14                STA $14
00022  C0BB  68                   PLA
00023  C0BC  85 15                STA $15
00024  C0BE  4C A2 B3             JMP $B3A2        ;CONVERSION ENTIER EN $14 -> FLOTTANT
00025  C0C1                      .END
```

ANNEXE 2

TRUCS POUR ELECTRONICIENS

Ces quelques petites additions au câblage de votre 64 vous faciliteront grandement la tâche. Si toutefois vous ne vous trouvez pas assez compétent pour les réaliser vous même, faites-vous aider car le résultat en vaut la peine.



LE BOUTON DE RESET

Pour provoquer le redémarrage du microprocesseur, il suffit de court-circuiter temporairement le fil RESET à la masse. Un petit bouton poussoir de type 'normalement ouvert' doit être raccordé d'une part à la masse et d'autre part à la broche RESET du 6510.

Cette dernière est la broche numéro 27 du 6510. La masse est entre autre disponible sur le pôle négatif du gros condensateur de 2200 microfarads, sur la broche 1 des 6526 ou encore la broche 21 du 6510. On peut trouver aussi RESET sur la broche 6 et la masse sur la broche 2 du connecteur de bus série IEEE (la prise disquette/imprimante).

L'INDICATEUR DE LECTURE DE LA CASSETTE

Le lecteur de cassette C2N est muni d'une tresse de masse. Commencez par la couper bien à ras. Elle ne sert à rien et représente un risque permanent de court-circuits. Ensuite, connectez entre masse et fil de lecture du câble cassette, une diode électroluminescente LED avec sa résistance de 1 Kohm. Ce petit voyant lumineux vous indiquera les débuts et fins de programmes sur la cassette. C'est très pratique pour le repérage. Les broches du connecteur sont repérées de gauche à droite :
A, B (creux de repérage), C, D, E, F. La masse est en A et le fil de lecture (READ) en D.

NOTICE TECHNIQUE

La composition de ce livre a été réalisée sur un micro-ordinateur CP/M fabriqué au Canada par MEGATEL. Le programme de traitement de textes qui a été utilisé est le bien connu WORDSTAR 3.0 de MICROPRO.

L'impression a été effectuée par une imprimante à marguerite DAISY M-50 équipée d'une roue d'impression "FRENCH PRESTIGE". Le jeu de caractères employé ne contient pas le symbole 'dièse' qui est remplacé dans l'ensemble du texte par le symbole '£'.

Les programmes de l'annexe 1 ont été listés par un COMMODORE 64 interfacé à l'imprimante DAISY au moyen du programme 'LISTING' décrit dans ce livre. Pour les programmes, l'imprimante était équipée d'une roue d'impression "PRESTIGE ELITE".

LE DISQUE DU 64

Conscient de la somme de travail que représente la frappe des programmes de l'annexe 1, BCM a décidé de proposer à ses lecteurs une copie de ceux-ci sur disquette au format 1541.

"Le disque du 64" peut être obtenu contre l'envoi d'un mandat postal (international si nécessaire) d'un montant de 1800 FB à l'ordre de B.C.M. S.C. et muni de la mention "le disque du 64" ainsi que de l'adresse précise de destination.

Adresse :

B.C.M. S.C.
24 ,ROUTE DE LA SAPINIERE
B-4960 BANNEUX
BELGIQUE

TABLE DES MATIERES

Réf. ISBN 2-87111001-8

B.C.M. soc.coop.
24, route de la sapinière
B-4960 LOUVEIGNE
BELGIQUE

D/1984/3827/2

erratas du "livre du 64"

```
P109      ligne 6           2048 devient 1024
P122      ligne 11          2270 devient 53270
P225      ligne 52769       ,80 devient ,91
P226      ligne 00035       $F250 devient $F25B
P256      ligne 340 devient:
          340 IF C>X-1 THEN S=S+20
          supprimer lignes 350,360 et 390
          ligne 440         ESPACE devient RETURN
P277      ligne 20015       ajouter au bout :GOTO 20017
P277      ligne 20215       ajouter au bout :GOTO 20217
P286      ligne 1 du paragraphe 1 : remplacer 27 par 40
```

Parution originale: 1984
Errata : 1987

Editeur:
Editions B.C.M. s.c.

ISBN: 2-87111001-8

Auteur:

Benoit Michel

www.benoitmichel.be

benoit.michel@gmail.com

Comprendre le fonctionnement interne du COMMODORE 64, transformer le clavier en AZERTY accentué et le BASIC en français, utiliser différents modes graphiques en même temps à l'écran, raccorder un bouton de RESET ou une imprimante parallèle, utiliser les 7 modes graphiques en 256 couleurs, programmer les lutins et transformer un programme en langage machine en BASIC...

Découvrez avec ce livre tout ce que vous n'avez jamais rêvé de réaliser sur votre 64.

« LE LIVRE DU 64 »

Convient à tous les possesseurs de CBM 64 ou SX 64 : Tous les sujets sont abondamment commentés et accompagnés de très nombreux programmes d'exemples en BASIC et en langage machine.

Vous y trouverez même plusieurs programmes générateurs d'images dont « S.V.P. », un générateur de dessins en couleurs absolument extraordinaire !!!

BCM s.c.

24, route de la Sapinière - 4960 Banneux Belgique

ISBN 2-87111001-8

DIFFUSÉ PAR P.S.I.