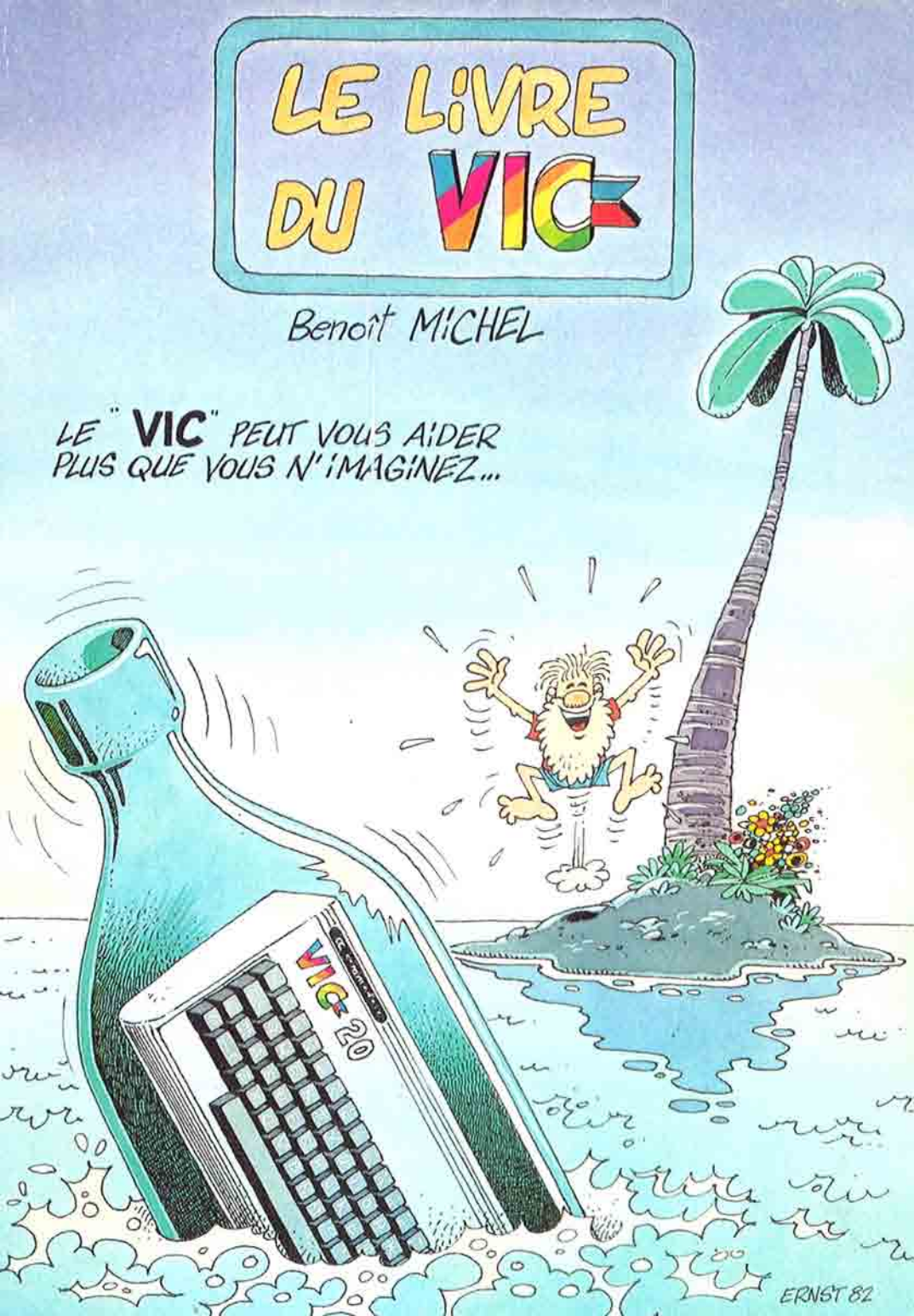
LE LIVRE DU VIC

Benoît MICHEL

LE "VIC" PEUT VOUS AIDER
PLUS QUE VOUS N'IMAGINEZ...



B . C . M .

L E L I V R E D U V I C

par B . MICHEL

INTRODUCTION

Le but de cet ouvrage est de servir de manuel de référence tant au programmeur qu'à l'utilisateur du VIC-20.

Le VIC-20 doit son succès à son prix, certes, mais aussi à ses nombreuses qualités. Le son, la couleur, la qualité du clavier et la fiabilité du système en général et du lecteur de cassette en particulier sont les plus remarquables. Mais dans l'ensemble, il s'agit d'une reconnaissance par la clientèle de la justesse de la politique de CBM en Europe. Les machines bon marché mais extensibles ont toujours eu la cote chez les amateurs. D'autre part, la robustesse des CBM leur ouvre une porte du côté des utilisateurs professionnels, même pour des petits ordinateurs comme le VIC.

Les professionnels ainsi que les amateurs éclairés - un débutant attend en général six mois avant de passer dans la catégorie des amateurs éclairés!- désirent en savoir plus sur leur machine pour en tirer plus. En tout cas, en tirer plus que ne le permettent les différents manuels Commodore. On ne sait trop si c'est la peur de divulguer des secrets de fabrication ou la peur d'effrayer le débutant qui pousse CBM à dissimuler les avantages de ses machines autant sinon plus que leurs inconvénients.

Le **livre du VIC** a donc un double but : Informer et aider. Pour l'information, "les éléments du système" et "les programmes internes du VIC" détaillent le fonctionnement du VIC-20 en tant que système informatique. Pour l'aide, les extensions matérielles et logicielles (extensions mémoires, moniteur langage machine) soulageront le programmeur et l'utilisateur d'un travail souvent fastidieux. De plus, un **INDEX** détaillé ainsi que la **CARTE MEMOIRE** font du LIVRE DU VIC un véritable ouvrage de référence.

Le LIVRE DU VIC a été composé grâce à un programme de traitement de texte fonctionnant sur un micro-ordinateur relié à une imprimante à marguerite. Le jeu de caractères utilisé ne comprenant pas le caractère DIESE, celui-ci est remplacé par le symbole £.

Terminons cette introduction en signalant l'aide apportée à l'écriture de cet ouvrage par Patrick BRUNET pour les programmes BASIC et Serge ERNST pour l'illustration.

CHAPITRE 1

LES ELEMENTS DU SYSTEME

Le VIC est un des micro-ordinateurs les plus en vogue du moment. Son succès est dû à ses nombreuses qualités et à son prix très concurrentiel. Tout ceci n'est possible que grâce à une conception intelligente. Un microsystème comme le VIC ce n'est pas seulement un microprocesseur et un peu d'électronique, c'est aussi les programmes permettant d'utiliser l'ensemble. Ce chapitre va disséquer le fonctionnement du VIC tout d'abord du point de vue électronique et ensuite, pour les périphériques comme le clavier et l'écran, d'un point de vue électronique et logiciel simultanément. En effet, pour nombre d'éléments du système, le matériel et le logiciel sont virtuellement indissociables.

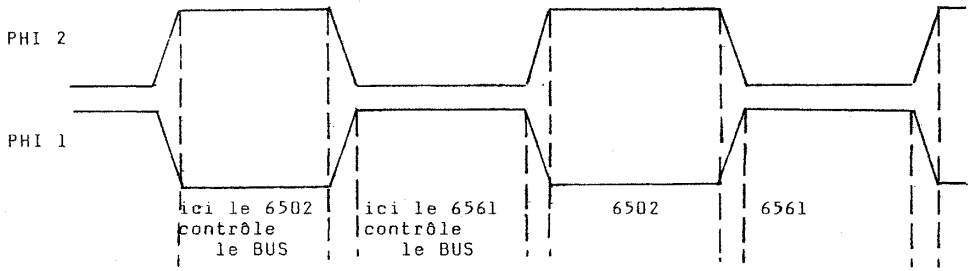
Les 2 processeurs

Comme tous les microsystèmes, le VIC a une architecture générale basée sur le concept de **BUS**. Un BUS est un groupe de fils sur lesquels transitent tous les signaux électriques représentant des informations à l'intérieur du système. Il y a trois groupes de fils répartis en BUS de contrôle, BUS de données et BUS d'adresses. Au départ des BUS se situe le processeur principal, le 6502A. Il s'agit d'un microprocesseur 8 bits, c'est-à-dire que les données sont transférées sur 8 fils : le BUS de données.

Le **6502A** a été conçu par CHUCK PEDDLE (le concepteur du PET et du SIRIUS, entre autres) et réalisé par MOS TECHNOLOGY, une division de COMMODORE. Le 6502 peut transférer des données entre de nombreuses mémoires différentes. Le nombre maximum de mémoires dans le système est de 2^16 soit 65536. Le numéro d'une case mémoire s'appelle **adresse** et est transmis par le 6502 sur 16 fils : le bus d'adresses.

Quant au bus de contrôle, il est constitué de différents signaux dont le principal est le R/W (READ-WRITE) qui détermine le sens du transfert d'informations sur le bus de données : venant du (écriture) ou allant vers (lecture) le microprocesseur. Quel que soit le sens du transfert, c'est toujours le microprocesseur qui décide de l'adresse de la mémoire concernée. Les fils du bus d'adresses conduisent donc les signaux d'adresses dans un seul sens : du microprocesseur vers la mémoire.

Les opérations internes du VIC sont synchronisées par une **HORLOGE** à quartz à deux phases. Cette horloge a une fréquence de 1,108 MHz soit environ 900 nanosecondes par impulsion d'horloge. Les deux phases de l'horloge sont portées par les fils de contrôle **PHI 1** et **PHI 2.**



Le second processeur est un circuit spécialisé en gestion d'images couleurs : le 6561, aussi appelé **V.I.C.** (video interface chip). Le V.I.C. est un microprocesseur spécial, incapable de faire des calculs comme le 6502, mais bien plus performant pour la génération de signaux spéciaux comme les **SYNCHRO** horizontal et vertical, sortie sonore, sortie vidéo composite. Le but principal du 6561 est de générer une image couleur au standard P.A.L. (Il existe un 6560 pour les U.S.A., qui génère une image au standard N.T.S.C). L'image est bien entendu créée à partir des données rangées en mémoire par le 6502. Ici se présente le premier problème : il y a deux processeurs et un seul bus d'adresses et de données. Le problème est résolu par l'horloge à deux phases. Quand le signal PHI 2 vaut +5V (état logique 1), c'est le 6502 qui contrôle les bus d'adresses et de données. Quand le signal PHI 1 vaut +5V, c'est le V.I.C. qui contrôle les bus. Comme les signaux PHI 1 et PHI 2 ne sont jamais simultanément à l'état logique 1, il n'y a pas de conflit.

Le second problème est bien moins simple à comprendre : le 6502 possède 8 fils de données et 16 fils d'adresses . Mais le 6561 possède 12 fils de données et 14 fils d'adresses : on dit que c'est un processeur 12 bits. Le 6561 n'accède donc qu'à 16384 données, soit 4 fois moins que le 6502. Le point le plus délicat vient du fait que pour lire une mémoire de 12 bits, le V.I.C. est bien obligé d'avoir accès à une mémoire et demie (du point de vue du 6502).

Pourquoi cette complexité ? Tout simplement parce qu' une image couleur est complexe : une information sur 8 bits donne 2^8 possibilités soit 256. Or le V.I.C. peut afficher sur l'écran 256 caractères différents et chacun peut avoir 8 couleurs différentes. De plus un caractère peut être encodé de deux manières différentes (mode **HIRES** ou mode **MULTICOLORE**). Il y a donc 256 * 8 * 2 combinaisons différentes, soit 4096 ou 2^12 ! Et voilà pourquoi un caractère doit être codé sur 12 bits.

Quant à la répartition de la mémoire par adresses, il y a divergence de vue la plus complète entre le 6502 et le V.I.C. Les décodeurs d'adresses réalisent cela pour optimiser l'usage de la mémoire par le langage BASIC, et

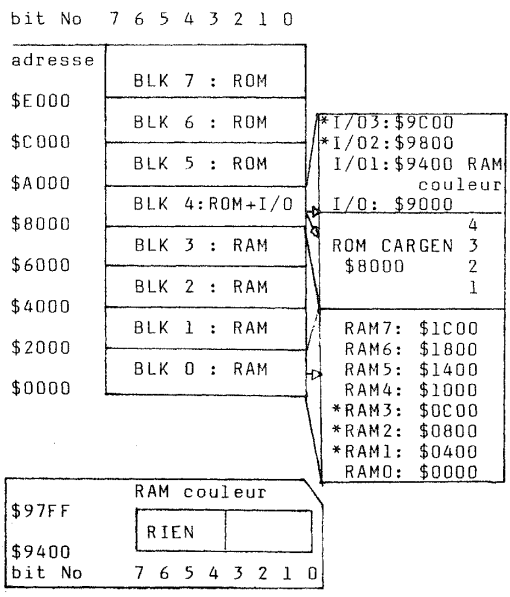
ce, quelle que soit la quantité de mémoire installée dans le VIC.

Les décodages d'adresses du 6561 sont parfois peu aisés à comprendre. Dans le VIC en effet, certaines mémoires sont partiellement décodées, c'est-à-dire que lors du décodage de l'adresse, l'électronique du système ne tient pas compte de la valeur de certains bits d'adresses. Il en résulte que, vue du 6561, la mémoire contient plusieurs exemplaires identiques du même circuit (il s'agit du circuit **RAM COULEUR**).

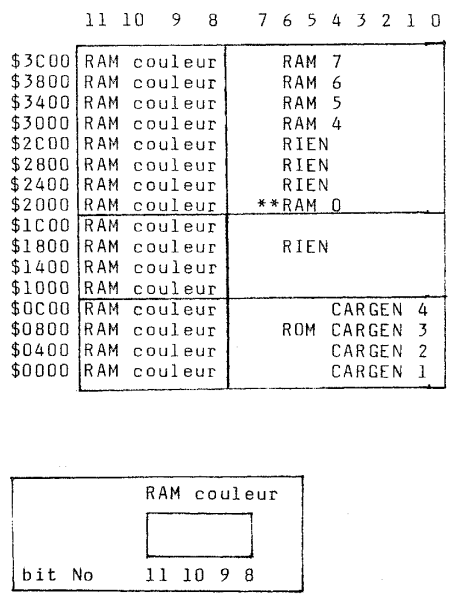
Les adresses générées par le 6561 ont 14 bits de long, mais la RAM couleur, (1 K de mémoire de 4 bits) est sélectionnée par toutes les adresses générées par le 6561. Or, il s'agit d'une mémoire de 1 K et l'espace d'adressage du 6561 est long de 16 K. Le 6561 "voit" donc 16 fois la même mémoire consécutivement. Par exemple, les adresses $0000, $2400, $2800 adressent les 4 mêmes bits 8, 9, 10 et 11 du mot de 12 bits situé à la première adresse de RAM COULEUR.

Dans les cartes-mémoire ci-dessous, les noms des différents blocs de mémoire sont les noms standard utilisés par C.B.M. Cette **CARTE MEMOIRE** indique les correspondances physiques entre les adresses des deux processeurs. Pour connaître le contenu de ces mémoires, reportez-vous à la carte mémoire détaillée au chapitre 5.

Carte-mémoire du VIC vue du 6502



Carte-mémoire du VIC vue du 6561



* : bloc n'existant pas dans la version de base du VIC

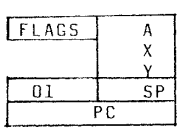
** : le 6561 peut adresser les 1024 premières mémoires du 6502, mais c'est en pratique inutilisable car cette zone contient les variables importantes du système (la pile, par exemple) et on ne peut impunément modifier le contenu de ces mémoires. Seuls les programmes en langage machine peuvent ainsi 'tuer' le BASIC.

Détaillons quelque peu les particularités de l'adressage du 6561 : la RAM couleur est une mémoire de 1 K *4 bits (type 2114) qui est sélectionnée soit quand PHI 2 vaut 1 et que l'adresse est comprise entre $9400 et $97FF : c'est un accès du 6502, soit quand PHI 1 vaut 1, quelle que soit l'adresse : il s'agit de tous les accès à la mémoire du 6561. Pour les blocs RAM 1, RAM 2, RAM 3 qui constituent l'extension mémoire RAM externe de 3K, la sélection ne s'effectue que si le signal PHI 2 vaut 1, c'est-à-dire lors des accès à la mémoire du 6502. Il est donc impossible au 6561 d'accéder à la mémoire de 3 K externe.

Le 6502.

Le microprocesseur 6502 est un microprocesseur 8 bits : il ne travaille que sur des données de 8 bits, c'est-à-dire des données comprises entre 0 et 255, transmises sur 8 fils simultanément. Le 6502 répond à 56 instructions différentes, possède 13 modes d'adressage et 6 registres internes.

Les registres internes du 6502 comprennent :
Le **registre A** ou accumulateur, où se trouve le résultat des opérations arithmétiques.
Les **registres X et Y** qui sont utilisés pour les modes d'adressage complexes ainsi que comme stockage provisoire pour des données.



Le **registre SP** ou pointeur de pile est un registre à 16 bits dont les 8 bits de poids fort sont fixes et valent $01. Le pointeur de pile ne peut donc contenir que des valeurs comprises entre $0100 et $01FF. Le pointeur de pile est automatiquement modifié par les instructions PHA, PLA, PHP, PLP, RTS, RTI qui mettent et enlèvent des valeurs dans la pile. Il ne peut être modifié explicitement que par l'instruction TXS, qui tranfère X dans SP.
Le **registre PC** contient en permanence l'adresse de la prochaine instruction à effectuer. On le modifie en général par JMP, JSR et RTS (équivalents en langage machine de GOTO, GOSUB et RETURN).

Les modes d'adressages sont les différentes manières de donner l'adresse d'un opérande. Le 6502 possède 11 modes différents, ce qui en fait un processeur très efficace.

1. Mode implicite : l'opérande est toujours connu, spécifié dans l'octet instruction (**OPCODE**) lui-même. Exemple : TAX transfère le contenu du registre A dans le registre X.

2. Mode impliqué : L'opérande est retiré de la pile. Par exemple, RTS transfère le contenu des deux premiers octets de la pile dans le registre PC.

3. Mode immédiat : L'opérande est l'octet qui suit l'OPCODE dans le programme. Dans le texte des programmes, le caractère £ signifie : **MODE IMMEDIAT.**

Exemple : LDA £$02 met la valeur $02 dans le registre A (£ représente le dièse).

4.Mode direct page 0 : L'adresse de l'opérande, qui est en page 0, est spécifiée sur un octet qui suit l'OPCODE. Exemple : LDA $02 prend la valeur contenue dans la mémoire à l'adresse $02 et met cette valeur dans le registre A.

5. Mode direct étendu : L'adresse de l'opérande est quelconque et spécifiée sur 2 octets qui suivent l'OPCODE (partie basse puis partie haute). Exemple: saut à l'adresse $1000 : JMP $1000 est codé comme suit :

```
$4C (OPCODE JMP)
$00 (partie basse de l'adresse)
$10 (partie haute de l'adresse)
```

6. Mode indexé page 0 : L'adresse de l'opérande se fait en deux parties : l'OPCODE est suivi d'un octet, c'est l'adresse de base. On ajoute à cette adresse le contenu du registre X ou du registre Y. C'est l'adresse de l'opérande : le 2me octet est l'adresse d'une table en page 0, le registre X ou Y, le numéro de l'octet dans la table.

```
Exemple : LDA $02,X     Si X contient $05,
          la valeur située en $02+$05 = $07 sera transférée dans le registre A.
```

7. Mode indexé étendu : L'adresse de l'opérande, comme ci-dessus, est spécifiée dans un registre X ou Y plus l'adresse de la table sur deux octets.

```
Exemple :       LDA $0400,X   est codé :
                $BD OPCODE LDA ,X
                $00 partie basse de l'adressse de base
                $04 partie haute de l'adresse de base
```

Si le registre X contient $04, la valeur située en $0400 + $04 = $0404 sera transférée dans le registre A.

8. Mode indirect étendu : Ce mode n'existe que pour une seule instruction : JMP. L'adresse spécifiée sur deux octets juste après l'OPCODE est l'adresse où se trouve l'opérande.

Exemple : si $0300 contient $04 partie basse de l'adresse indirecte
si $0301 contient $10 partie haute de l'adresse indirecte
si $1004 contient $01 partie basse de l'opérande
si $1005 contient $00 partie haute de l'opérande ,
l'instruction JMP ($0300) codée $6C, $00, $03 prend la valeur en $0300 et $0301 et la considère comme l'adresse de l'opérande. L'opérande est saisi en $1004 et $1005, et le JMP charge la valeur $0001 dans le registre PC, ce qui effectue un saut à cette adresse.
Attention: Si la partie haute de l'adresse indirecte est $XX et la partie basse de l'adresse indirecte est $FF, le 6502 prendra les 2 octets de l'opérande en $XXFF et en $XX00 qui ne sont pas consécutives.En effet avec cette instruction, l'opérande ne peut être à cheval sur la frontière entre 2 pages.

9. **Mode indirect préindexé page 0** : L'instruction LDA($04,X) par exemple a un opérande trouvé en trois temps : l'octet qui suit l'OPCODE est l'adresse de début d'une table en page 0. On y ajoute le contenu du registre X (jamais le registre Y) et on obtient ainsi une adresse. Cette adresse et la suivante contiennent l'adresse de l'opérande.
Soit LDA ($04,X) codée $A1, $04
avec, dans le registre X la valeur $05, et
en $0004 + $05 = $0009 la valeur $02 (partie basse adr.indirecte)
en $0004+$05+1= $000A la valeur $10 (partie haute adresse indir.)
en $1002 la valeur $47,
la valeur $47 est transférée dans le registre A.
2ème octet = adresse d'une table d'adresses

10. **Mode indirect post-indexé page 0** : Par rapport au mode 9, on constate ici que l'indexation s'effectue après l'indirection. L'instruction LDA ($04),X a également un opérande trouvé en trois temps. L'octet qui suit l'OPCODE est l'adresse en page 0 de 2 octets qui contiennent l'adresse de base. A cette adresse on ajoute la valeur du registre Y (on ne peut employer X) pour obtenir l'adresse de l'opérande.

Exemple : LDA($04),Y est codé $B1, $04
avec en $0004 la valeur $00 : partie basse de l'adresse de base
en $0005 la valeur $C0 : partie haute de l'adresse de base
dans le registre Y, la valeur $10.
en $C000 + $10 = $C010, la valeur $53,
la valeur $53 est transférée dans le registre A.

11. **Mode relatif** : Utilisé uniquement avec les instructions de branchement conditionnel. L'octet qui suit l'OPCODE représente un déplacement par rapport à la valeur actuelle du registre PC qui contient l'adresse du prochain OPCODE.
Exemple : BNE $04 additionne au registre PC la valeur $04 et stocke le résultat dans le registre PC. Ceci ne s'effectue bien entendu que si la condition est remplie.
Dans l'exemple ci-dessus : BNE $04 signifie brancher en PC+$04 si le drapeau Z vaut 0, sinon continuer le programme sans modifier le registre PC.
Les déplacements sont spécifiés en avant ou en arrière à l'aide du bit 7. $FF = déplacement de -128 octets, $00 = pas de déplacement, $7F = déplacement de +127 octets.

LE JEU D'INSTRUCTIONS DU 6502

OPCODE	Mode d'adressage											Drapeaux affectés							
	1	2	3	4	5	6	7	8	9	10	11	S	V	.	B	D	I	Z	C
LDA			X	X	X	X	X		X	X		X						X	
STA				X	X	X	X		X	X									
ADC			X	X	X	X	X		X	X		X	X					X	X
SBC			X	X	X	X	X		X	X		X	X					X	X
CMP			X	X	X	X	X		X	X		X						X	X
AND			X	X	X	X	X		X	X		X						X	
ORA			X	X	X	X	X		X	X		X						X	
EOR			X	X	X	X	X		X	X		X						X	
LDX			X	X	X	X	X					X						X	
LDY			X	X	X	X	X					X						X	
STX				X	X	X													
STY				X	X	X													
INC				X	X	X	X					X						X	
DEC				X	X	X	X					X						X	
ROL	X			X	X	X	X					X						X	X
ROR	X			X	X	X	X					X						X	X
ASL	X			X	X	X	X					X						X	X
LSR	X			X	X	X	X					0						X	X
CPX			X	X	X							X						X	X
CPY			X	X	X							X						X	X
BIT			X	X								7	6					X		(*)
TAX	X											X						X	
TXA	X											X						X	
TAY	X											X						X	
TYA	X											X						X	
TSX	X											X						X	
TXS	X																		
INX	X											X						X	
INY	X											X						X	
DEX	X											X						X	
DEY	X											X						X	
CLI	X																0		
SEI	X																1		
CLC	X																		0
SEC	X																		1
CLD	X															0			
SED	X															1			
CLV	X												0						
NOP	X																		
JMP					X				X										
JSR					X														
RTS		X																	
RTI		X										X	X		X	X	X	X	X
BRK		X													X		1		
PHA		X																	

(*) avec BIT, les bits 7 et 6 de l'opérande sont copiés dans S et V.
(**)le drapeau B mis à 1, mais en plus tout le registre d'état est sauvé dans la pile après le registre PC.

OPCODE	Mode d'adressage											Drapeaux affectés							
	1	2	3	4	5	6	7	8	9	10	11	S	V	.	B	D	I	Z	C
PLA		X										X						X	
PHP		X										X	X		X	X	X	X	X
PLP		X																	
BCC											X								
BCS											X								
BEQ											X								
BNE											X								
BMI											X								
BPL											X								
BVC											X								
BVS											X								

Le registre d'état (DRAPEAUX)

S	V	.	B	D	I	Z	C

Drapeau est la traduction du mot anglais **flag** . Le registre d'état du 6502 est constitué de 7 bits représentant 7 drapeaux (si le bit = 1, le drapeau est levé, sinon il est abaissé). Chacun de ces drapeaux représente un état particulier du 6502.

Le bit C vaut 1 si la dernière instruction modifiant le registre A conduit à y mettre une valeur supérieure à $FF (débordement).
Appelé aussi **CARRY FLAG.**

Le bit Z vaut 1 si la dernière instruction modifiant le registre A, X ou Y y a amené une valeur nulle.
Appelé aussi **ZERO FLAG.**

Le bit I indique qu'une interruption est interdite. Ce bit est mis à 1 par l'instruction SEI et remis à 0 par CLI.
Appelé aussi **INTERRUPT FLAG.**

Le bit D indique que l'ont travaille en mode décimal et non en binaire, n'est jamais utilisé dans le VIC, ne pas s'en soucier.
Appelé aussi **DECIMAL FLAG.**

Le bit B indique que l'instruction BRK (break) a été exécutée, c'est à dire que le 6502 exécute un programme d'interruption déclenché par logiciel et non par matériel.
Appelé aussi **BREAK FLAG.**

Le bit . ne sert à rien.

Le bit V indique qu'un signal a été reçu sur la broche No 38 du 6502. Peut être remis à 0 par CLV. (Ceci n'est pas utilisé dans le VIC). Il est aussi mis à 1 par les instructions ADC et SBC en cas de débordement.

Est également modifié comme CARRY par les additions et les soustractions.
Appelé aussi **OVERFLOW FLAG.**

Le bit S indique le signe de la valeur affectée à un des registres A, X ou Y.
Appelé aussi **SIGN FLAG.**

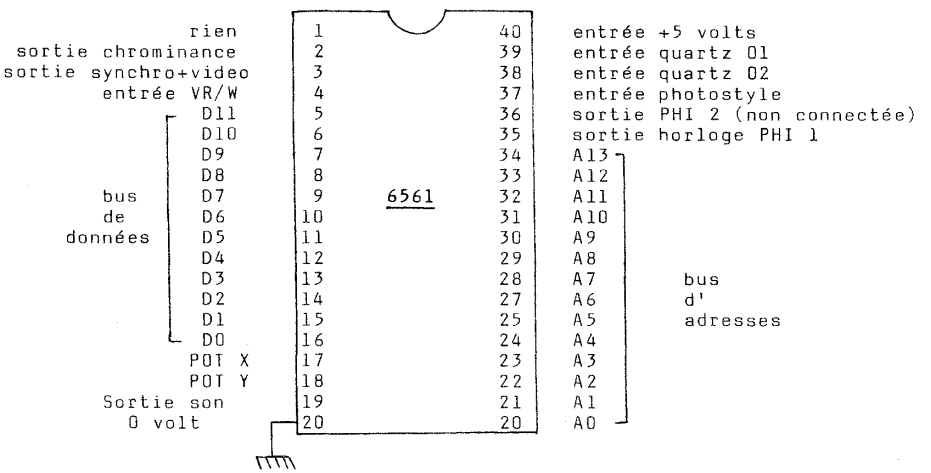
Pour plus de détails sur la programmation du 6502, nous renvoyons le lecteur à l'ouvrage "La programmation du 6502" édité par SYBEX. Voir également la bibliographie en fin de volume.

Comme il n'existe pas de moyen simple d'écrire en langage machine sur le VIC de base, nous ne pouvons que vous conseiller l'emploi du programme - **MONITEUR 6502** que l'on trouvera au chapitre **Extensions logicielles.** Avant de se lancer dans la programmation en langage machine, il est préférable d'étudier le fonctionnement du système VIC dans son ensemble. On verra entre autres que les adresses "sûres" pour ranger des programmes en langage machine sont assez nombreuses :

Adresses sûres :

	taille
- en page 0, les adresses $FB à $FF	5
- les adresses $02A1 à $02FF	94
- le tampon cassette : à ne pas utiliser si le programme nécessite l'usage de la cassette. $033C à $03FF	195
- à l'intérieur d'une ligne REM**.... en BASIC. L'adresse peut être déterminée avec le programme SUPERLIST. Le programme en langage machine ne peut pas contenir de $00.	x
- entre sommet de BASIC et sommet de mémoire. réserver la zone en abaissant le sommet de BASIC par POKE aux adresses entre $35 et $36. Exemple : POKE 54, PEEK (54)-1 :CLR réserve 256 octets. Ne pas oublier d'effectuer le CLR pour ajuster les autres pointeurs.	x
- Dans n'importe quelle mémoire RAM non contiguë avec la zone BASIC. Exemple : une cartouche de 8 K RAM dans le bloc 2, 3 ou 5. Avantage de la zone bloc 5 (voir **AUTOSTART**): Redémarrage par **RESET** (voir extensions matérielles). Les adresses I/O bloc 2 et I/O bloc 3 (2 blocs de 1 K) sont disponibles pour extensions logicielles ou périphériques.	x

Le 6561



Le ciruit 6561 représente le lien le plus complexe entre le programme et le monde extérieur : écran couleur, caractères programmables, haute résolution graphique, générateurs sonores, photostyle, entrées analogiques.

Si le cerveau est le 6502, le 6561 représente une bonne part de ce système complexe qu'est le VIC.

Le dialogue entre le 6561 et le programmeur s'établit par l'intermédiaire de 16 registres adressables par le 6502. Ces registres sont situés en $9000 ou 36864. Nous y ferons référence régulièrement par leurs noms officiels : VIC-0 à VIC-F.

Un résumé de l'usage des 16 registres du 6561 se trouve en fin de volume dans la carte-mémoire. Nous les étudierons au fur et à mesure de leur usage dans les différentes rubriques ci-dessous.

Préparation de l'image : Les principales caractéristiques de l'image sont modifiables par programmation du 6561.

Entrelacement : Le balayage TV est entrelacé ou non : la TV affiche d'abord les lignes paires puis ensuite les lignes impaires en entrelacé. En non entre-

lacé, les 625 lignes de l'image sont affichées à l'écran à chaque image.
Mode entrelacé : VIC 0, BIT7 = 1 (35864 ou $9000)
Pour passer en mode entrelacé : POKE 36864,PEEK (36864) OR 128
Le mode entrelacé doit servir si l'image créée par le VIC doit être mixée avec une image vidéo normale. En effet, c'est ainsi que travaillent les images TV habituelles.
Normalement : non entrelacé.

Marge gauche : la zone utile de l'écran peut se positionner n'importe où par rapport au bord gauche de l'écran. Ceci est surtout utile si on modifie la largeur de l'écran (voir plus bas) : on peut alors recadrer correctement l'écran.
Marge gauche : VIC-0 BITS 0 à 6 (36864 ou $9000)
Gamme pratique de valeurs : 0-40

```
Exemple : 10 FORI=0 TO 40 : POKE 36864,I
          20 FORJ=0 TO 500 : NEXTJ
          30 NEXTI : POKE 36864,12
Normalement : VIC-0 =12
```

Marge supérieure : De même, la marge supérieure est modifiable.
VIC-1 (35865 ou $9001)
Gamme pratique des valeurs : 0 - 150

```
Exemple : 10 FORI=0 TO 150 :POKE 36865,I
          20 FORJ=0 TO 100 : NEXT J
          30 NEXTI : POKE 36865,38
Normalement : VIC-1 = 38
```

Les variations importantes de marge sont très spectaculaires. Il faut les employer par exemple pour écrire des titres, des messages, des modes d'emploi pour jeux etc. On décale l'écran vers la droite, puis lorsqu'il a disparu, on y écrit le texte, on change éventuellement la couleur du fond, puis on ramène lentement le texte par la gauche. Les effets de ce genre sont toujours très appréciés.

Largeur écran : Le nombre de caractères d'une ligne écran est modifiable. BASIC choisit 22 caractères de large car 22*23 est la taille maximale de l'écran qui autorise l'accès à toute l'image en mode "haute-résolution". La largeur de 22 est la seule qui soit 'digérée' par BASIC. Pour toutes les autres valeurs, à vous de gérer les déplacements du curseur en conséquence.
Largeur écran : VIC-2, BITS 0 à 6 (36866 ou $9002)
Gamme pratique de valeurs : 0-32 (en changeant la marge à gauche)

```
Exemple : 10 A=PEEK(36866)AND128
          20 FORI= 0 TO 32 : POKE 36866, I OR A
          30 FORJ= 0 TO 300 : NEXTJ
          40 NEXTI : POKE 36866,22 OR A

Normalement : 22 ou 150 (suivant valeur du bit 7)
              150    : VIC de base ou +3K
              22     : VIC +8K
```

Hauteur écran : Le nombre de lignes de l'écran est modifiable également.
Hauteur écran : VIC-3, BITS 1 à 6 (36867 ou $9003)
Gamme pratique de valeurs : 0 - 28 (en changeant la marge supérieure)

```
Exemple : 10 A=PEEK(36867)AND129
          20 FORI=0 TO 28
          30 POKE36867,A OR (2*I)
          40 FORJ= 0 TO 500 :NEXTJ
          50 NEXTI :POKE 36867, A OR (2*23)
```

Adresse de l'écran : Normalement l'écran est en $1E00 (7680) dans le VIC de base ou avec extension 3K. Avec 8K ou plus d'extension RAM, BASIC remet l'écran en $1000 (4096). On en verra la raison à la rubrique "haute résolution".
Pour spécifier l'adresse de l'écran au 6561, on doit bien entendu lui donner l'adresse 'vue du 6561'et non vue du 6502. La correspondance entre les 2 adresses se trouve ci-dessus (voir les deux processeurs). Pour mettre l'écran en $1000 pour le 6502, on doit donner au 6561 l'adresse $3000. On voit que la conversion est simple : pour avoir l'adresse 6561, prendre l'adresse 6502 et y ajouter $2000 (8192).
L'adresse de l'écran est donc une adresse sur 14 bits, mais seuls les bits 9 à 13 sont modifiables, c'est-à-dire que l'adresse d'écran ne peut varier que par pas de 512 ($0200).

```
Adresse écran :    bit 9          = VIC-2,bit 7 (36866)
                   bits 10-13     = VIC-5,bits 4-7(36869)
```

Les adresses écran possibles sont reprises ci-dessous (les seules qui existent dans la carte mémoire vue du 6561).

NOM DE LA MEMOIRE	ADRESSE ECRAN POUR LE 6561	VAL1	VAL2	ADRESSE ECRAN POUR LE 6502	No PAGE ECRAN PR LE 6502	ADRESSE RAM COULEUR POUR LE 6502	
RAM 0	$2000	128	0	$0000	0	$9400	(1)
RAM 0	$2200	128	128	$0200	2	$9600	(2)
RAM 4	$3000	192	0	$1000	16	$9400	(3)(4)
RAM 4	$3200	192	128	$1200	18	$9600	(5)
RAM 5	$3400	208	0	$1400	20	$9400	(5)
RAM 5	$3600	208	128	$1600	22	$9600	(5)
RAM 6	$3800	224	0	$1800	24	$9400	(5)
RAM 6	$3A00	224	128	$1A00	26	$9600	(5)
RAM 7	$3C00	240	0	$1C00	28	$9400	(5)
RAM 7	$3E00	240	128	$1E00	30	$9600	(6)

(1) interdit : page 0 du 6502 !
(2) écran pour haute résolution en langage machine uniquement.
(3) écran ici si VIC > 8K.

(4) Pour vic de base : ici début du texte BASIC !
(5) autre écran possible pour toutes configurations.
(6) écran VIC de base ou +3K.

Pour modifier l'adresse de l'écran, il y a deux méthodes. La première qui est la plus simple, réinitialise l'écran : fond blanc, bord bleu, écran effacé, curseur HOME. Il suffit de mettre la partie haute de l'adresse de l'écran (adresse pour le 6502) en 648 et d'appeler la routine SETSCR en $E518.
Exemple : mettre l'écran en $1000 ou 4096, nécessite le calcul de la partie haute de l'adresse écran : $10 ou 16 (=4096/256).

```
POKE 648,16:SYS 58648
```

La deuxième méthode consiste à modifier directement les 5 bits concernés du V.I.C. (voir programme **écrans multiples** en annexe). Notez que la table des adresses de lignes-écran en 217 ($00D9) doit être remise à jour pour que BASIC fonctionne correctement.
On peut cependant changer l'adresse de l'écran seule si on n'utilise aucun PRINT. On utilisera :

```
POKE 36869,PEEK(36869)AND 15 OR VAL1
POKE 36866,PEEK(36866)AND 127 OR VAL2
```

(VAL1 etVAL2 sont donnés au tableau de la page précédente)

De plus, il faut noter que la **RAM couleur** n'a pas une adresse indépendante de celle de l'écran, vue du 6561. Comme cette mémoire de 1K se recopie dans tout l'espace d'adressage du 6561, il est donc normal que, si l'écran occupe la première moitié d'une page de 1K, la première moitié de la RAM couleur soit est utilisée. C'est le contraire dans l'autre cas. On voit donc au tableau que si l'adresse de l'écran est divisible par 1024, la RAM couleur est en $9400, ce qui est le cas si un VIC a plus de 8K et que l'écran est donc en $1000 pour le VIC. Si l'adresse de l'écran n'est pas divisible par 1024, la RAM couleur est en $9600, ce qui est le cas dans le VIC de base avec l'écran en $1E00.
($9400 = 37888, $9600 = 38400)

Taille des caractères : Il est possible d'avoir des caractères de taille normale (8*8 points) ou de taille double (8*16 points). En conséquence, l'écran du VIC qui contient normalement 506 caractères, n'en contient plus que 253 en doubles caractères. Or il se trouve que le 6561 ne peut afficher simultanément plus de 255 caractères différents. L'intérêt des caractères doubles est donc que l'écran peut être rempli de caractères tous différents.Comme ils sont **tous** programmables, on obtient la haute résolution graphique où chaque point de l'écran est modifiable indépendamment. A part cela, les gros caractères ont leur avantage. Sans modifier la taille de l'écran on a évidemment deux fois moins de lignes : seule la moitié supérieure de l'écran est visible. On peut agrandir légèrement la partie visible en relevant la marge supérieure et en augmentant la hauteur de l'écran. (Voir en annexe le programme **Double Alpha**).
Taille caractères : VIC-3 BITS 0 (36867 ou $9003)
Gamme de valeurs : 0 - 1 0=simples, 1=doubles

```
Exemple :       10   A=PEEK(36867)AND254
                20   POKE 36867,A OR 1 : REM DOUBLES
                30   POKE 36867,A      : REM SIMPLES
Normalement : 0
```

Position balayage écran : 9 bits du 6561 contiennent l'actuel numéro de ligne en cours d'affichage à l'écran. Il ne s'agit pas ici de lignes de caractères mais de lignes de points. La valeur peut varier de 0 à 512 bien qu'une image réelle possède 625 lignes. Ces 9 bits ne sont d'aucune utilité pour le programmeur en BASIC. Mais en langage machine, les choses sont bien différentes. La durée d'une ligne de balayage de l'écran est de 64 microsecondes. Le BASIC n'arrive donc pas à suivre le phénomène. En langage machine, il est cependant possible d'effectuer plusieurs instructions sur la durée d'une seule ligne. Citons quelques exemples pour illustrer le procédé :
Un programme fonctionne normalement puis est interrompu par le déclenchement d'une des 2 minuteries (voir les 6522) qu'il a programmées. A ce moment, le programme attend l'arrivée d'une ligne précise puis modifie le contenu de certains registres du 6561 ou des 6522 et redéclenche ensuite la minuterie. En effectuant ceci deux fois par image, il est donc possible d'afficher à chaque balayage complet de l'écran deux images 'haute résolution' successives : l'une occupe la moitié supérieure de l'écran, l'autre la moitié inférieure.
Il est bien entendu possible d'avoir plus de deux images et chacune de celle-ci a pu être programmée indépendamment dans des **modes différents**. Par exemple, il est ainsi possible d'obtenir deux images entourées et séparées par la "couleur de fond", chacune possédant un jeu de caractères différents, des couleurs différentes, etc. Ce domaine est encore peu exploré mais semble receler un nombre de possibilités énormes. Affaire à suivre !

BIT0 position balayage = VIC-2 BIT7 (36866)
BITS 1-8 position balayage = VIC-4 (36868)

Adresse du générateur de caractères : Le générateur de caractères est la zone de mémoire qui contient le dessin de chacun des points de chacun des 256 caractères. Suivant la taille des caractères, simple ou double, le générateur de caractères occupe 2 ou 4 K octets en mémoire. D'origine, le générateur de caractères occupe 2 K en $8000 (32768) pour le 6502. La combinaison des touches LOGO et SHIFT modifie l'adresse du générateur pour le mettre en $8800 (34816). En modifiant l'adresse du générateur, on peut le mettre en partie ou en totalité en mémoire RAM, ce qui permet de modifier le dessin des caractères. Voir **Caractères programmables**. Il y a 16 adresses possibles pour le générateur.

Point important : Regardons l'espace d'adressage du 6561, qui couvre 16 K mots de 12 bits (dont seuls 8 sont utiles ici). Si le générateur de caractères commmence 2K en dessous du sommet (adresse $3800 pour le 6561) et que la longueur du générateur est de 4K (caractères doubles), les deux premiers Kilo-mots de mémoire seront en RAM et les deux derniers Kilo-mots seront en ROM à partir de l'adresse 0 (pour le 6561). En effet, le compteur d'adresses compte jusque $3FFF puis passe à $4000, qui est une adresse de 15 bits de long. Le 15me bit est ignoré car le matériel ne prévoit pas son existence. Le registre interne d'adresses du 6561 contient donc $0000. Au tableau ci-dessous, les configurations inutilisables ne sont pas reprises.

configuration no	taille caractères	nom des mémoires	adresse du générateur pour le 6561	adresse du générateur pour le 6502 hexa.	adresse du générateur pour le 6502 déc.	taille utile du générateur	nbre carac. en ROM	nbre car. en RAM	valeur
1	8	CARGEN1,2	$0000	$8000	32768	2K	256	0	0
2	8	CARGEN2,3	$0400	$8400	33792	2K	256	0	1
3	8	CARGEN3,4	$0800	$8800	34816	2K	256	0	2
4	8	CARGEN4, rien	$0C00	$8C00	35840	1K	128	0	3
5	8	RAM4,5	$3000	$1000	4096	2K	0	256	12
6	16	RAM4,5,6,7	$3000	$1000	4096	4K	0	256	12
7	8	RAM5,6	$3400	$1400	5120	2K	0	256	13
8	16	RAM5,6,7, CARGEN1	$3400	$1400 $8000	5120 32768	3K 0	(*)	192	13
9	8	RAM6,7	$3800	$1800	6144	2K	0	256/ 192	14
10	16	RAM6,7 + CARGEN1,2	$3800	$1800 $8000	6144 32768	2K 0	0 (*)	128	14
11	8	RAM7,CAR-GEN1	$3C00	$1C00	7168	1K	128	128/ 64	15
12	16	RAM7,CAR-GEN1,2,3	$3C00	$1C00 $8000	7168 32768	1K 0	0 (*)	64	15

(*) = inutilisables

Pour choisir une configuration, il faut modifier la taille des caractères si nécessaire et exécuter les deux lignes suivantes :

```
10 A=PEEK(36869) AND 240
20 POKE 36869,A OR(VALEUR)
```

(La valeur est lue dans la dernière colonne du tableau ci-dessus)

Configuration no 1 : Majuscules plus graphiques et inverses : caractères normaux à l'allumage du VIC.

Configuration no 2 : Majuscules et graphiques inversés + majuscules et minuscules : c'est utilisable mais peu utile. Seul intérêt : certains graphiques sont disponibles en plus des minuscules.

Configuration no 3 : Majuscules + minuscules et inverses : obtenus en appuyant sur SHIFT et LOGO (ou PRINT CHR$(14)).

Configuration no 4 : Majuscules + minuscules inversées seules + caractères aléatoires correspondant à la zone de l'espace d'adressage du 6561 qui ne contient aucune mémoire. Peu utile.

Configuration no 5 : 256 caractères programmables en RAM, format simple. En pratique, configuration employable avec extension RAM 3K. L'écran étant en $1E00, le générateur de caractères en $1000 à $17FF, il reste les adresses $0400 à $0FFF pour le programme BASIC, soit 3K.

Configuration no 6 : La seule où le générateur de caractères est complètement programmable. On a donc accès à tous les points de tous les caractères. Mais, inconvénient majeur, seule la RAM 0 reste utilisable pour la mémoire écran que l'on est donc obligé de définir en $0200 (512) car $0000 est interdit. Or le BASIC utilise abondamment les pages 2 et 3 de la RAM 0. On ne peut donc pas utiliser cette configuration en BASIC. Si un programme langage machine utilise cette configuration, il doit sauver les 256 octets de la page 2 (512 à 767) dans une zone sûre et les restaurer avant de repasser au BASIC.

Configuration no 7 : 256 caractères simples sont programmables, donc 1/2 écran. Utilisable par un VIC de base ou +3K : faire POKE 36869,253. Le générateur de caractères occupe RAM5 et RAM6, l'écran est en $1E00 dans RAM7. Il reste donc la mémoire RAM4 pour le BASIC. 1K de programme, c'est petit mais utilisable.

Configuration no 8 : La plus utilisée pour faire de la haute résolution graphique. Seuls 192 caractères sont programmables mais en 8*16 points chacun, cela fait tout de même 24576 points, soit en pratique 24 caractères par ligne et 8 lignes mais on peut aussi programmer l'écran en 22*8 ou 16*12 ce qui donne un grand rectangle vertical. Le générateur de caractères occupe les RAM 5, 6 et 7 : on peut donc positionner l'écran dans la RAM 4 (en $1000 par exemple). Si on a plus de 8K d'extension, le BASIC a déjà mis l'écran en $1000. Si on n'a que 3K d'extension on placera plutôt l'écran en $1200 pour laisser le plus de place possible au BASIC : de $0400 à $11FF soit 3,5K.Pour la version de base, il ne resterait que 500 octets pour le programme : c'est trop peu pour être utilisable. (Voir le programme HIRES en annexe : il reste 11 octets après exécution du programme).

Configuration no 9-1: Utilisable pour toutes les configurations VIC. Si les extensions mémoire dépassent 8K, l'écran sera en $1000 et le BASIC à partir de $2000. Dans un VIC de base ou +3K, il faut établir le sommet de mémoire en $1600 et installer l'écran en $1600, ce qui laisse $600 = 1536 octets pour le BASIC en version de base ou 4608 octets avec 3K d'extension. C'est la plus intéressante configuration qui programme 256 caractères simples (dans un VIC de base, elle laisse plus de place au BASIC que la configuration no 7). On a

accès à la moitié de l'écran en haute résolution.

Configuration no 9-2: Identique à 9-1 mais en laissant l'écran à sa place en $1E00. On ne pourra utiliser les 64 derniers caractères : il en restera donc 192 disponibles.

Configuration no 11-1: Seulement 128 caractères programmables. Pour le VIC de base ou avec 3K d'extension, nécessite le repositionnement du sommet de BASIC et de l'écran en $1A00 (6656), ce qui laisse au BASIC 2560 ou 5632 octets. Pour les mémoires de plus de 8K, l'écran peut rester en $1000.
Configuration **très intéressante :**
ECRAN en $1A00
CARGEN en $1C00
Car on a 128 caractères programmables et les 128 caractères normaux du VIC (qui remplacent les inverses) : on peut donc mêler harmonieusement textes, chiffres et graphiques complexes de façon **simple en BASIC.**

Configuration no 11-2: on peut aussi laisser l'écran en $1E00 et n'utiliser que les 64 premiers caractères programmables. Ce qui laisse plus de place au BASIC. On gagne 1/2 K octet.

Configuration no 12 : utilisable mais sans intérêt.

Note : Si la méthode de choix de l'adresse écran et de l'adresse générateur paraît de prime abord complexe (emploi de AND et OR), elle permet de séparer radicalement les deux opérations. En effet, ces deux adresses se programment grâce au même registre du VIC.

Préparation des couleurs

Couleur du fond

La couleur de fond est celle sur laquelle apparaissent les caractères, elle est donc commune à tous les caractères. Il y a 16 couleurs de fond possible.
Fond : registre VIC-F ($900F ou 36879) bits 4 à 7
Valeur normale : blanc ((VIC-F AND 240)=16)
Exemple : voir ci-dessous.
voir également 'couleur auxiliaire'.

Couleur du bord.

La couleur du cadre entourant la partie utile de l'écran est programmable. Il y a 8 possibilités. Choisir une couleur identique à celle du fond augmente souvent l'agrément du programme, mais cela limite alors le choix de la couleur de fond à 8 possibilités. En règle générale, on programme simultanément la couleur du fond et celle du bord.
Bord : registre VIC-F ($900F ou 36879) bits 0 à 3.
Valeur normale : CYAN ((VIC-F AND 15)=3)

Exemple : POKE 36879,248
La valeur est extraite du tableau ci-dessous.
La valeur en caractère gras dans le tableau est celle choisie à l'allumage du VIC.
Attention : un bord blanc peut créer sur certaines TV un décrochage latéral du haut de l'image : la TV confond le signal "blanc" et les impulsions de synchro-lignes.

	Couleur du bord							
Couleur du fond	noir	blanc	rouge	cyan	mauve	vert	bleu	jaune
noir	8	9	10	11	12	13	14	15
blanc	24	25	26	**27**	28	29	30	31
rouge	40	41	42	43	44	45	46	47
cyan	56	57	58	59	60	61	62	63
mauve	72	73	74	75	76	77	78	79
vert	88	89	90	91	92	93	94	95
bleu	104	105	106	107	108	109	110	111
jaune	120	121	122	123	124	125	126	127
orange	136	137	138	139	140	141	142	143
orange clair	152	153	154	155	156	157	158	159
rose	168	169	170	171	172	173	174	175
cyan pâle	184	185	186	187	188	189	190	191
mauve pâle	200	201	202	203	204	205	206	207
vert pâle	216	217	218	219	220	221	222	223
bleu pâle	232	233	234	235	236	237	238	239
jaune pâle	248	249	250	251	252	253	254	255

La modification de la couleur du bord seule ou du fond seul est bien entendu possible. Utilisez les deux tables ci-dessous.

Couleur du bord	
0	noir
1	blanc
2	rouge
3	cyan (*)
4	mauve
5	vert
6	bleu
7	jaune

(*)couleur intermédiaire entre bleu et vert

couleur du fond ou auxiliaire	
0	noir
1	blanc
2	rouge
3	cyan
4	mauve
5	vert
6	bleu
7	jaune
8	orange
9	orange pâle
10	rose
11	cyan pâle
12	mauve pâle
13	vert pâle
14	bleu pâle
15	jaune pâle

Pour modifier la couleur du bord :

```
10 A=PEEK(36879)AND 248
20 POKE 36879,A OR (16*VALEUR)
```

Pour modifier la couleur du fond :

```
10 A=PEEK(36879)AND 15
20 POKE 36879,A OR (16*VALEUR)
```

Couleur auxiliaire :

En mode '**multicolore**', on dispose de deux fois moins de points sur l'écran mais chaque point d'un caractère peut avoir non plus deux mais quatre couleurs : celle du caractère dans la RAM couleur, celle du fond, celle du bord et l'auxiliaire.

Cette dernière se choisit comme celle du fond : il y a 16 valeurs possibles.

Couleur auxilaire : VIC-E ($900E ou 36878), bits 4 à 7
Valeur normale = 0
Exemple :

```
10 A=PEEK(36878)AND 15
20 POKE 36878, A OR(16*VALEUR)
```

La valeur se trouve dans le tableau ci-dessus.

Inversion :

Les couleurs de bord et de fond peuvent être inversées pour tout l'écran simultanément. Ce n'est guère utilisé si ce n'est comme "effet spécial' pour une explosion ou autre phénomène graphique.

Inversion : VIC-F($900F ou 36879), bit 3
Valeur normale : 0
Exemple :

```
10 A=PEEK(36879)AND 247
20 POKE 36879,A OR 8
```

PHOTOSTYLE : Un crayon lumineux ou photostyle peut être raccordé au VIC par le connecteur de jeux à 9 broches. A la réception d'une impulsion sur ce connecteur, le 6561 verrouille dans deux registres les coordonnées X et Y de la position du balayage écran. Si l'impulsion vient d'une photocellule placée contre l'écran, ce sont les coordonnées de ce point qui sont enregistrées. Ceci permet d'utiliser l'écran comme entrée de données !! La résolution obtenue est de l'ordre de 1/4 de caractère.

Coordonnée horizontale X en VIC-8 ($9008 ou 36872)
Coordonnée verticale Y en VIC-9 ($9009 ou 36873)
Exemple : X=PEEK(36872):Y=PEEK(36873)

Si vous employez un photostyle bon marché, la précision sur la valeur X est nettement moins bonne car le faisceau de balayage TV se déplace bien plus vite horizontalement que verticalement. La solution est soit de changer de dispositif, soit de s'arranger pour utiliser principalement la coordonnée Y. (Voir extensions matérielles).

Entrées analogiques: Les broches 17 et 18 du 6561 (nommées **POT X** et **POT Y**) sont reliées aux broches 9 et 5 du connecteur de jeux à 9 broches. Chacune est également reliée à un condensateur relié à la masse du circuit. Le 6561 peut détecter la valeur d'une résistance externe reliée

entre le +5 volts et une de ces broches d'entrée en mesurant le temps nécessaire à la charge du condensateur. La tension sur la broche 17 ou 18 du 6561 croît d'autant plus vite que la résistance externe est faible. Dès qu'une certaine tension de seuil est atteinte, le 6561 court-circuite la broche à la masse, ce qui décharge instantanément le condensateur et le processus recommence. A chaque période, le 6561 effectue une mesure du temps de charge, qui est donc une image de la valeur de la résistance extérieure. Les valeurs obtenues sont lisibles aux adresses :

POT X = VIC-8 ($9008 ou 36872)
POT Y = VIC-9 ($9009 ou 36873)

(Voir schéma au chapitre 'Extensions matérielles'). Usuellement on utilise des potentiomètres linéaires de 100 Kohms. Pour l'usage "poignées de jeu", les poignées de commande des jeux vidéo ATARI conviennent bien.(en anglais,PADDLES). La précision peut être estimée à environ 5%. Il est possible d'imaginer bien des dispositifs où le VIC utilise ces entrées : en raccordant une résistance à coefficient de température négatif (CTN 100K), on peut aisément transformer le VIC en double thermomètre. On peut également monter un potentiomètre sur l'axe d'un flotteur : on a un capteur de niveau. On peut aussi employer une longue boucle de fil contenant une dizaine de résistances de 5000 ohms et reliant des interrupteurs d'**alarme** tout autour de la maison : le VIC détectera toute ouverture du circuit ou toute modification de résistance de la boucle. Si une des résistances est commandée par la température, le VIC se transformera en détecteur d'incendies... Votre imagination trouvera, c'est certain, bien d'autres applications à ces deux entrées analogiques. Voyez aussi, au chapitre extensions matérielles le paragraphe "Les entrées analogiques du VIC". On trouvera en annexe le programme "POIGNEES DE JEU" démonstratif du principe d'emploi des poignées.

La sortie sonore :

Le 6561 possède une sortie sonore. Le son étant fort important dans les applications de jeu, le 6561 est donc relativement bien pourvu en possibilités sonores. 4 générateurs de signaux rectangulaires à fréquence programmable sont réunis et le signal résultant est envoyé au connecteur de sortie vidéo en passant par un amplificateur dont le gain est programmable.
Trois des quatre générateurs produisent des fréquences stables, programmables sur 7 bits:128 valeurs possibles sur 3 octaves. Chacune de ces 3 voix est différente par sa fréquence de base : normale, alto et soprano. Le quatrième générateur produit un bruit blanc c'est-à-dire dont la répartition suit une courbe de GAUSS ou courbe en cloche dont la fréquence de base est également programmable de la même façon que la première voix. Comme 7 bits servent à choisir une

fréquence, le 8me bit du registre correspondant sert à préciser si le registre en question est en fonction ou pas. Comme le son rendu par un seul générateur de signaux carrés n'est pas très harmonieux, on utilise fréquemment plusieurs voix simultanément avec des valeurs identiques ou proches en tout cas. Voir en annexe le programme **"MINI-ORGUE"** à cet effet.

```
Voix normale en marche : VIC-A bit7=1 ($900A : 36874)
Voix alto en marche    : VIC-B bit7=1 ($900B : 36875)
Voix soprano en marche : VIC-C bit7=1 ($900C : 36876)
Voix bruit en marche   : VIC-D bit7=1 ($900D : 36877)
```

```
Exemple : mise en marche voix alto :
                POKE 36875, PEEK(36875) OR 128

Exemple : arrêt de la voix bruit :
                POKE 36877, PEEK(36877) AND 127
           ou   POKE 36877,0
```

Notez que ce dernier exemple remet également à 0 les 7 premiers bits du registre VIC-D : on modifie donc la fréquence du générateur de bruit.

Programmation des fréquences sonores

Les 7 bits peuvent prendre les valeurs 0 à 127. 0 donne une fréquence basse, 126 donne la note la plus haute. Attention, la valeur 127 donne une fréquence encore plus basse que 0. Pourquoi? Personne ne le sait !

```
Fréquence voix normale : VIC-A, bits 0-6($900A : 36874)
Fréquence voix alto    : VIC-B, bits 0-6($900B : 36875)
Fréquence voix soprano : VIC-C, bits 0-6($900C : 36876)
Fréquence voix bruit   : VIC-D, bits 0-6($900D : 36877)
Valeurs normales : 0
```

Exemple :POKE 36874,(PEEK(36874)AND 127)OR VALEUR ne modifie pas le bit 7, mais en général, le bit 7 est à 1 car on désire entendre le son donc il est suffisant d'écrire :

```
          POKE 36874,VALEUR+128
```

pour mettre en marche la voix normale à une fréquence donnée.

```
          POKE 36874,PEEK(36874)AND 127 :
```

arrêt provisoire.

```
          POKE 36874,PEEK(36874)OR 128 :redémarrage
          POKE  36874,0 :  Arrêt définitif de cette
```

voix

Programmation du volume général

Il y a 16 valeurs possibles de 0 à 15.
15 = volume maximum, 0 = arrêt complet.
Volume : VIC-E, bits 0-3 ($900E ou 36878)
Valeur normale : 0

```
Exemple :   10   A=PEEK(36878)
            20   POKE36878,A OR VALEUR
```

LES MODES GRAPHIQUES

Après avoir passé en revue les différents registres de commande du 6561, voyons ce que l'on peut en tirer au point de vue de l'image. Plusieurs modes graphiques sont possibles : les caractères graphiques, les caractères programmables, la haute résolution en deux couleurs, la résolution moyenne en quatre couleurs.

Les caractères graphiques :

Dès l'allumage de la machine, 128 caractères et leurs inverses sont disponibles, et tous accessibles directement au clavier, ce qui est exceptionnel pour un micro-ordinateur. La facilité et la rapidité du tracé graphique avec ces caractères sont incontestables.

L'emploi des caractères graphiques comprend deux parties distinctes : le choix du caractère en lui-même et le calcul de ses coordonnées pour le positionnement. On verra également comment dessiner des graphiques multi-caractères.

Il y a deux manières d'envoyer un caractère à l'écran : par PRINT et par POKE. PRINT est une instruction plus avantageuse en général car on peut imprimer le caractère et sa couleur simultanément. POKE accède directement à la mémoire de l'écran mais il faut un second POKE pour modifier la couleur du caractère sans quoi il reste invisible (caractère blanc sur fond blanc = caractère difficile à lire !). PRINT gère le curseur,c'est-à-dire une adresse où viendra le prochain caractère, tandis que POKE doit définir chaque fois l'adresse. (voir ci-dessous).

On choisit les caractères "à l'oeil" pour réaliser un graphique. Mais il faut quand même noter que quelques familles de caractères graphiques sont plus intéressantes, notamment par le tracé de barres verticales ou horizontales de diverses longueurs, ou pour les diagrammes. Notez que pour avoir l'inverse d'un caractère, il faut ajouter 128 à son code. En utilisant seulement le caractère 'espace' et son inverse, on peut réaliser des images grossières certes mais relativement aisées à créer.

En annexe, on trouvera le programme "**LORES**" qui sera ensuite extrapolé pour faire la démonstration de tous les autres modes graphiques du VIC. Il est donc important de bien comprendre son fonctionnement. Aussi allons-nous le

détailler.

LORES

Ligne	Commentaire
3	efface l'écran.
4	EC est l'adresse de l'écran : l'adresse 648 contient la partie haute de l'adresse (partie entière de l'adresse divisée par 256).
5	NX et NY sont les nombres maxima de points suivant X et Y. On comptera toujours Y verticalement de haut en bas et X horizontalement de gauche à droite (HOME: X=0, Y=0).
6	CO est l'adresse de la RAM couleur. Si on connaît EC qui est l'adresse de l'écran, cette formule est toujours valable dans tous les cas et dans tous les VIC.
7	On écrit dans tous les caractères utiles de la RAM couleur la valeur 0 soit 'noir'. L'écran reste cependant blanc car il n'y a pas de points"allumés". Mais si on en allume un, il sera noir. Ceci est indispensable car à la ligne 3, le BASIC a remis du blanc (code 1) dans toute la RAM couleur et il est difficile de voir un caractère blanc sur fond blanc !
8	On allume un point dans le coin en haut à gauche de l'écran.
9	Le : ne sert à rien mais rend le texte plus lisible.
10	Entre un caractère au clavier. Si on n'a rien frappé A$ contient la chaîne vide ("").
11	Si le caractère est "HOME", on remet X et Y à 0 : on revient en haut à gauche de l'écran.
12	Si le caractère est "CLR", on efface l'écran en recommençant tout à la ligne 3.
13-16	Les déplacements de curseur modifient X et Y en conséquence.
17-20	Technique de 'passage derrière l'écran' : si X est à fond à droite, il revient à fond à gauche sur la même ligne et de même pour Y. le -1 vient du fait que l'on compte à partir de 0 : donc pour compter 23 caractères, on compte 0, 1, 2,...,21, 22.
21	C est l'adresse du caractère où l'on va écrire, l'adresse du point de coordonnées (X,Y).
22	Si on n'a rien frappé, on ne modifie pas l'image. Supprimer ce test rend le point (X,Y) clignotant mais le tracé n'est plus continu, ce qui n'est pas beau.
23	Si le point (X,Y) est allumé, on l'éteint (code 32 = espace, code 160 = espace inversé, c'est-à-dire carré noir).

24 Si le point (X,Y) n'est pas allumé, on l'allume. Il en résulte que toute frappe au clavier sauf "CLR" inverse la couleur du point XY. (Dans le cas des codes-curseur, X ou Y ont préalablement été modifiés). Essayez une lettre puis la barre d'espace qui bénéficie de la répétition automatique.

25 Et on recommence.

(Voir en annexe le tableau des caractères et les programmes caractères graphiques).

Les caractères à barrettes :

horizontales			
		POKE	CHR$
à fond en haut	1	99	195
	2	69	197
	3	68	196
	4	67	195
	5	64	192
	6	70	198
	7	82	210
à fond en bas	8	100	196

verticales			
		POKE	CHR$
à fond à gauche	1	101	197
	2	84	212
	3	71	199
	4	66	194
	5	93	221
	6	72	200
	7	89	217
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		
taille	POKE	CHR$
1/8	101	165
2/8	116	180
3/8	117	181
4/8	97	161
5/8	246	182 *
6/8	234	170 *
7/8	231	167 *
8/8	160	32 *

Noircis à partir de la droite		
taille	POKE	CHR$
1/8	103	167
2/8	106	170
3/8	118	182
4/8	225	161 *
5/8	245	181 *
6/8	244	180 *
7/8	229	165 *
8/9	160	32 *

Noircis à partir du bas		
taille	POKE	CHR$
1/8	100	164
2/8	111	175
3/8	121	185
4/8	98	162
5/8	248	184 *
6/8	247	183 *
7/8	227	163 *
8/8	160	32 *

Noircis à partir du haut		
taille	POKE	CHR$
1/8	99	163
2/8	119	183
3/8	120	184
4/8	226	162 *
5/8	249	185 *
6/8	239	175 *
7/8	228	164 *
8/8	160	32 *

* = sur fond inversé : précédé de 'REV' et suivi de 'OFF'

Les caractères blocs 4*4 points

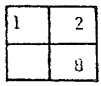
Le VIC contient tous les caractères possibles pour obtenir une résolution par 1/4 de caractères dans les deux axes. Des effets très spéciaux peuvent être obtenus avec ces caractères en mode multicolore. Pour écrire un bloc de 4*4 points dans l'écran, il convient de lire le caractère qui se trouve sous 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 10 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, pour trouver le code du caractère correspondant à une configuration, il suffit d'additionner les numéros des blocs noircis dans le caractère.

Exemple :



sera le caractère numéroté 1+2+8 = 11

Numéro 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'

(Voir le programme MEDRES en annexe, détaillé au paragraphe suivant).

Le positionnement s'effectue différemment suivant que l'on emploie PRINT ou POKE. Avec PRINT, la méthode classique consiste à effectuer des déplacements de curseur : HOME puis descendre de X lignes et avancer de Y colonnes. On peut aussi s'arranger pour grouper plusieurs caractères et déplacements de curseur en une seule chaîne de caractères. La méthode la plus rapide pour positionner le curseur est l'emploi d'une routine en langage machine déjà présente à cet effet dans la ROM KERNAL : pour positionner le curseur en X, Y (X=0 et Y=0, c'est HOME) :

```
POKE 214,X:POKE 211,Y:SYS 58759    ($E587)
```

et c'est fait.

Avec POKE, il faut calculer l'adresse où l'on va écrire et pour cela il faut d'abord connaître l'adresse de l'écran. (7680 dans le VIC de base, 4096 si plus de 8K). Si vous avez modifié cette adresse, conservez bien la nouvelle valeur. Appelons-la ECRAN. Il nous faut également l'adresse de la RAM couleur. Il y a le choix entre 2 possibilités : 37888 ou 38400 ($9400 ou $9600) suivant que l'écran est à une adresse divisible par 1024 ou non. VIC de base : COULEUR=38400. Si plus de 8K, COULEUR=37888. Ceci étant posé, calculons l'adresse d'un caractère de coordonnées X,Y. (0<=X<=21, 0<=Y<=22) :

```
ADRESSE = ECRAN+X+Y*22
ADCOULEUR = COULEUR+X+Y*22
```

En BASIC, on emploiera par exemple :

```
10   EC=7680 : CO = 38800
20   DEF FNA(X)=EC+X+22*Y : DEF FNC(X)=CO+X+22*Y
30   POKE FNA(X),CARACTERE : POKE FNC(X),COULEUR
```

où code est le code POKE du caractère à imprimer et C la couleur de ce caractère (normalement entre 0 et 7 : 0=noir, 1=blanc, etc). A noter que les lignes 10 et 20 ne doivent être exécutées qu'une fois en début de programme. Voir également le programme 'LORES' ci-dessus.

On peut illustrer l'usage des caractères graphiques par un petit programme : le programme "BLOC DATA" est un exemple typique de l'usage des graphiques standards pour un VIC de base. On trouvera également en annexe le programme **"MEDRES"** qui se sert des 16 caractères 'blocs 4*4'. Détaillons-le brièvement :

MEDRES

Ligne	Commentaire
1	Protège le sommet de la mémoire car nous allons changer l'adresse de l'écran pour pouvoir augmenter sa taille. L'écran ira à l'adresse EC=7168 ou $1C00, ce qui nous laisse 1024 octets pour l'écran au lieu de 512. Nous nous contenterons (modestement) de 720 caractères (24*30).
2	Déplace l'écran (air connu !)
3	V est l'adresse du registre VIC-0, NX et NY le nombre maxi de blocs de notre dessin.
4	En V et V+1, les nouvelles marges horizontales et verticales; en V+2 on ne modifie que 7 bits : c'est le nombre de caractères par ligne; en V+3, les bits 1 à 7 sont modifiés : c'est le nombre de lignes de l'écran.
5	CO est l'adresse de la RAM couleur : Notez que le contenu de la () vaut toujours 0 ou -1, donc CO vaut toujours 38400 ou 37888. Ici EC est divisible par 1024, donc la parenthèse est vraie et vaut -1 donc CO = 37888.
6	Met 32, code 'espace' dans tout l'écran et 'noir' dans toute la RAM couleur.
7	Crée (grâce au DATA en ligne 33) un tableau contenant les codes POKE des 16 caractères 'blocs'.
8	Lit un caractère au clavier.

9-14 Gestion des valeurs X et Y en fonction du caractère frappé.

15 Ne s'effectue que si on a frappé un caractère : en effet ASC ne peut s'appliquer à A$ si A$ est une chaîne vide. On teste ensuite si le caractère frappé est un chiffre. Et si oui, on s'en servira pour modifier la couleur du tracé.
Exercice : faire la même chose pour le programme LORES décrit plus haut.
G contient une valeur -1, 0, 1...7. On les interprétera comme les couleurs 0 à 7 ou pour -1, comme l'instruction : ne pas modifier la couleur qui y est déjà. (frappe d'un **zéro** au clavier). Les autres couleurs sont indiquées sur les touches correspondantes.

16-19 Passage 'derrière l'écran.'

20 S et T sont les coordonnées horizontale et verticale du caractère qui contient le bloc (X,Y).

21 C = adresse de ce caractère et CL = valeur du caractère présent dans l'écran avant modification.

22 Si pas de modification de couleur, ne pas exécuter les lignes 23 et 24.

23 Calcul de l'adresse de la couleur du caractère dans la RAM couleur.

24 Ecrit la couleur en cours à cette adresse.(G contient toujours la valeur de la frappe de chiffre précédente).

25-26 Retrouve le numéro du caractère bloc existant à la place de son code POKE. En effet, le numéro est égal à la somme des numéros des carrés 'allumés' dans le caractère, valeur entre 0 et 1+2+4+8= 15.
Notez le I=15 pour forcer la sortie de la boucle FOR au prochain passage : c'est la seule façon propre de sortir d'une boucle FOR-NEXT prématurément.

27 U, V : coordonnées (0 ou 1) horizontale et verticale du bloc à allumer dans le caractère.

28 Calcul du numéro du bloc à allumer à partir de V=0 ou 1 et U=0 ou 1. (Essayez : ça marche !).

30 Si pas de frappe, recommence.

31 Si CL contient le bloc CH, il faut éteindre ce bloc. Pour cela, on prend le caractère existant (numéro CL) et, avec **AND**, on n'en garde que les blocs qui ne sont pas CH. Ce numéro de caractère étant trouvé, on prend dans le tableau A(CL) qui est le code POKE de ce caractère et on l'affiche par POKE.

32 Si CL ne contient pas encore le bloc CH, il faut l'allumer. On force le bit CH - c'est un seul bit car sa valeur est 1, 2, 4 ou 8 soit 2^0, 2^1, 2^2, 2^3 - dans le code CL par **OR**. On prend le code POKE correspondant en A(CH OR CL) et on l'envoie à sa place.

LES CARACTERES PROGRAMMABLES.

Nous avons vu que le générateur de caractères peut être déplacé en mémoire RAM et modifié à volonté. La procédure se divise en trois parties : préparation de la configuration mémoire, calcul des pointeurs et composition des caractères.

La préparation de la mémoire : Il faut décider quelle quantité de caractères on désire modifier. Plus grande sera la zone de caractères et plus petite sera la zone restant pour le programme BASIC. De plus les adresses ne sont pas les mêmes si le VIC possède une extension mémoire de 8K ou plus. Nous avons vu les 12 configurations possibles du générateur de caractères : pour utiliser efficacement l'une de ces variantes en BASIC, il faut choisir celle(s) qui laisse(nt) le maximum de place au programme. Dans un VIC de base, si peu de caractères programmables (- de 64) sont nécessaires, il faut employer la configuration no 11-2 :

- 64 caractères programmables + 128 normaux. **N=64**
- Générateur de caractères en $1C00(7168) **TOP=GE=7168/256=28**
- Ecran en $1E00 (7680) **EC=7680/256=30**
- BASIC en $1000 à $1BFF en VIC de base soit 3072 octets.
  $0400 à $1BFF avec +3K soit 6144 octets.

Pour programmer 128 caractères, employez la configuration 11-1 :

- 128 caractères programmables+128 normaux **N=128**
- Générateur de caractères en $1C00 (7168) **GE=7168/256=28**
- Ecran en $1A00 (6656) **TOP=EC=6656/256=26**
- BASIC en $1000 à $19FF en VIC de base soit 2560 octets.
- $0400 à $19FF avec +3K soit 5632 octets.

Pour obtenir 192 caractères programmables et garder les 64 caractères ROM de base (alphanumériques et ponctuation), employez la configuration 9-2 :

- 192 caractères programmables+64 normaux **N=192**
- Générateur de caractères en $1800 (6144) **TOP=GE=6144/256=24**
- Ecran en $1E00 (7680) **EC=7680/256=30**
- BASIC en $1000 à $17FF en VIC de base soit 2048 octets.
  $0400 à $17FF en +3K soit 5120 octets.

Enfin pour 256 caractères, employez la configuration 9-1 :

- 256 caractères programmables **N=256**
- Générateur de caractères en $1800 (6144) **GE=6144/256=24**
- Ecran en $1E00 (7680) **TOP=EC=5632/256=22**
- BASIC en $1000 à $15FF en VIC de base soit 1536 octets!!!
  $0400 à $15FF en +3K soit 4608 octets.

Dernière possibilité pour un VIC de base (configuration 7), 256 caractères programmables mais sans déplacer l'écran qui reste en $1E00 :

- 256 caractères programmables **N=256**
- générateur de caractères en $1400 : **TOP=GE=5120/256=20**
- écran en $1E00 **EC=7680/256=30**
- BASIC en $1000 à $13FF soit 1024 octets!!!
- de $0400 à $13FF avec +3K soit 4096 octets.

Cette solution est proposée par de nombreux auteurs comme la seule façon d'avoir 256 caractères programmables. Il est évident que la configuration 9-1 est plus intéressante : on récupère 5120 octets pour le BASIC et les touches SHIFT-LOGO donnent accès au jeu de caractères d'origine, ce qui n'est pas le cas de cette configuration-ci.

Dans un VIC possèdant 8 K ou plus d'extensions les problèmes d'espace sont moins critiques et on n'emploiera usuellement qu'une seule configuration décrite ci-dessous mais les autres sont utilisables aussi, bien entendu :

- 256 caractères programmables — N=256
- Générateur de caractères en $1400 (5120) — GE=5120/256=20
- Ecran en $1000 (4096) — EC=4096/256=16
- BASIC à partir de $1C00 (7168) (au lieu de $1200 = 4608) — BA=7168/256=28

Le calcul des pointeurs

Ayant porté notre choix sur une de ces possibilités, modifions maintenant les pointeurs de manière à déplacer à leur juste place les différents protagonistes de la scène.

Premier point délicat, pour les VIC avec plus de 8K, il faut modifier le pointeur de début de BASIC. Ceci doit se faire en mode direct car cela a pour effet d'effacer le programme actuellement en mémoire : BASIC doit démarrer en 7168 soit 256*28 (28 est la valeur BA du paragraphe précédent)
Mettre BASIC en 7168 :

```
BA=28:POKE 7168,0:POKE43,1:POKE44,BA:NEW
```

Notez le 0 en 7168 : sans lui l'interpréteur BASIC refuse d'exécuter la première ligne du programme. Pour remettre BASIC à sa place en 4608 on aurait dû faire :

```
BA=18:POKE 4608,0:POKE 43,1:POKE 44,BA:NEW
```

Pour les VIC de base ou +3K, ce problème ne se pose pas, BASIC ne change pas d'adresse au départ. Par contre, pour eux, c'est le sommet de BASIC qui doit redescendre pour faire place aux caractères programmables. Ceci n'efface heureusement que les variables et non le programme. Le sommet de mémoire sera établi à l'adresse **TOP** définie dans les configurations ci-dessus. (Si vous avez l'extension 8K, ne faites pas ceci : la fin de BASIC arriverait avant son début!).

```
10 HI=TOP:LO=0
20 POKE 53,LO:POKE 54,HI
30 POKE 55,LO:POKE 56,HI :CLR
```

Il ne nous reste plus qu'à modifier les pointeurs d'écran et de générateur de caractères:

```
40 GE=GE/2 :REM BITS 10-13 de l'adresse
50 POKE 36869,PEEK(36869)AND240 OR GE: REM générateur
60 POKE648,EC:SYS 58648:REM écran
```

La programmation des caractères :

L'ordinateur est maintenant prêt à afficher les caractères programmables. Mais encore faut-il les programmer ! Le dessin d'un caractère occupe 8 octets représentant les 8 lignes élémentaires de 8 points. Pour situer un caractère dans le générateur, prendre son code POKE*8 : c'est l'adresse relative du premier octet dans le générateur. Si on ne désire modifier qu' un ou deux caractères, il faut programmer la mémoire ppour le minimum de codes programmables, soit 64 avec la configuration 11-2. Puis on recopie les 64*8=512 premiers octets du générateur de caractères en ROM dans le nouveau générateur de caractères en RAM. A ce moment, il n'y a plus de différence avec le jeu de caractères d'origine, si ce n'est que chaque caractère est maintenant modifiable.

Exemple : Nous allons modifier un seul caractère, le chiffre 0 dont le code POKE est 48. (Voir le programme "**CARA**" en annexe)

1. Configuration 11-2 : l'écran est en 7680, le générateur en 7168 donc :

```
10 POKE 53,0:POKE 54,28:POKE 55,0:POKE 56,28:CLR
20 POKE 36869,PEEK(36869)AND 240 OR 15
```

2. Recopier 512 octets du générateur ROM en 7168 et suivants :

```
30 FOR I=0 TO 511:POKE 7168+I,PEEK(32768+I):NEXT I
```

3. Repérer les octets à modifier (code POKE48)

```
40 C=7168+48*8
```

4. Modifier les 8 octets :

```
50 FOR I=0 TO 7 : READ A:POKE C+I, A:NEXT
60 PRINT "0":END
70 DATA 0,62,127,99,99,127,62,0
```

Dans l'exemple, on peut voir une manière simple de modifier un caractère. Mais comment trouve-t-on les 8 valeurs correspondant à un caractère ?

Première règle : la ligne supérieure du caractère, c'est le premier octet

Deuxième règle : la colonne gauche du caractère, ce sont les bits 7 de chaque octet.

La conversion se fait, soit à l'aide d'un programme, soit à la main ! On dessine le caractère sur une grille de 64 cases . Ainsi numérotées et on en déduit les valeurs :

128	64	32	16	8	4	2	1	
								=0
		X	X	X	X	X		=32+16+8+4+2=62
	X	X	X	X	X	X	X	=64+32+16+8+4+2+1=127
	X	X				X	X	=64+32+2+1=99
	X	X	X	X	X	X	X	=64+32+16+8+4+2+1=127
		X	X	X	X	X		=32+16+8+4+2=62
								=0

Attention au curseur! Le curseur disparaît lors de cette manoeuvre. En effet, le curseur clignotant inverse régulièrement le fond du caractère. En pratique, l'interpréteur BASIC inverse le bit 7 du caractère sous le curseur. Il en résulte que, à l'affichage, le caractère provient une fois sur deux de la première partie (128 caractères) du générateur et le reste du temps de la seconde partie qui est sensée contenir les dessins inversés des caractères. En recopiant en RAM la première moitié ou le premier quart des caractères ROM, il se crée deux demi-générateurs identiques et il n'y a donc plus accès aux caractères inversés. Le caractère sous le curseur a toujours son bit 7 qui clignote, mais ceci corespond à deux dessins de caractère identiques : on ne voit donc plus le curseur même s'il est toujours là.

La combinaison STOP-RESTORE rétablit le générateur de caractères en ROM mais ne rétablit pas l'adresse de l'écran ni le sommet de mémoire vu du BASIC. Dès que les caractères programmables ne sont plus nécessaires, on peut rétablir l'écran et le sommet de mémoire aux valeurs de départ, mais en général ce n'est pas nécessaire.

La programmation de nombreux caractères implique souvent une grande occupation de mémoire par des valeurs stockées en DATA. Il y a donc deux solutions en pratique. La première : séparer le programme en deux parties : chargement des caractères puis programme proprement dit. La seconde : n'employer qu'un seul programme mais le faire suivre sur la cassette ou sur un disque par un **fichier de données** contenant les différentes valeurs. Par la même occasion, ce fichier peut contenir les chaînes de caractères contenant le mode d'emploi du programme ou toutes autres informations auxquelles on ne fait référence qu'une seule fois.

Exemple : pour sauver sur cassette n valeurs numériques correspondant aux octets du générateur de caractères, on crée d'abord le fichier de données sur papier à partir du dessin des caractères puis on l'introduit sur la cassette avec le petit programme suivant :

```
10 OPEN1,1,2,"FICHIER"
20 INPUT A
30 IF A=999 THEN CLOSE 1:END
40 PRINT£1,A
50 GOTO 20
```

A noter : on attend la frappe de la valeur 999 pour refermer le fichier. Après avoir créé ce fichier de données, il faut écrire le programme proprement dit sur la même cassette mais juste avant, pour que en fin de "LOAD", la cassette soit correctement positionnée. Le programme principal lui-même, contiendra l'instruction de lecture des données comme ceci :

```
10 OPEN 1,1,0,"FICHIER"
20 INPUT£1,A
30 .....(ici vous utilisez A).....
40 IF ST>3 GOTO 20 :REM fin de fichier ??
50 CLOSE 1:.....suite du programme....
```

Remarquez en ligne 40 l'utilisation de la variable réservée **ST-status.**

Cette méthode élégante peut faire des merveilles. Son inconvénient est la copie des fichiers, qui n'est guère évidente. En effet, pour copier un fichier de données, il faut écrire un programme qui lit toutes les données, puis attend que l'on change de bande et les recopie toutes ensuite. Voici un exemple de copie d'un fichier contenant exactement 512 octets (valeurs entre 0 et 255) et nommé "FICHIER" :

```
10 REM COPIE FICHIER cassette
20 OPEN 1,1,0,"FICHIER"
30 DIM A%(511)
40 FOR I=0 TO 511: INPUT£1,A:A%(I)=A:NEXT I
50 CLOSE 1
60 PRINT "Changez de cassette et tapez RETURN"
70 GETA$ : IF A$=""GOTO 70
80 OPEN 1,1,2,"FICHIER"
90 FORI=0 TO 511:A%(I)=A:PRINT£1,A:NEXT I
100 CLOSE 1
```

Utilisateurs de disquettes, remplacez `OPEN 1,1,0,"FICHIER"`
par `OPEN 1,8,1,"FICHIER,SEQ,READ"`
et `OPEN 1,1,2,"FICHIER"`
par `OPEN 1,8,2,"FICHIER,SEQ,WRITE"`.

Un avantage supplémentaire des fichiers cassettes est le fait que ceux qui ne possèdent pas les caractéristiques (nombre et type de variables) de vos fichiers ne pourront faire de copies de vos données sans devoir au préalable lire ce livre puis "décortiquer" votre programme !!

NOTE: Les caractères du VIC sont cadrés à droite et en haut dans le bloc de 8 x 8 points qui leur est attribué.

LA HAUTE RESOLUTION BICOLORE.

On a vu plus haut que le plus grand écran accessible en haute-résolution en BASIC représente 192 caractères de 8*16 points avec la configuration 8. (Générateur de caractères en $1400). Si l'écran reste en $1E00, on perd 512 octets et on ne peut donc employer que 160 caractères de 8*16 points et par exemple programmer la taille de l'écran en 16 caractères de large et 10 lignes de haut (doubles) : on obtient un rectangle vertical assez plaisant et il reste $1400-$1000=$04000=1024 octets pour un programme BASIC. C'est trop peu pour créer un jeu mais suffisant pour créer des images à partir du clavier, du manche de commande, du photostyle ou d'un fichier de données sur cassette...

Mais il est évident que 3K d'extension ou plus sont recommmandés vivement pour réaliser des programmes évolués. Dans ce cas, on déplacera l'écran en $1000 ou $1200. Cette dernière variante nous donne 192 caractères 8*16 et il reste encore $1200-$0400=3584 octets en BASIC avec l'extension 3K, ou $0200 soit 512 octets pour un VIC de base. Nous utiliserons pour illustrer ce paragraphe cette dernière configuration également possible pour tous les types de VIC. Il reste bien évident que d'autres configurations sont possibles.

Configuration HIRES

- écran en $1200 à $13FF soit 4608 à 5119
- générateur en $1400 à $1FFF soit 5120 à 8192
- début de BASIC -inchangé et reste en $0400 (VIC+3K)
  -inchangé et reste en $1000 (VIC de base)
  -doit passer de $1200 à $2000 (VIC avec 8K et plus)
  (voir caractères programmables)
- la RAM couleur est donc en $9600 (38800) comme dans le VIC de base
- Taille de l'écran :
  largeur : 24 caractères * 8 points = 192 points
  hauteur : 8 lignes * 16 points = 128 points
- Taille des caractères : 8 * 16

Principe de fonctionnement :

En haute résolution, chaque caractère de l'écran est différent et ils sont tous programmables. Autrement dit, il suffit d'écrire dans l'écran les 192 caractères numérotés de 0 à 191 (on peut maintenant ne plus y penser) pour constater que chacun des 3072 octets de RAM du générateur de caractères correspondent à 8 points consécutifs de l'écran. Il nous reste à définir les primitives de base permettant l'allumage ou l'extinction d'un point de l'écran. Ceci se joue en quatre étapes : pour un point donné par ses coordonnées X, Y -origine en haut à gauche- trouver où il est situé :

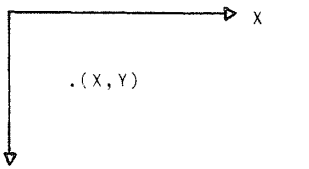
1. Dans quel caractère ? CH=?
2. Dans quelle ligne du caractère ? ROW=?
3. A quel octet correspond cettte ligne ? OCT=?
4. Quel bit de cette ligne-octet ? BIT=?

Mémoire-écran :

```
  0  2  3  ......                          23

 24  ............                          47

  .
  .
  .
  .
  .
168  ........                  ......  192
```

La technique que nous allons détailler peut s'adapter aisément à d'autres tailles d'écran que 24*8. 24*8 donne un rectangle très large. Essayez 19*10, vous aurez un écran de proportions inverses.

Soit X et Y les coordonnées du point de l'écran.



* Calcul du caractère contenant (X,Y) : 0<=CH<=191
CH = INT(X/8) + INT(Y/16) * 24
(Noter que 24 = nombre de caractères par ligne, 8 = nombre de points par caractère en largeur, 16= nombre de points par caractère en hauteur).

* Calcul de la rangée du caractère. 0 <= ROW <= 15
ROW = (Y/16 - INT(Y/16) * 16)

* Calcul de l'adresse de l'octet 520 <= OCT <= 8191
OCT = 5120 + 16*CH + ROW
(Noter que 5120= adresse de début du générateur de caractères).

* Calcul du bit 0 <= BIT <= 7
BIT = 7 - (X - (INT(X/8) * 8))

* Ecriture du point en X,Y par :
POKE OCT , PEEK (OCT) OR (2^BIT)

* Effacement du point en X,Y par :
POKE OCT , PEEK (OCT) AND (255 - 2^BIT)

Important : Il est indispensable d'être certain des valeurs de X et Y en entrée car pour des valeurs inexactes, on risque d'"écrire" des bits n'importe où en mémoire et de modifier le programme BASIC lui-même !

Pour écrire des lettres en haute résolution, il suffit de recopier les octets voulus de la ROM générateur de caractères à l'endroit voulu.

Il peut également être utile de tester si un point de l'écran est allumé ou éteint avec :
IF PEEK (OC) AND (2^BI) THEN... (point allumé)...

Exemple : Le programme **HIRES** en annexe peut fonctionner dans un VIC de base, mais il a dû être extrèmement compacté : en exécution, il reste 11 octets libres! Passons en revue les éléments de ce programme :

HIRES

1. Protéger le sommet à partir de $1200 par :
   POKE 54 , 18 : POKE 56 , 18 : CLR
   Noter que l'on n'a pas mis de POKE en 53 et 55 car cette valeur est déjà correcte à l'allumage du VIC : si on l'a modifiée au préalable, le programme ne fonctionne pas.

2. V=36864 est l'adresse du registre VIC-0. Ceci permet de gagner quelques octets sur la longuer de la ligne 4.
   POKE 648 , 18 : SYS 58648 déplace l'écran en $1200. (18*256)

3. FOR I = 0 TO 191 : POKE 38400 + I , 0 : POKE 4608 + 1 , I : NEXT
   remet à 'noir'la RAM couleur et écrit les valeurs 0 à 191 dans la mémoire écran.

4. POKE V,10 :réduit la marge gauche
   POKE V+2,152 :nombre de caractères par ligne = 24
   POKE V+3,17 :nombre de lignes=8 et caractères doubles
   POKE V+5,205 ;adresses écran et générateur
   Noter l'emploi de valeurs numériques. Les classiques AND et OR prennent trop de place en mémoire d'un VIC de base.

5. FOR I = 5120 TO 8191 : POKE I , 0 : NEXT remet à zéro le générateur de caractères.
   CLR : efface les variables V et I pour récupérer de la place en mémoire.

6 à 11. Modifie X et Y suivant les frappes au clavier.

12 à 15. Teste la validité de X et Y avec "passage derrière l'écran'(en anglais WRAP-AROUND).

16. S = INT (X/8) : T = INT (Y/16) Coordonnées du caractère contenant le point.

17. C = 5120 + 16 * (S + T*24 + Y/16 - T) : Adresse de l'octet contenant le point. C'est une version compactée des lignes CH=...,ROW...,OCT... ci-dessus.

18. Si pas de frappe au clavier, attendre.

19. IF PEEK (C) AND (2^B) THEN : on teste si le point est allumé.
    POKE C,PEEK (C) AND (255 - 2^B) : GOTO 6 : On éteint le point s'il était allumé.

20. POKE C , PEEK (C) OR (2^B) : GOTO 6 : Si le point était éteint, on l'allume.

On voit donc que les lignes 19 et 20 inversent la couleur du point X, Y et que si l'on a frappé une touche autre que les déplacements de curseur, CLR ou HOME, la couleur du point s'inverse. En maintenant la barre d'espace enfoncée, on voit clignoter le point ce qui permet de le situer.

La méthode dynamique :

Il existe une variante permettant d'avoir un écran bien plus grand que 8*24 tout en conservant la même résolution. Par exemple, en programmant la taille de l'écran à 15*24. Evidemment, il n'y a pas assez de caractères pour accéder à toute la surface de l'écran. On réservera donc cette technique aux dessins peu chargés et où on ne dépasse pas le nombre de caractères programmables. Il faut au départ remplir l'écran avec un caractère vide, par exemple celui dont le code est 0. Puis à chaque point que l'on écrit à l'écran, ou bien il tombe sur un caractère encore vierge (code = 0) et on le remplace par un nouveau caractère que l'on crée à ce moment, ou bien il tombe sur un caractère déjà existant et on modifie ce caractère en conséquence. Le programme étant plus complexe, on utilisera (pour le VIC de base en tout cas) de préférence une configuration avec 64 ou 128 caractères programmables. (Un écran de 15*24=360 caractères pourra être rempli au tiers environ de sa surface avec 128 caractères programmables). La meilleure manière d'illustrer cette méthode est d'analyser le programme **DYNAHIRES** en annexe. Le fonctionnement est identique à celui du programme HIRES mais l'écran est plus grand : 15*24 caractères doubles soit la totalité de l'image TV y compris les bords. Lorsque 128 caractères ont été définis, la couleur du dessin passe de noir à bleu et attend l'enfoncement de la touche "CLR" pour redémarrer. Il faut être patient au démarrage du programme : il lui faut environ 20 secondes pour démarrer.

Il est encore possible d'améliorer les effets dans ce mode graphique en employant des motifs répétitifs qui réemploient les mêmes caractères à plusieurs endroits de l'écran en "consommant" bien moins de caractères. On peut évidemment colorer certaines parties de l'écran en modifiant les zones voulues de l'écran : ciel bleu, toit rouge, maison noire, herbe verte, etc...

DYNAHIRES :

Ligne 10 :Protège le sommet de mémoire en $1600 (5632).

Ligne 15 :Taille de l'écran en nombre de caractères : 24*15

Ligne 16 :Nombre maxi de points en horizontal et vertical : 192*240

Ligne 20 :Valeur de départ X, Y au milieu de l'écran. V=36864 est l'adresse du registre VIC-0.

Ligne 30 :En V et V+1, on modifie les marges gauche et supérieure. En V+2, la largeur de l'écran. En V+15, les couleurs du fond et du bord sont mises à blanc.

Ligne 40 :En V+3, on programme les caractères doubles et le nombre de lignes. En V+5, on place l'écran en $1600 et le générateur de caractères en $1C00. (Configuration 10 du tableau adresses générateur).

Ligne 50 :Peinture de l'écran en noir.

Ligne 60 :Attendre si fin de programme avant de recommencer.

Ligne 70 :Remplir l'écran de codes 127 (le caractère espace dan ce cas précis).

Ligne 80 :Effacement du générateur de caractères.

Ligne 90-190 :Entrée d'un caractère et modification de X et Y en fonction avec "passage derrière l'écran". La barre d'espace fait clignoter le point X, Y.

Ligne 200 :S et T sont les coordonnées H et V du caractère où se trouve le point.

Ligne 210 :AD est l'adresse de ce caractère dans la mémoire-écran.

Ligne 230 :C est le code du caractère à l'endroit où l'on doit écrire. Si C = 127, on n'a encore rien écrit dans le caractère, on y met donc un caractère pas encore utilisé et on incrémente N, le compteur de caractères utilisés.

Ligne 240:Teste s'il reste encore des caractères.Si non, c'est fini.

Ligne 250:Calcul de l'adresse de l'octet concerné dans le générateur de caractères = adresse de début du générateur (6144) + avance du nombre de caractères nécessaires (16*C)+avance d'une fraction de caractère, valeur entre 0 et 15 (16*(Y/16-T)). Position du bit concerné = 7-X+5*8 Où (X-5*8) représente la position du bit dans l'octet à partir de la gauche (bit 7).

Ligne 270:Si on n'a rien frappé au clavier, on recommence.

Ligne 280:Teste si le point est déjà allumé et, si oui, l'éteint comme dans le programme HIRES.

Ligne 290:Dans le cas contraire, on allume le point concerné.

LA RESOLUTION MOYENNE MULTICOLORE

Jusqu'ici, on n'a guère fait usage de la RAM couleur sauf pour y mettre une valeur fixe, couleur de chaque point d'un même caractère. Les valeurs employées étaient toujours comprises entre 0 et 7. Or la RAM couleur est une mémoire à 4 bits dont les valeurs peuvent aller de 0 à 15. Les 8 valeurs non encore employées jusqu'ici nous ouvrent un monde nouveau : le mode multicolore dont la clé est mettre à 1 le bit 3 de la RAM couleur. Que faire après avoir tourné cette clé ?

En mode normal (haute résolution), il y a pour chaque caractère deux couleurs possibles : une parmi huit rangée dans un octet de la RAM couleur propre à chaque caractère et une parmi seize rangée dans le registre VIC-F et donc commune à tout l'écran (en anglais, couleurs FOREGROUND et BACKGROUND).

L'usage de ces deux couleurs est interchangeable en modifiant le bit 3 du registre VIC-F.

En mode multicolore, il y a pour chaque caractère quatre couleurs possibles : les couleurs possibles sont la couleur de fond, la couleur d'écriture (comme en mode normal) plus la couleur du bord et la couleur auxiliaire.

Mode multicolore :

Numéro de la couleur		Adresse de programmation	gamme de valeurs	Désignation
décimal	binaire			
0	00	registre VIC-F bits 4-7	0-15	couleur de fond (background)
1	01	registre VIC-F bits 0-2	0-7	couleur de bord (border)
2	10	RAM couleur bits 0-2	0-7	couleur d'écriture(foreground)
3	11	registre VIC-E bits 4-7	0-15	couleur auxiliaire (auxiliary)

On voit bien que 3 de ces couleurs sont communes à tout l'écran alors que la dernière n'est commune qu'aux divers points d'un même caractère. Reste à choisir pour chaque point parmi les 4 couleurs celle qui sera utilisée : on le réalise en utilisant **deux** bits et non un pour chaque point de l'écran. Le VIC dans ce mode utilise donc deux fois moins de points, mais ceux-ci sont deux fois plus gros : leur largeur est doublée. Il est donc évident qu'un générateur de caractères conçu pour le mode haute résolution ne donne aucun résultat avec Le mode multicolore.

Pour les caractères programmables, la méthode classique de construction de caractères à la main sur feuille quadrillée reste valable et n'appelle que peu de modifications. Par contre, pour les tracés graphiques, il faut adapter les calculs d'adresses de points suivant la direction horizontale X, la résolution en largeur étant moitié moindre. La complexité des programmes s'accroit légèrement par rapport au mode dynamique vu ci-dessus mais reste supportable.

IMPORTANT : les effets graphiques les plus spectaculaires sur le VIC sont obtenus en se servant de toutes ses possibilités simultanément : certains caractères en haute résolution, d'autres multicolores sont possibles sur le même écran.

Programmer la couleur du fond (VIC-F, bits 4-7):

```
10   CO = COULEUR (ENTRE 0 ET 15 -VOIR TABLEAU CI-DESSOUS)
20   A = PEEK (36879) AND 15 : POKE 36879 , A
30   POKE 36879 , A OR (CO*16)
```

Programmer la couleur du bord (VIC-F,bits 0-2):

```
10   C1 = COULEUR (ENTRE 0 ET 7)
20   A = PEEK (36879) AND 248 : POKE 36879 , A
30   POKE 36879 , A OR C1
```

On peut aussi programmer ces deux couleurs à l'aide de la table déjà citée en début de chapitre ou par calcul commme ci-dessous :

```
10   A = PEEK (36879) AND 8
20   POKE 36879 , A OR (C0*16) OR C1
```

Programmer la couleur auxiliaire (VIC-E,bits 4 à 7):

```
10   C3 = COULEUR (ENTRE 0 ET 15 - COMME COULEUR DE BORD)
20   A = PEEK (36878) AND 15 : POKE 36878 , A
30   POKE 36878 , A OR (C3*16)
```

Programmer la couleur d'écriture d'un caractère quelconque:

```
10   C2 = COULEUR (ENTRE 0 ET 7 -COMME COULEUR DE BORD)
20   CH = ADRESSE DU CARACTERE PAR RAPPORT AU DEBUT DE L'ECRAN
30   CR = ADRESSE DE LA RAM COULEUR (38400 ou 38800)
40   POKE CR + CH , 8 AND C2
```

Noter le 8 AND et non 8 + qui est utilisable aussi : le AND est plus rapide et ne conduit pas à des erreurs de mode en cas de mauvaise valeur de C2. Le 8 correspond au bit 3, c'est-à-dire : mode multicolore.

couleurs de bord et d'écriture	
0	noir
1	blanc
2	rouge
3	cyan
4	mauve
5	vert
6	bleu
7	jaune

Couleurs supplémentaires pour fond et auxiliaire	
8	orange
9	orange pâle
10	rose
11	cyan pâle
12	mauve pâle
13	vert pâle
14	bleu pâle
15	jaune pâle

L'écriture dans un caractère proprement dit : nous allons l'illustrer par la programmation du caractère "drapeau français".
Choisisssons le blanc commme couleur de fond. R=rouge, B=bleu, N=noir

On aura le dessin suivant :

Couleurs	Octet	
N B R	10110001	= 177
N B R	10110001	= 177
N B R	10110001	= 177
N B R	10110001	= 177
N	10000000	= 128
N	10000000	= 128
N	10000000	= 128
N N	10100000	= 160

	binaire	décimal
R=couleur de bord	(01)	1
B=couleur auxiliaire	(11)	3
N=couleur d'écriture	(10)	2
espace=couleur de fond	(00)	0

En utilisant quelques caractères différents pour créer une figurine, on peut obtenir des résultats extraordinaires en utilisant les caractères multicolores pour le corps du personnage et un caractère haute résolution (noir sur blanc) pour le visage. Si on modifie ce caractère seul, on peut obtenir des expressions différentes, les caractères multicolores étant bien plus agréables pour le dessin de vêtements, animaux, etc...

Nous reprenons également notre désormais classique programme de dessin **DYNAHIRES**, modifié pour la circonstance. En mode multicolore, on a ainsi accès à 4 couleurs suivant que l'on enfonce les touches numériques correspondantes : 1 pour noir, 2 pour blanc, 3 pour rouge, 7 pour bleu. Attention, le programme ci-dessous dans un VIC de base, ne laisse que 51 octets libres après exécution !!

MULTIRES

Ligne	Commentaire
2	Place le sommet de mémoire en 20*256=5120($2400)
4	24*30 caractères (en 8*8)soit 96*240 points : les points sont très larges (comme 2 points de haute résolution) : il y a donc 4*8 points par caractère.
6	X,Y : coordonnées de départ du tracé au milieu de l'écran. V est l'adresse du registre VIC-0.
8	C0 est l'adresse de la RAM couleur : on n'a pas utilisé la formule générale pour gagner de la place en mémoire.
10	En V et V+1, programmation des marges. En V+2, (bits 0-6), nombre de caractères par ligne.
12	En V+3, nombre de lignes et caractères de 8*8 points. En V+5, adresse du générateur de caractères en page 24 : 24*256=6144 ($2800 pour le 6502, $3800 pour le 6561). G est l'adresse du générateur de caractères.
14	V=14: programmation de la couleur auxiliaire (96,c'est 3*32 et 3 c'est bleu). V+15 : la valeur 26 donne fond blanc, bord rouge.

16 Remplit l'écran avec le caractère 255.

18 Remplit la RAM couleur avec du noir (couleur d'écriture).

20 Remplit le générateur de caractères avec des 0 : le générateur contient donc dans chaque octet 4 paires de 0. Or 00=blanc : l'écran devient donc blanc dès que les 8 octets du caractère 255 se remettent à 0.

22-42 Gestion des déplacements (voir les autres programmes de la série).

44-50 Test des frappes 1, 2, 3 et 7 pour modifier la couleur dans laquelle s'effectue le tracé. Celle dernière est imposée par la valeur V.

52 S et T sont les coordonnées de la paire de bits (X,Y) dans le caractère. AD est l'adresse du caractère contenant le point dans la mémoire écran.

54 C'est le numéro du caractère. On teste s'il a déjà été utilisé ou non. (non utilisé= code 255). N contient le nombre de caractères déjà programmés.

56 Si on a employé tous les caractères c'est fini.

58 OC est l'octet du générateur qui contient la paire de bits caractérisant le point. B est l'adresse (le numéro du bit) du bit de poids faible de la paire de bits à l'intérieur de l'octet. Il faut noter que B ne peut prendre que les valeurs 0, 2, 4 ou 6.

59 Si on ne doit pas modifier le point, on recommence la boucle.

60 Le test IF PEEK (OC) AND (3*2^B) THEN... teste si la paire de bits (X,Y) est différente de 0. Si oui, on y a déjà écrit quelque chose, donc on laisse programmé ce point mais en modifiant sa couleur. On réécrit donc (POKEOC,) la valeur de l'octet (PEEK(OC)) sauf les deux bits concernés (AND(255-3*2^B)) que l'on remplace par les deux bits de couleur 3-U au lieu de U. Par exemple, noir (U=2) devient rouge (3-U=1). Notez que la couleur U est la couleur avec laquelle on trace et non la couleur qui existait à l'endroit du point (X,Y). La couleur du point X,Y ne sert pas dans le programme; on l'aurait calculée par U=(PEEK(OC)AND(3*2^B))/2^B.

62 Si la paire de bits (X,Y)=0, le point (X,Y) est blanc, donc on peut y écrire un point de couleur U. Comme ci-dessus, on conserve la partie de l'octet qui ne doit pas être modifiée. On rajoute la valeur (U*2^B) par l'opération OR.

64 Pour terminer le programme, (on arrive ici de la ligne 56), variation aléatoire de la couleur du fond et du bord. Plus sobrement, on peut écrire aussi :

```
64   GET A$ : IF A$= "" GOTO 64
66   GOTO 16
```

(Attention, il ne reste plus beaucoup de mémoire libre).

Texte et graphiques.

Si on n'utilise que 64 caractères programmables en 8*8, on dispose encore des caractères alphanumériques. Si on les programme tous, cet avantage disparaît. Il y a donc deux solutions : ou bien tracer des lettres en haute résolution, point par point, ou bien reprogrammer simplement un caractère 8*8 ou un demi-caractère 8*16 avec les huits octets d'un caractère alphanumérique qu'il suffit d'aller lire successivement dans le générateur de caractères en ROM. Soit C le code du caractère à dessiner, le caractère se programmera donc avec les 8 valeurs situées en 32768 + C*8 à 32768 + C*8 + 7

Ceci n'est cependant pas applicable en mode multicolore. Rien n'empêche cependant d'utiliser en mode multicolore quelques caractères dans un autre mode et appliquer la méthode ci-dessus. Il suffit de mettre à 0 le bit 3 de la RAM couleur dans l'octet correspondant au caractère. Avec la RAM couleur en CO, le caractère en (X,Y) et des lignes de NX caractères, il est difficile de passer du mode multicolore au mode ordinaire par :

POKE C , PEEK (C) AND 7

LES DEUX VIA.

Il y a dans le VIC 4 circuits intégrés L.S.I. (à très haute densité) munis de 40 broches : les 2 processeurs et les 2 **VIA**. VIA est l'abréviation de "Versatile Interface Chip" ou circuit d'interface à usage multiple. La complexité interne d'un VIA est équivalente à celle d'un microprocesseur. Du point de vue de l'utilisateur un VIA présente 20 entrées ou sorties digitales utilisables pour y connecter divers périphériques. Du point de vue du microprocesseur, le VIA représente 16 adresses en mémoire qui sont les registres de contrôle du VIA. De plus, le VIA possède une sortie spéciale **interruption** dont le but est de signaler au processeur l'apparition de certains événements particuliers relatifs aux périphériques qu'il contrôle.

Dans le VIC, les VIA sont des 6522, construits par Mos Technology, filiale de COMMODORE et dont la conception est due à l'ingénieur CHUCK PEDDLE, concepteur du 6502 et du P.E.T. Le circuit est probablement le VIA à 8 bits le plus sophistiqué disponible à ce jour.

Les deux VIA sont caractérisés dans le VIC par le type d'interruption qu'ils génèrent. Nous avons vu que le 6502 possède deux entrées d'interruption,une masquable, **IRQ** que le programme peut ne pas reconnaître s'il le désire et une non masquable, **NMI**. Dans le VIC, le VIA 1 génère les interruptions NMI et le VIA 2 les interruptions IRQ.

En mémoire, le VIA 1 occupe les adresses $9110 à $911F, le VIA 2 les adresses $9120 à $912F. En fait, on constate que le décodage d'adresses du VIC est incomplet et que les adresses $9010 à $93FF contiennent des "images" plus ou moins complètes de l'un ou l'autre VIA. Par exemple, les adresses $9130 à $913F sélectionne les 2 VIA à la fois, ce qui peut être extrèmement dommageable pour les VIA ou même pour les amplificateurs du bus de données. Une simple lecture à une de ces adresses provoque la liaison de 2 circuits simultanément au bus de données. Si ces deux circuits imposent des valeurs différentes au bus, il y a un conflit et ceci provoque les échauffements anormaux des cir-

cuits. En résumé, si vous souhaitez longue vie à votre VIC, n'utilisez jamais les adresses $9010 à $910F (36880 à 37135) ni les adresses $9130 à $93FF (37169 à 37887).

Nom du VIA	Type d'interruption	Adresses (hexa et déc.)	
VIA 1	NMI	$9110-$911F	37136-37151
VIA 2	IRQ	$9120-$912F	37152-37167

Usage du VIA 1 :Port A :

*lecture de la touche RESTORE	(entrée)
*interface IEEE série : horloge	(entrée)
donnée	(entrée)
attention	(sortie)
*lecture de l'interrupteur cassette	(entrée)
*moteur cassette	(sortie)
*entrées joystick 0, 1, 2, TIR	(entrée)
Port B :	
*libre pour l'utilisateur (USER'SPORT)	(entrées/sorties)
ou (au choix)	
*interface RS.232	(entrées/sorties)

Usage du VIA 2 :Port A :

*lecture cassette	(entrée)
*lecture des 8 rangées clavier	(entrées)
*interface IEEE série: horloge	(sortie)
Port B :	
*interface IEEE série : données	(sortie)
service	(entrée)
*sortie des 8 colonnes clavier	(sorties)
*écriture cassette	(sortie)
*entrée joystick 3	(entrée)

Description d'un VIA 6522

Pour suivre le détail de ce paragraphe, il est conseillé de se référer à la carte mémoire (adresses I/O bloc 0).

1.Les ports parallèles :

2 registres PORT B(VIA-0) et PORT A(VIA-F) représentent l'état réel des 2*8 fils d'entrées/sorties notés PB0 à PB7 et PA0 à PA7. Chacun de ces fils est soit une entrée, soit une sortie suivant l'état des registres de direction. Pour chaque bit d'un port, si le bit de direction vaut 0, il s'agit d'un fil d'entrée. Si c'est 1, c'est un fil de sortie.

Nous appelerons VIA-0 le premier registre d'un VIA, que ce soit le 1 ou le 2, dans le cours de ce paragraphe. Pour les adresses réelles, voir la carte-mémoire.

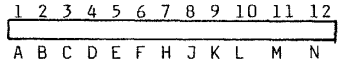
VIA-0 : PB : port B
VIA-2 : DDRB : direction port B (0=entrée, 1=sortie)
VIA-3 : DDRA : direction port A (idem)
VIA-F : PA : port A

(Pour créer un octet à partir de la valeur de chacun de ses bits, voir l'annexe : "notation hexadécimale"). Voici par exemple la méthode à suivre pour programmer le port B avec les bits 0 à 3 en sortie et les bits 4 à 7 en entrée .Les valeurs des bits 4 à 7 seront lues dans les variables B4 à B7. Les sorties bits 0 à 3 seront remises à l'état 1. L'exemple concerne le VIA1:

```
10   W = 37136                          :REM adresse VIA 1
20   POKE W+2,15                        :REM DDRB
30   A=PEEK(W)                          :REM PB
40   B4=(A AND 16)  /  16               :REM lecture du bit 4
50   B5=(A AND 32)  /  32               :REM bit 5
60   B6=(A AND 64)  /  64               :REM bit 6
70   B7=(A AND 128) / 128               :REM bit 7
80   POKE W , PEEK (W) AND 240 OR 15 :REM mise à 1 des  bits 0 à 3.
```

L'état 1 en sortie comme en entrée signifie : tension entre 2.4 et 5 volts.
L'état 0 signifie : tension entre 0 et 1 volt.

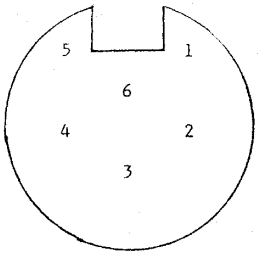
Connecteur du port parallèle "USER'S PORT" : Vue de derrière



Broche no	fonction	Connection VIA
1	masse	-
2	+5V	100 mA max.
3	RESET(normal.=1)	-
4	joy 0	VIA 1 port A bit 2
5	joy 1	VIA 1 port A bit 3
6	joy 2	VIA 1 port A bit 4
7	photostyle/bouton de tir	VIA 1 port A bit 5
8	interrupteur cassette	VIA 1 port A bit 6
9	entrée série ATN	broche 3 du connecteur IEEE série
10	9 volts alternatif	100mA
11	9 volts alternatif	maxi.
12	masse	-
A	masse	-
B	CB 1	VIA 1 : CB 1
C	PB 0	VIA 1, port B, bits 0 à 7
D	PB 1	-
E	PB 2	-
F	PB 3	-
H	PB 4	-
J	PB 5	-
K	PB 6	-
L	PB 7	-
M	CB 2	VIA 1 : CB 2
N	masse	-

A noter : Pour lire l'état de la broche 3 de l'IEEE série, on peut relier les broches 7 et H du port utilisateur et tester l'entrée H (PB4).

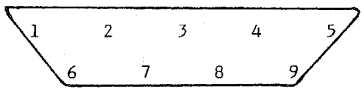
Vue arrière du connecteur IEEE SERIE :



Broche	Fonction	Connection
1	service	entrée VIA 2-CB 1
2	masse	-
3	attention	sortie VIA 1-PA 7
4	horloge	entrée VIA 1-PA 0 sortie VIA 2-CA 2
5	donnée	entrée VIA 1-PA 1 sortie VIA 2-CB 2
6	-	-

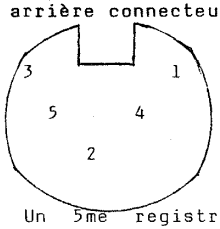
A noter : la broche 3 peut être lue moyennant une connexion sur le port utilisateur (voir ci-dessus).

Vue de droite du connecteur de jeu :



Broche	Fonction	Connection
1	joy 0	VIA 1-PA 2
2	joy 1	VIA 1-PA 3
3	joy 2	VIA 1-PA 4
4	joy 3	VIA 2-PB 7
5	pot Y	pot Y-6561
6	photostyle/ bouton tir	photostyle 6561 VIA 1-PA 5
7	+5V	max 100 mA
8	masse	-
9	pot X	pot X-6561

Vue arrière connecteur vidéo :



Broche	Fonction
1	+6V-10mA max
2	masse
3	sortie son
4	sortie image niveau faible.
5	sortie image niveau fort

Un 5me registre concerne l'usage du port A : le registre VIA-1 a une fonction semblable au registre VIA-F : c'est l'image des 8 bits du port A. Mais en utilisant le port A en entrée avec le registre VIA-1, on dispose d'une fonction supplémentaire, le verrouillage. Un dispositif externe peut déposer une donnée à 8 bits sur le port A et l'y verrouiller par une impulsion sur le fil CA 1, ce qui met à un le drapeau CA 1(voir plus loin le contrôle des interruptions). La simple lecture du registre VIA-1 par le 6502 remet à zéro

ce drapeau : ceci est très rapide et pratique pour des communications en mode parallèle (avec, par exemple, le dialogue type 'Centronics'). Malheureusement, dans le VIC, le fil CA1 n'est pas accessible.

2. La minuterie-compteur 1

Il y a dans chaque VIA deux minuteries T1 et T2 qui sont en fait des décompteurs.

La minuterie T1 : Est contrôlée par 4 registres : deux pour la valeur du compteur et deux pour le verrou qui permet de lire ou d'écrire une valeur de décompte sans interrompre le comptage. La valeur se lit dans les registres VIA-4 et VIA-5 (partie basse, partie haute). Le décomptage peut donc décompter à partir de 65535 au maximum. Les entrées horloge de comptage des 6522 du VIC sont reliées à l'horloge du microprocesseur à 1.108 Mhz. Le décomptage maximum a donc une durée de 65535/1.108200 = 0,059 secondes. Donc nous avons une minuterie capable de décompter pendant des durées comprises entre 1 microseconde et 5 centièmes de secondes soit entre 20 et 1.000.000 de fois par seconde.

Quand le décomptage atteint 0, un drapeau (IFR6) passe à 1 dans le registre VIA-D et une interruption est générée. Dans le VIC, c'est la méthode utilisée pour générer une interruption IRQ 60 fois par seconde avec le VIA-2 : elle déclenche la routine de balayage clavier, comptage horloge et clignotement du curseur.

Le registre de décomptage VIA-4 (partie basse) ne peut qu'être lu et non modifié, ce qui explique que, en pratique, la valeur de décompte est écrite non dans les registres de décompte mais dans un verrou temporaire en VIA-6 et VIA-7. On écrit d'abord la partie basse, puis la partie haute, ce qui provoque le transfert simultané du verrou dans le compteur. De plus, ceci a pour effet de remettre à 0 le drapeau IFR6 qui indique la fin de décomptage de la minuterie 1.

Exemple : la minuterie du VIA-2 génère une interruption tous les 1/60me de seconde . Si on ne se sert pas de l'horloge (qui se déréglerait), on peut augmenter le nombre d'interruptions par seconde : les déplacements et clignotements du curseur s'accélèrent avec le programme suivant qui génère 100 interruptions par seconde:

```
10   W = 37152          : REM VIA 2
20   POKE W+6, 74       : REM verrou (bas)
30   POKE W+7, 43       : REM verrou (haut)
```

La valeur la plus agréable pour obtenir un **curseur rapide** est POKE 37159 , 35 .
Si la valeur descend jusque 2 ou même 1, il ne reste plus de temps au BASIC pour travailler entre deux interruptions et on obtient un comportement du VIC très bizarre. (retour à la normale par STOP-RESTORE).

La valeur courante de la minuterie 1 peut être lue par :

```
10   W = 37152
20   PRINT PEEK (W+4) + 256 * PEEK (W+5)
```

La minuterie 1 peut se programmer en 4 modes différents suivant les valeurs des 2 bits ACR6 et ACR7, bits 6 et 7 du registre VIA-B.
Le mode normal pour le VIA 2 est ACR6 = 1, ACR7 = 0.

ACR6 = 1 signifie : à la fin du décompte recommencer le décompte à l'infini en générant à chaque fois une interruption.(FREE-RUN)
ACR6 = 0 signifie que le décompte ne doit se faire qu'une seule fois (ONE-SHOT).
ACR7 = 0 signifie que le bit 7 du port B ne doit pas être affecté.
ACR7 = 1 signifie que le bit 7 du port B doit changer de signe à chaque fin de décompte.(Attention, PB7 doit être programmé en sortie).

Pour le VIA-2, la seule combinaison intéressante outre celle d'origine est ACR6=1, ACR7=1. Si on a programmé PB7 en sortie, on obtient ainsi des impulsions régulières au rythme des interruptions IRQ (normalement au 1/60me de seconde) sur le fil joy 3 disponible sur le connecteur.

Dans le VIA 1, l'interruption qui est générée est la NMI : elle est utilisée pour l'RS232 et la fréquence d'interruption est alors programmée suivant la vitesse de transmission désirée. Le mode utilisé est alors ACR6=0, ACR7=0

Exemple : pour mettre ACR6 et ACR7 à 1 dans le VIA 2 avec PB7 en sortie, on écrira :

```
10   POKE 37154 , PEEK (37154) OR 128          :REM DDRB-Z bit 7=1
20   POKE 37163 , PEEK (37163) AND 63 OR 192   :REM ACR6 et 7 = 1
```

Il y a deux méthodes pour remettre à 0 le drapeau IFR6 de la minuterie 1 : lire la partie basse du compteur (VIA-4) ou écrire dans la partie haute du verrou (VIA-6).

Le compteur de la minuterie 1 compte toujours un délai supérieur à la valeur programmée. L'excédent est faible : 2 cycles de l'horloge soit 1,8 micro-secondes. Ceci ne doit pas être négligé pour des délais inférieurs à 200 microsecondes où cela représente 1% d'erreur déjà.

3.La minuterie 2 :

Elle est plus simple. Elle n'occupe que deux registres : VIA-9 et VIA-10, parties basse et haute de la valeur de décomptage. Les fonctions sont plus ou moins semblables à celles de la minuterie 1 mais l'usage des registres a été comprimé : le registre VIA-9 est un verrou en écriture donc les deux registres fonctionnent comme les deux verrous VIA-5 et VIA-6 pour la minuterie 1. Mais en lecture, le registre VIA-9 contient toujours la valeur réelle du décomptage-partie basse. Le fait de lire le registre VIA-9 remet à 0 le drapeau IFR5 qui se repositionne à 1 à la fin du décomptage.

La minuterie 2 se programme suivant 2 modes qui dépendent de la valeur du bit 5 du registre VIA-B.

Mode VIA-B,5=0 : Décompteur simple. Le décomptage s'effectue dès l'écriture de la partie haute du compteur en VIA 10. La fin du décomptage provoque une interrruption. Ce mode est employé dans le VIA 1 pour la réception sur l'RS232. Dans le VIA 2, ce mode est utilisé par le système lors des transferts de données en IEEE série et pour les entrées/sorties cassette. En dehors des

périodes d'accès aux périphériques, les deux minuteries 2 sont disponibles pour l'utilisateur.

Mode VIA-B,5=1 : Décompteur d'impulsions : dans ce mode, les impulsions qui décrémentent le contenu de la minuterie 2 ne sont plus celles de l'horloge du 6502, mais celles présentées sur le fil PB6. Le VIC ne se sert pas de cette fonction mais on peut l'utiliser dans le VIA-2 où le port est disponible. Dans le VIA-2, le bit 6 du port B est relié au clavier et il n'y a guère d'intérêt à utiliser ce mode. Le compteur ne détecte que les flancs descendants, c'est-à-dire les transitions de l'état 1 vers l'état 0 du fil PB6. Si on désire compter 1000 impulsions sur le fil PB6, on programme la minuterie 2 avec la valeur 1000 et VIA-B,5 à 1. La fin du décomptage est indiquée par la génération d'une interruption et le passage à 1 du drapeau IFR5 dans le registre VIA-D.

Exemple :
minuterie 2,VIA 1. Pour compter jusque 1000 :

```
10   W = 37136
20   POKE W + 11 , PEEK (W+11) AND 223 :REM mode comptage
30   POKE W +  2 , PEEK (W+ 2) AND 192 :REM PB6=entrée
40   POKE W +  8 , 232                 :REM valeur basse
50   POKE W +  9 ,   3                 :REM valeur haute.
```

La minuterie 2 est utilisée également pour déterminer le rythme de décalage du registre à décalage.

4.Le registre à décalage :

Le registre VIA-A du 6522 est un registre à décalage de 8 bits. Sous le contrôle d'une horloge, les 8 bits sont décalés vers la gauche (bit0 dans bit 1,etc...) et le bit 7 "tombe" dans le bit de sortie **CB 2** si on est en mode sortie. Si on est en mode entrée, le décalage a lieu dans le même sens et c'est dans le bit 0 que tombe la valeur du bit d'entrée CB 2. L'horloge qui synchronise les décalages peut être soit le passage par 0 de la partie basse du décompteur minuterie 2 (2 tops d'horloge sont séparés par 4 à 550 microsecondes environ suivant programmation), soit l'horloge du 6502, soit un signal externe appliqué sur le fil **CB 1**. Avec l'horloge du 6502, la fréquence d'horloge est divisée par deux. Le signal sur CB 1 ne peut excéder la moitié de la vitesse de l'horloge du 6502. Sauf dans le mode automatique, le registre à décalage génère une interruption après 8 décalages, c'est-à-dire après avoir transmis ou reçu un octet en série sur le fil CB 2. Le VIC se sert du registre à décalage du VIA 2 pour envoyer les données en série sur l'interface IEEE et de celui du VIA 1 pour la transmission en RS 232. Les modes de fonctionnement du registre à décalage sont au nombre de 8 mais ne sont pas tous utilisables du fait de l'accès du registre de contrôle par la routine d'interruption clavier. Ces modes sont choisis suivant la valeur des bits 2, 3 et 4 du registre VIA-B.

Soit U le numéro du mode, on programme celui-ci par :

```
10   W = 37136 (ou 37152) (VIA 1 ou VIA 2)
20   POKE W + 11 , PEEK (W+11) AND 227 OR (U*4)
```

Mode U=0 : Registre à décalage hors service.
Entrée de données sur CB 2.

Mode U=1, U=2, U=4 :
Ne pas utiliser.

Mode U=3 : Mode de sortie automatique. Le contenu du registre se décale continument et est envoyé régulièrement sur CB2. En ajoutant au fil CB2 un transistor et un haut-parleur, on dispose d'un générateur sonore supplémentaire pour autant que l'on ait chargé le registre avec un octet différent de $00 ou $FF. Ceci est utilisé dans les PET/CBM pour la sortie sonore.

Mode U=4 : Ne pas utiliser.

Mode U=5 : Sortie sur CB2, l'horloge provenant de la partie basse de la minuterie 2. Dans ce mode, la lecture du registre à décalage remet à 0 le drapeau d'interruption du registre à décalage IFR2 bit 2 du registre VIA-D et provoque le redémarrage du processus de décalage, même si celui-ci n'est pas fini.

Mode U=6 : Idem, mais l'horloge est celle du 6502 divisée par deux.

Mode U=7 : Idem, mais l'horloge est sous le contrôle du fil CB1.

5.Les registres de contrôle. :

Ils sont au nombre de deux : le PCR ou VIA-C contrôle le fonctionnement des fils de dialogue CA1, CA2, CB1 et CB2. Le registre ACR ou VIA-B contrôle les fonctions des minuteries, du registre à décalage et des verrous de port A et B et son emploi a été détaillé ci-dessus.

CA1 Se programme par le bit 0. Si le bit PCR0 vaut 0, les interruptions que provoque CA1 apparaîtront lors des transitions descendantes, lorsque CA1 passe de +5 volts à 0 volt. Si PCR0 vaut 1, les interruptions dues à CA1 dépendront des transitions montantes, lorsque CA1 passe de 0 volt à + 5 volts. Dans le VIA1, CA1 est lié à la touche RESTORE, dans le VIA2, au fil de lecture cassette.
Exemple :
PCR0 à 1

```
10   W = 37136 (ou 37152)      (VIA1 ou VIA2)
20   POKE W + 12 , PEEK (W+12) AND 254 OR 1.
```

CA2 Les bits PCR1, PCR2 et PCR3 du registre VIA-C donnent 8 possibilités de programmation de CA2. Nous appèlerons U le numéro de mode (0<=U<=7).

CA2/U=0 CA2 est un fil d'entrée. L'interruption CA2 survient lors d'une transition descendante de CA2. Dans ce mode, le drapeau d'interruption de CA2 (IFR0) est remis à 0 par un accès lecture ou

écriture au registre VIA-1 (voir ci-dessus l'usage du registre VIA-1).

CA2/U=1 Comme ci-dessus mais les accès à VIA-1 ne modifient pas le drapeau IFRO.

CA2/U=2 Comme U=0 mais on détecte les transitions montantes.

CA2/U=3 Comme U=1 mais on détecte les transitions montantes.

CA2/U=4 CA2 est un fil de sortie normalement à l'état 1 (+5volts). Il passe à l'état 0 lors d'un accès lecture ou écriture et est remis à 1 par une transition active de CA1 (active=montante ou descendante suivant la valeur du bit PCR0).

CA2/U=5 CA2 est un fil de sortie normalement au +5V. Il passe à l'état 0 volt pendant 1,1 microsecondes lors de chaque lecture ou écriture du registre VIA-1.

CA2/U=6 CA2 est un fil de sortie toujours à 0 volt. Ceci est utilisé dans le VIC pour le contrôle du moteur cassette (moteur à l'arrêt) et pour l'horloge de l'IEEE série.

CA2/U=7 Idem mais CA2 toujours à 1.

En pratique, vu l'usage que fait le VIC des 2 port A, seuls les modes U=6 et U=7 présentent une utilité.

Pour modifier PCR1, PCR2 et PCR3:

```
10   W=37136  (ou 37152)      (VIA1 ou VIA2)
20   POKE W + 12 , PEEK (W+12) AND 241 OR (2*U)
```

Pour arrêter le moteur de la cassette :

```
POKE 37148 , PEEK (37148) AND 241 OR 14
```

Pour démarrer le moteur de la cassette :

```
POKE 37148 , PEEK (37148) AND 241 OR 12
```

CB1 est contrôlé par le bit PCR4. Si le bit vaut 0, CA1 est une entrée dont les transitions descendantes génèrent une interruption. S'il vaut 0, c'est l'inverse. De plus, si le registre à décalage est en fonction, CB1 présente une impulsion lors de chaque décalage : c'est l'horloge de décalage. Dans le VIA 1, CB1 est utilisé pour l'entrée RS 232. Dans le VIA 2, CB1 est raccordé à l'entrée **SRQ** : demande de service sur l'IEEE série. Cependant, le VIC de base ne contient aucun logiciel surveillant l'état de cette ligne (Voir le programme "IMPRIMANTE PARALLELE" en annexe).

CB2 est contrôlé par les 3 bits PCR5, PCR6 et PCR7. Il y a donc 8 modes possibles. Soit U le numéro du mode (0<=U<=7). Ces modes sont symétriques de ceux de CA1 pour le port A.

CB2/U=0 Entrée: Le drapeau IFR3 est sensible aux transitions descendantes. Un accès au registre VIA-0 remet IFR3 à 0.

CB2/U=1 Idem mais pas d'influence des accès à VIA-0.

CB2/U=2 Comme U=0 mais transitions montantes.

CB2/U=3 Comme U=1 mais transitions montantes.

CB2/U=4 Sortie : CB2 passe à 0 lors d'une écriture dans le registre VIA-0 et repasse à 1 lors d'une transition active de CB1.

CB2/U=5 Sortie : CB2 passe à 0 pendant 1,1 microsecondes lors de l'écriture dans le registre VIA-0.

CB2/U=6 Sortie, CB2 est toujours à 0 volt.

CB2/U=7 Sortie, CB2 est toujours à +5 Volts.

Dans le VIA 1, seuls ces deux derniers modes sont utilisés, CB2 est la sortie RS232. Dans le VIA 2, CB2 représente la sortie données sur l'IEEE série. De plus CB2 peut être utilisé comme sortie sonore ou générateur de signal rectangulaire sous le contrôle du registre à décalage. (Voir plus haut). Ces deux modes sont utilisés dans le programme "IMPRIMANTE PARALLELE" en annexe

Pour programmer CB2 en mode U :

```
10   W=37136                          (ou 37152)     (VIA 1 ou VIA 2)
20   POKE W+12 , PEEK (W+12) AND 31 OR (U*32)
```

La génération des interruptions par les 6522

Rappelons que le VIA1 génère des **interruptions** NMI et le VIA 2 des interruptions IRQ. Chaque VIA contient deux registres de contrôle liés aux interruptions : VIA-D ou IFR (interrupt flag register) et VIA-E ou IER (interrupt enable register). Le premier contient 7 drapeaux indiquant quelle partie du VIA a généré une interruption. Le second contient 7 drapeaux correspondants qui peuvent être positionnés à 1 par le programmeur pour indiquer que la source d'interruption concernée est autorisée. La remise à 0 des drapeaux de VIA-D s'effectue de différentes façons suivant la source de l'interruption. On peut bien entendu toujours le remettre à 0 explicitement par POKE. Le 8me bit de chacun de ces registres indique si une interruption provient du VIA ou non (VIA-D bit 7) et si le VIA est autorisé à générer des interruptions (VIA-E bit 7). Ceci est surtout utile dans les systèmes où plusieurs VIA peuvent générer le même type d'interruption.

Pour autoriser une interruption mettre à 1 le bit correspondant ainsi que le bit 7 de VIA-E :

soit en 37150 ($911E) pour VIA-1 (NMI)
soit en 37166 ($912E) pour VIA-2 (IRQ)

Pour voir si une interruption a eu lieu, tester le bit correspondant dans VIA-D :

soit en 37149 ($911D) pour VIA-1 (NMI)
soit en 37165 ($912D) pour VIA-2 (IRQ).

Bit		Cause de l'interruption	Si le bit à 1 dans VIA-E, le bit de VIA-D est modifié comme suit :	
no	valeur		Le bit passe à 1 si	Le bit passe à 0 si
0	1	CA2	transition active	lire/écrire registre VIA-1
1	2	CA1	transition active	lire/écrire registre VIA-1
2	4	reg.à décalage	fin de décalage	lire/écrire registre VIA-A
3	8	CB2	transition active	lire/écrire registre VIA-0
4	16	CB1	transition active	lire/écrire registre VIA-0
5	32	minut. 2	fin de décompte	lire VIA-8 ou écrire VIA-9
6	64	minut. 1	fin de décompte	lire VIA-4 ou écrire VIA-7
7	128	*	*	*

L'* indique que le bit 7 est global : dans VIA-E, il autorise toutes les interruptions dont le bit correspondant est à 1. Dans VIA-D, le bit 7 à 1 indique que l'un des 7 autres bits du registre est à 1. les bits IER7 et IFR7 ne sont remis à zéro que par l'écriture explicite d'un 0 dans le bit : il n'y a pas de procédé automatique.

Exemple : La touche **RESTORE** au clavier ne générera une interruption NMI que si les bits IER7 et IER1 sont à 1, ce qui est normalement le cas.

Essayez : supression de la fonction STOP/RESTORE par :

```
POKE 37150, 0
```

Rétablissement de la fonction STOP/RESTORE par :

```
POKE 37150,130          (130=128+2)
```

Les routines d'interruptions sont elles-même modifiables : il faut intercepter les vecteurs en RAM en $0314 et $0315 pour IRQ, en $0318 et $0319 pour NMI. Suivant les conditions du moment, le VIC modifie ces valeurs pour utiliser différentes routines d'interruption en fonction du moment. Pour les NMI en provenance du VIA-1, il y a des routines différentes pour le mode normal (CA1=touche RESTORE), pour les accès RS-232 (minuteries 1 et 2), pour les transferts cassette (minuterie 1).

Pour les IRQ en provenance du VIA 2, il y a des routines pour le mode normal (minuterie 1), pour les accès cassette (minuteries 1 et 2) et pour l'interface IEEE série (minuterie 2).

LES PERIPHERIQUES.

Par l'intermédiaire des 6522 que nous venons de découvrir ,le système communique avec différents périphériques : clavier, cassette, manche de commande, disquettes, imprimantes, etc... L'étude des périphériques nécessite obligatoirement une connaissance de base dans les programmes internes du VIC. Les points qui resteront obscurs après le lecture de ce paragraphe seront éclaircis dans le chapitre 2.

Le clavier.

La routine d'interruption INT passe en revue le **clavier** 60 fois par seconde et le résultat de cette opération est mémorisé dans le tampon de clavier en RAM de 631 à 641 (KBDBUF= $0277 à $0280). La taille maximum de ce tampon est de 10 caractères, mais cette valeur est variable. La taille est rangée en RAM à l'adresse 649 (KBDMAX=$0289). On ne peut mettre en 649 qu'une valeur maximale de 15 , car l'adresse du début de tampon clavier est invariable en 631. Or cette adresse est suivie de 14 autres inoccupées et réservées à cet usage. Si on choisit une taille de 16, l'adresse 646 est éventuellement occupée lors d'une frappe au clavier. Et en 646 est rangée la couleur du caractère actuellement sous le curseur. Modifier cette valeur donne des résultats fort curieux. Au delà de 646, ce sont les pointeurs d'écran et l'octet 649 lui-même qui risquent d'être détruits. Par contre, POKE649,1 autorise une seule touche d'avance, ce qui est parfois un avantage si on ne veut pas laisser la possibilité de frapper plusieurs réponses d'avance. POKE649,0 interdit tout emploi du clavier sauf STOP et RESTORE qui sont testés indépendamment. Voir la description des routines **GETIN, STOP, SCNKEY.**

Pour vider le tampon de clavier avant un input, par exemple, on peut modifier et remettre à zéro le pointeur actif de tampon clavier en RAM page 0 en 198 (KBDPTR=$C6). De plus, ce pointeur peut être utilisé pour simuler des frappes au clavier dans le corps d'un programme. Il suffit d'écrire (par POKE en BASIC, STA en **L.M.**) les caractères dans le tampon, puis le nombre de caractères dans le pointeur. A la prochaine lecture du clavier, soit par INPUT ou GET en BASIC, ou par l'interpréteur BASIC s'il est revenu au mode direct en fin de programme, les caractères seront pris dans le tampon pour être considérés comme des frappes au clavier. On peut ainsi envisager un programme qui se modifie lui-même. Il suffit que le programme écrive une nouvelle ligne BASIC en haut de l'écran ainsi qu'un GOTO 3 lignes plus bas, dépose <HOME> <RETURN> dans le tampon clavier puis exécute l'instruction STOP pour qu'une nouvelle ligne soit créée et que le programme se soit auto-modifié. Attention, cette méthode remet à zéro toutes les variables du programme. (Voir le programme AUTOPRINT en fin de volume).

Le clavier est en fait une matrice de 8*8 interrupteurs reliés aux ports A et B du VIA-2. De plus la touche RESTORE est reliée à la broche CA1 du VIA-1 qui génère à ce moment une interruption NMI. Le balayage du clavier comprend une technique logicielle appelée "N-KEY ROLLOVER" qui rend au programme les codes des touches frappées dans l'ordre de leur enfoncement sans tenir compte du fait que l'une ou l'autre reste enfoncée au moment ou l'on presse la suivante. L'inconvénient est qu'ainsi on n'a pas accès aux tests d'enfoncement de plusieurs touches à la fois. Heureusement le VIC a tout prévu : le balayage

clavier se termine par la rangée de touches qui comprend le STOP. La valeur de décodage de cette rangée est toujours accessible en RAM en 145($91). Cette valeur est remise à jour 60 fois par seconde par la routine **STOP**. Ceci permet le test en simultané de n'importe quelle touche parmi STOP, SHIFT GAUCHE, X, V, N, <, ? ,CUR.DROIT.

Les **touches programmables** ont des valeurs ASCII qui peuvent être testées par programme très simplement.

```
10   ?"POUSSEZ SUR F1"
20   GET A$
30   IF A$<>CHR$(133)GOTO 20
40   ...
```

Le code ASCII de chaque touche de fonction se retrouve dans le tableau de décodage clavier ci-après.

Il est aisé d'attribuer à chaque touche de fonction une série de caractères qui s'écriront comme si on les avait frappés successivement au clavier. Il suffit pour cela d'intercepter la routine d'interruption par son vecteur en RAM, de tester la frappe d'une des 8 touches F1-F2 et de mettre dans le tampon clavier la suite de caractères correspondants. Il est évident que cette méthode convient pour intercepter d'autres touches que celle-la mais ce peut être gênant car dans ce cas ,on supprime la fonction préexistante de la touche concernée.(Voir en annexe le programme **"TOUCHEPROG"**).

On sait que les touches de déplacement de curseur et la barre d'espace ont une fonction de **répétition** automatique. Ceci est dû à la routine d'interruption. On peut par l'intermédiaire de cette routine mettre la répétition automatique sur toutes les touches ou sur aucune.

```
Répétition sur toutes les touches        POKE 650,128
           sur aucune touche             POKE 650,127
           touches curseur+espace        POKE 650,0
```

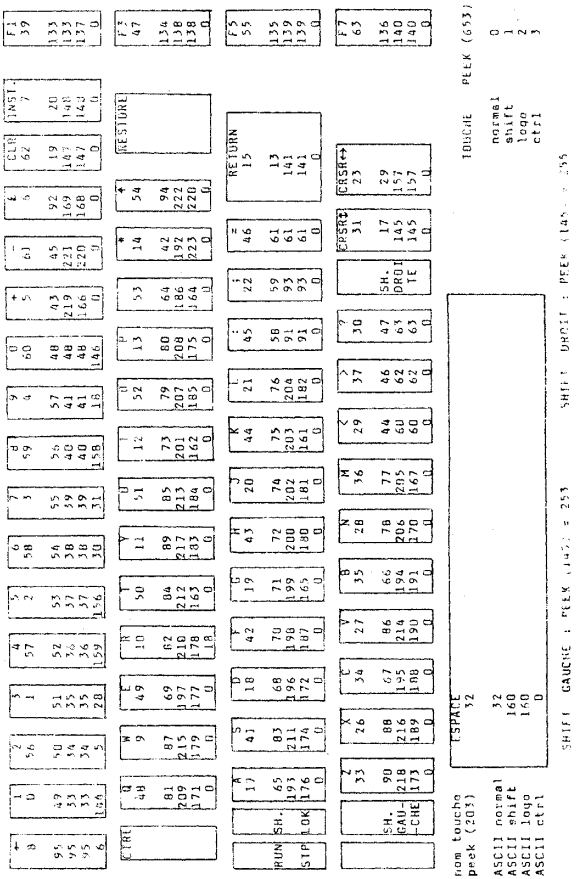
Le taux de répétition de touches est de 15 par seconde, après un délai de 1/3 de seconde, ceci pour éviter des frappes doubles par erreur en fonctionnement normal. Si toutefois on obtient de temps à autre des doubles frappes on peut allonger ce délai par :

```
POKE 651,32    (valeur par défaut = 4)
```

La touche **STOP** est testée au début de la routine d'interruption. Il est donc possible de modifier le vecteur en RAM de cette routine de façon à ce qu'il pointe non plus sur le début de la routine où on teste le STOP et où on incrémente l'horloge, mais vers la seconde partie de la routine : lecture clavier, clignotement du curseur, etc... :

```
Supprime la touche STOP        POKE 788,194
Réautorise la touche STOP      POKE 788,171
(Attention ceci arrête l'horloge).
```

La touche **RESTORE** n'est pas reliée au décodage clavier principal mais bien à la masse et au fil CA1 du VIA 1. Si - ce qui est le cas à l'allumage du VIC- le VIA 1 est programmé pour cela, cette touche génère une interruption. or l'interruption du VIA 1 est reliée à l'interruption NMI du 6502. Il n'est donc pas possible de masquer cette interruption dans le microprocesseur. Par contre, il est possible de reprogrammer le VIA de façon à ce qu'il ne génère pas d'interruption lors de la réception d'un signal 0 sur CA 1.
Ceci s'effectue comme suit :


RETOUR
RETURN
CRSR↔
CRSR↕
SH. DROITE
ESPACE
touche PEEK (653)
normal
shift
logo
ctrl
ASCII normal
ASCII shift
ASCII logo
ASCII ctrl
nom touche
peek (203)
SHIFT GAUCHE : PEEK (
SHIFT DROIT : PEEK (

Supprime la touche RESTORE POKE 37150,2
Réautorise la touche RESTORE POKE 37150,130

L'adresse 37150 ($911E) est le registre d'autorisation d'interruption du VIA-1. (Voir pour d'autres détails le paragraphe concernant les 6522).

La touche **LOGO** permet l'impression de divers graphiques (voir tableau de décodage clavier). Elle permet aussi, en combinaison avec SHIFT, le passage au jeu de caractères majuscules/minuscules. Cette dernière fonction est subordonnée à la présence d'un 0 dans le drapeau en mémoire RAM (adresse 657-$0291).

SHIFT/LOGO est autorisé par : POKE657,0 ou <RUN> et <RESTORE>.
SHIFT/LOGO est interdit par : POKE657,128
mais l'interpréteur BASIC gère cette adresse en fonction de la réception des caractères 8 ou 9.

SHIFT/LOGO est autorisé par : PRINT CHR$(8)
SHIFT/LOGO est interdit par : PRINT CHR$(9)

Dans le VIC, la méthode particulière de décodage du clavier matricé fait que certaines combinaisons d'enfoncements multiples de touches sont parfois reconnues comme enfoncement d'une seule touche. En règle générale, ce n'est pas un phénomène gênant. La seule application connue de cette particularité est le truc suivant qui remplace le mot-clé BASIC RUN.

Pour remplacer RUN <RETURN> :

1. enfoncer la touche 'RUN'
2. enfoncer la touche 'SHIFT GAUCHE'
3. enfoncer la touche 2 (ou 4,ou 6)
4. relâcher la touche 2.
5. relâcher les touches 'RUN' et 'SHIFT'.

Ceci permet de ne frapper que 3 touches au lieu de 4. L'intérêt est plus démonstratif qu'utile car on peut aussi employer la combinaison R, SHIFT-U, <RETURN>.

Le lecteur de cassettes.

Le connecteur plat à 12 contacts à l'arrière du VIC est certainement le plus vital de tous : il permet de sauver et relire vos programmes.

connecteur cassette, vue d'arrière

Les connections sont toutes reliées 2 à 2.

Broche	Fonction
A-1	masse
B-2	+5 volts
C-3	moteur cassette 0 ou 6 volts
D-4	fil de lecture (READ)
E-5	fil d'écriture (WRITE)
F-6	détection de touche (SENSE)

Le lecteur de cassettes spécial de COMMODORE est très fiable. Il existe cependant diverse possibilités de connecter des lecteurs ordinaires à un VIC. Pour notre part, nous n'avons jamais eu l'occasion de tester un interface pour lecteur ordinaire qui donne vraiment satisfaction.

Pour utiliser le lecteur de cassettes de manière fiable, il faut que la plupart des pointeur en page 0 et 3 soient corrects, ce qui n'est pas toujours le cas après l'exécution d'un programme. Il faut donc toujours sauver un programme **avant** de l'essayer, surtout s'il contient des POKE, SYS et autres USR. Si le programme refuse de se sauver pour cause de 'OUT OF MEMORY", il faut essayer de le sauver sans nom de programme.

Le lecteur de cassettes est utilisé à deux fins : sauver des programmes et sauver des données. Le KERNAL utilise pour cela deux formats différents : les programmes sont sauvés en un seul bloc tandis que les données sont sauvées par petits blocs de 192 caractères, pour laisser le temps au lecteur de cassettes de s'arrêter et de redémarrer entre chaque bloc. En mode SAVE, le KERNAL crée un bloc de données avec les valeurs de tous les octets compris entre les deux adresses de début et de fin de BASIC, en écrivant les adresses de début et de fin de zone dans les pointeurs en 43, 44 ($2B, $2C) début de BASIC et en 45, 46 ($2D, $2E) fin de BASIC mais il est nécessaire de restaurer ensuite les valeurs correctes de ces pointeurs, ceci ne pouvant se faire par une sauvegarde des valeurs dans des variables BASIC, car celles-ci sont inaccessibles quand les adresses 45 et 46 sont modifiées. En effet, fin de BASIC et début de variable représentent une seule et même adresse. On sauvegarde donc les valeurs PEEK(43) à PEEK(46) en les écrivant à des adresses précises par POKE. Pour cet usage, on peut utiliser les adresses 251 à 254 par exemple.

La création de fichiers de données est très simple. Consultez le chapitre 'Programmes internes' aux instructions OPEN, PRINT£ et INPUT£. On trouvera des exemples de fichiers de données avec les programmes SUPERLIST et PROGDATA.

Lors de la création d'un fichier de données, le tampon de cassettes de 192 octets se remplit progressivement. Il commence en 828 ($033C). Le pointeur qui contient l'adresse du tampon de cassette est en 178 et 179. Le nombre de caractères présents dans le tampon (en pratique, le nombre -1, car on compte à partir de 0 inclus) est gardé en 166 ($A6). On voit ainsi une solution pour sauver 192 octets consécutifs (une image graphique, un générateur de caractère,...) : on écrit en 178 et 179 les parties haute et basse de l'adresse de début de zone à sauver. Ensuite un POKE166,191 fait croire au KERNAL que le tampon est plein et il l'écrit sur la cassette. Il faut au préalable avoir ouvert le fichier par, par exemple : OPEN 1,1,1,"DATA". La grande fiabilité des cassettes enregistrées avec le VIC dépend de la complexité du logiciel de gestion de la cassette. Tout d'abord les bits sont enregistrés en modulation de ° fréquence à l'aide de deux fréquences (1488 Hz et 2840 Hz) bien centrées dans la bande passante des cassettes audio ordinaires. Ensuite, il y a au début de chaque enregistrement un bloc de titre (HEADER) d'une dizaine de secondes dont le but est de synchroniser la lecture : le VIC, en lecture, se base sur la fréquence des bits du bloc de tête pour calculer la correction de vitesse due aux variations de la vitesse de défilement de la bande. De plus, chaque bloc de données (ou chaque programme) est enregistré deux fois, et à chaque fois les octets sont sauvés par blocs de 8 suivis d'un neuvième octet de contrôle, représentant la somme des 8 octets Modulo 256 (c'est-à-dire en laissant tomber les bits de poids supérieur ou égal à 256). Dernier niveau de

contrôle, chaque octet est enregistré sur 9 bits et non 8, le neuvième étant un bit de parité tel que la somme des 9 bits soit toujours impaire. Le résultat est lent mais fiable. En effet, grâce à ces techniques, le VIC s'autorise un certain nombre d'erreurs (en théorie 32 maxi) qu'il peut non seulement détecter mais aussi corriger : une fois l'octet faux repéré grâce aux redondances du 9me bit et du 9me octet, la valeur correcte est lue dans le second bloc de données identique au premier et pour autant qu'à la seconde lecture il n'y ait pas d'erreur dans cet octet, l'erreur disparaît.

On trouvera dans les routines du KERNAL les morceaux de logiciel nécessaires à lire et à écrire des données et des programmes sur cassette.

Routines principales :

LOAD	$F542	WHEAD	$F8E3
SAVE	$F675	TAPE	$F8F4
FHEAD	$F7AF	SETTO	$F95D
SENSE	$F8AB	CREA	$F98E
RECPL	$F8B7	BYHAND	$FABD
RHEAD	$F8C0	WCASS	$FBEA
RLOAD	$F8C9		

Mais dans la plupart des cas, il est plus aisé de se servir des équivalents des classiques OPEN, PRINT£, INPUT£ et GET£ que sont les routines :

OPEN	SETLFS	$FFBA
	SETNAM	$FFBD
	OPEN	$FFC0
PRINT£	CHKOUT	$FFC9
	CHROUT	$FFD2
GET£	CKKIN	$FFC6
	CHRIN	$FFCF
CLOSE	CLOSE	$FFC3

Pour plus de détails, voir les rubriques concernant ces différentes routines dans le chapitre "Programmes internes" ainsi que au paragraphe IEEE série.

Etant donné la difficulté de duplication des fichiers de données sur cassettes (on ne peut pas faire simplement LOAD puis SAVE), voici un truc qui fonctionne avec la plupart des VIC : on raccorde ensemble deux lecteurs de cassettes COMMODORE en croisant les fils lecture et écriture et en les alimentant tous deux avec une source de tension 6 Volts pour les moteurs, le + 5 Volts pouvant être pris sur le connecteur classique du VIC. Attention, il ne faut pas alimenter les moteurs des 2 cassettophones avec le VIC : le transformateur d'alimentation n'apprécie pas. De plus, ce genre de copie "inintelligente" dégrade la qualité du signal et une copie de copie de copie...peut devenir inintelligible pour le VIC.

L'interface IEEE série.

Le standard IEEE-488 est un mode de dialogue sophistiqué.Inventé à l'origine par HEWLET-PACKARD pour l'instrumentation sous le vocable HP-IP, il est maintenant très utilisé car il permet des transfert entre un nombre variable d'appareils (16 maximum) à des vitesses pouvant atteindre le million d'octets par seconde. COMMODORE a adopté dès les premiers P.E.T. ce standard de communication pour relier ordinateur, imprimante, disquette, etc. Une version série plus lente mais bien moins onéreuse a été choisie pour le VIC. Le connecteur est détaillé plus haut à la rubrique "les 6522". Le VIC utilise réellement 4 des 6 fils de ce connecteur : une masse qui sert de référence, un fil de données, un fil d'horloge pour synchroniser celles-ci et un fil 'ATTENTION" indiquant si la donnée transmise est une commande du VIC à un de ses périphériques ou une simple donnée. Horloge et données étant bidirectionnelles, le VIC peut donc dialoguer effectivement avec ses périphériques.

Les transferts de données sont toujours contrôlés par le VIC. Les périphériques, eux, peuvent, à un moment donné appartenir à une seule des deux catégories suivantes :

ECOUTEUR (LISTENER) : le VIC envoie des données
PARLEUR (TALKER) : le périphérique envoie des données.

Celle de ces deux fonctions qui a lieu est déterminée par une commande spéciale envoyée par le VIC.

L'emploi des périphériques en BASIC est simple et n'appelle guère de commentaires. En langage machine par contre, le sujet devient vite complexe et l'on pourrait consacrer un ouvrage entier aux transferts de données sur IEEE. Ceci a déjà été fait. Voir dans la bibliographie l'ouvrage 'PET AND THE IEEE-488", excellent ouvrage clair et précis qui contient tout le détail des dialogues IEEE.

Comme pour le lecteur de cassettes, de nombreuses informations sont disponibles dans le chapitre 'Programmes internes' aux instructions OPEN, PRINT£ et INPUT£.

Du point de vue de l'électronicien, on remarque que les broches ATN, CLK (horloge) et DATA (données) sont bi-directionnelles. Les signaux sont séparés électroniquement à l'entrée du VIC, si bien que l'entrée et la sortie n'ont pas lieu par le même bit de périphérique. Par exemple, l'entrée ATN n'est raccordée qu'à une broche du "USER'S PORT" et n'est pas accessible directe-

ment. Rappelons à quoi sont liées les fonctions de l'IEEE série.

Entrée horloge	VIA1-F bit 0	(port A)
Sortie horloge	VIA2-C bits 1 à 3	(CA2)
Entrée données	VIA1-F bit 1	(port A)
Sortie données	VIA2-C bits 5 à 7	(CB2)
Entrée ATN	U-port broche 9	(le VIC ne se sert pas de cette fonction)
Sortie ATN	VIA1-F bit 7	(port A)
Entrée SRQ	VIA2-D bit 4	(CB1 : le VIC ne se sert pas de cette fonction)

Du point de vue du programmeur, il est quasi-impensable de se servir de l'IEEE série en direct. Les routines KERNAL seront toujours employées que ce soit en langage machine ou en BASIC. L'instruction OPEN se décrit comme suit:

OPEN A, B, C,"DATA"

où la chaîne de caractères est le nom du fichier (indispensable pour le lecteur de disques. A est un numéro de fichier entre 1 et 255 qui n'est utilisé que par le VIC : il ne "sort" pas de la machine. B est le numéro du périphérique. Si 4<=B<=31, le périphérique est supposé se trouver rattaché à l'IEEE série. S'il n'y est pas, après 256 microsecondes de test, le VIC affiche "DEVICE NOT PRESENT ERROR" et interrompt le dialogue. L'adresse secondaire qui n'a de signification que pour le périphérique concerné. Cette adresse lui est transmise et c'est au microprocesseur du périphérique que revient la tâche de décoder cette adresse et d'en déduire la tâche à exécuter (0<=C<=31).Le nom de fichier est transmis au périphérique comme un PRINT£ le ferait. Dans le cas des lecteurs de disquettes, le nom de fichier est interprété comme tel. Pour les autres périphériques IEEE, il s'agit d'une donnée comme une autre. (A noter, le nom de fichier de deux caractères possède aussi une signification spéciale en RS-232, adresse 2 mais ce n'est pas un périphérique IEEE).

Les routines que l'on utilise en IEEE sont les suivantes :

OPEN 1,4,0	ou	SETLFS	$FFBA	prépare les adresses 1, 4 et 0.
		SETNAM	$FFBD	prépare le nom de fichier.
		OPEN	$FFC0	ouvre le fichier.
PRINT£1,A$	ou	CHKOUT	$FFC9	canal de sortie lié au fichier.
		CHKOUT	$FFD2	envoie l'octet.
		CLRCHN	$FFCC	referme le canal de sortie.
GET£1, A$	ou	CKKIN	$FFC6	canal d'entrée lié au fichier.
		CHRIN	$FFCF	reçoit l'octet.
		CLRCHN	$FFCC	referme le canal d'entrée.
CLOSE 1	ou	CLOSE	$FFC3	referme le fichier.

Si les commandes BASIC ci-dessus sont assez évidentes à comprendre, il n'en est pas nécessairement de même pour les routines langage machine équivalentes. Explicitons les maintenant : le KERNAL considère que le VIC possède deux canaux d'accès; normalement c'est le clavier et l'écran. On lit ou écrit des caractères sur ces deux canaux par les routines **CHRIN** ($FFCF) et **CHROUT** ($FFD2).

Le KERNAL permet d'aiguiller ces deux routines vers d'autres périphériques. Pour ce faire, il faut d'abord déclarer les caractéristiques du périphérique, ce qui est réalisé par OPEN. La routine OPEN associe un nombre arbitraire, le numéro de fichier, à toute une série de caractéristiques du périphérique concerné. Les routines SETLFS et SETNAM doivent précéder la routine OPEN car leur but est de déposer à des adresses connues les caractéristiques du périphérique. Donc SETLFS, SETNAM, OPEN associent le numéro de fichier et les caractéristiques en question.

Ensuite, pour envoyer une série d'octets vers le périphérique, on bascule l'aiguillage du canal de sortie vers le fichier défini dans l'OPEN. C'est CHROUT qui bascule l'aiguillage. L'usage en est simple : on met dans le registre X le numéro de fichier et on appelle CHKOUT. Dans l'exemple ci-dessus, on aurait : LDX£1 ; JSR CHKOUT .

L'aiguillage étant réalisé, on envoie tous les octets que l'on désire par la routine classique CHROUT. L'aiguillage est ensuite refermé grâce à la routine CLRCHN qui rétablit les entrées/sorties vers le clavier et l'écran.

De même, pour entrer un caractère depuis un périphérique, on bascule l'aiguillage d'entrée par LDX£ n. de fichier; JSR CKKIN. Ensuite on lit le nombre d'octets que l'on désire par CHRIN et on rétablit l'aiguillage par CLRCHN.

Quand les accès au périphérique sont terminés, on referme le fichier par CLOSE, c'est-à-dire qu'on l'enlève d'une table située en mémoire les caractéristiques du périphérique associé à ce fichier.

Le VIC n'accède donc bien qu'à un seul périphérique à la fois, bien que plusieurs fichiers puissent être ouverts simultanément. Fichier ouvert signifie uniquement que les caractéristiques du périphérique sont disponibles dans une table et qu'un accès à ce fichier ne nécessite que l'aiguillage, le transfert et le second aiguillage.

Il est bien entendu possible d'accéder au bus IEEE série par des routines d'accès de plus bas niveau telles que TALK, UNTLK, LISTEN, UNLSTN, ACPTR et CIOUT (voir carte mémoire entre $FF8A et $FFF3). Mais ceci est réservé aux spécialistes de l'IEEE qui se réfèreront à l'ouvrage précédemment cité. Ils trouveront au chapitre 'Programmes internes.' les renseignements nécessaires pour la mise en oeuvre de ces diverses routines.

Une chose très importante lors des transferts de données sur le bus IEEE série est le contrôle des erreurs de transfert. A chaque accès au bus, le KERNAL remet à jour la variable **ST-STATUS** en $90. Il est de bonne pratique de tester cette valeur après chaque routine IEEE. La façon la plus simple est

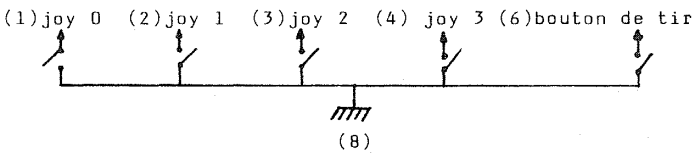
d'utiliser la routine READST en $FFB7. Les bits à 1 indiquent le type d'erreur, soit pour l'IEEE :

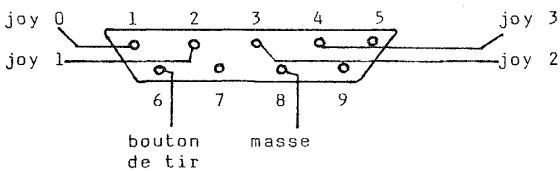
bit 0 : pas de réponse du périphérique en écriture.
bit 1 : pas de réponse du périphérique en lecture.
bit 6 : dernier octet reçu.
bit 7 : "DEVICE NOT PRESENT".

LE MANCHE DE COMMANDE.

Le connecteur de jeu décrit ici plus haut possède un schéma électrique fort simple : de simples interrupteurs reliés au connecteur. Les manches de commande ATARI ou COMMODORE sont identiques mais pas très agréables à manipuler. La solution idéale consiste à récupérer un manche de commande agréable d'emploi et à le recâbler de la bonne façon.

Le schéma du manche :





Bouton de tir =VIA1-F bit 5
joy 0 = haut =VIA1-F bit 2
joy 1 = bas =VIA1-F bit 3
joy 2 =gauche =VIA1-F bit 4
joy 3 = droit =VIA2-0 bit 7

Pour tester les 5 interrupteurs ci-dessus il suffit de lire 5 bits correspondants dans les ports parallèles du VIA 1 et du VIA 2. On peut voir dans la carte-mémoire que l'interrupteur JOY 3 (droite) est connecté au port B du VIA2, qui est également utilisé par la routine de lecture du clavier. A certains moments donc, le même bit devrait avoir deux usages différents. Pour éviter ce genre de conflit, on supprime la lecture d'une rangée de clavier,

(celle qui comprend les chiffres 0, 2,4...) pour se consacrer à la lecture du manche de commande. Il convient de minimiser le temps pendant lequel ceci arrive et également de ne pas arrêter le programme par STOP, car on risquerait de se trouver devant un clavier dont certaines touches ne fonctionnent plus (effectuer STOP-RESTORE). On peut tout d'abord tester le manche à l'aide de 5 variables qui contiennent 0 si le bouton correspondant est enfoncé.

```
Exemple : 10   JH = PEEK (37151)AND 4   : rem haut
          20   JB = PEEK (37151)AND 8   : rem bas
          30   JG = PEEK (37151)AND 16  : rem gauche
          40   JT = PEEK (37151)AND 32  : rem bouton
          50   POKE 37154,127           : VIA 2-0 bit 7 = entrée
          60   JD = PEEK(37152)AND 128: rem droite
          70   POKE 37154,255           : VIA 2-0 bit 7 = sortie
          80   PRINT"(CLR)"JH,JB,JG,JD,JT
          90   RUN
```

On notera dans l'exemple ci-dessus les lignes 50 et 70 qui modifient le registre de direction du port B du VIA 2. Le décodage clavier est donc partiellement incorrect durant l'intervalle de temps entre ces deux lignes.

On trouvera en annexe le programme "MANCHE de commande" qui est un sous-programme de lecture du manche. En sortie de ce programme, les variables T, H et V sont mises à jour en fonction du manche. V indique les déplacement verticaux et vaut -1 si on va vers le haut, +1 si on va vers le bas, 0 si on est immobile. De même, H indique que l'on va à gauche (-1), à droite (+1) ou que l'on est immobile (0). T vaut 1 si le bouton de tir est enfoncé.

En insérant cette routine à la place du test clavier des programmes comme **LORES** par exemple, on voit qu'il ne manque pour faire un déplacement XY que les instructions X=X+H et Y=Y+V. Si on désire obtenir une action unique et non répétée du bouton de tir, il suffit de tester non seulement l'enfoncement du bouton mais aussi son relâchement :

```
10   IF PEEK (37151) AND 32 - 32 THEN PRINT "on a tiré"
20   IF PEEK (37151) AND 32 - 32 GOTO 20
30   PRINT "on ne tire pas" : GOTO 10
```

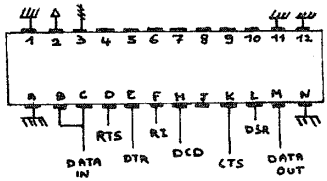
L'INTERFACE SERIE RS-232

Un des avantages majeurs du VIC par rapport à ses concurrents d'un prix comparable est sa capacité de communication. L'interface RS-232 permet la communication d'informations avec d'autres ordinateurs , des modems téléphoniques, des imprimantes, etc. RS-232 est le nom d'un standard de communication série bi-directionnel, dont plusieurs variantes sont utilisées couramment. Le VIC peut travailler suivant deux de ces variantes, la variante "3 fils" et la variante "X-line", cette dernière permettant des dialogues plus élaborés entre deux ordinateurs. Les entrées/sorties de l'interface RS-232 sont en fait les lignes de l'interface parallèle "USER'S PORT". Lorsque l'on utilise l'RS-232 , chacune des lignes du port B du VIA1 est allouée à une fonction bien précise (voir tableau ci-dessous).

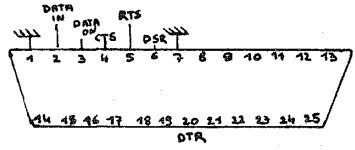
Le standard RS-232 utilise un connecteur DB-25 qui n'est malheureusement pas présent d'origine sur le VIC. De plus, les niveaux logiques des signaux sont à la sortie du VIC au standard TTL (1 = 5V, 0 = 0v) tandis qu'en RS-232 les niveaux sont différents (1 = MARK = -12v à -3v, 0 = SPACE = +3v à +12v). Il est donc nécessaire pour se conformer à la norme d'installer un petit convertisseur de niveaux logiques muni d'une prise DB25.

Un des avantages principaux du VIC employé comme terminal RS-232 est son mode de dialogue entre l'interface série et le programme Basic de l'utilisateur du VIC. Deux raisons en sont la cause : L'interface série est réalisé dans le VIC par un procédé logiciel à 90% ce qui a permis de désynchroniser le dialogue entre BASIC et RS-232 du dialogue entre RS-232 et le monde extérieur.

Connecteur U.P vue arrière du VIC



Prise D.B. Vue de face



DATA OUT : donnée envoyée
DATA IN : donnée reçue
RTS : request to send : prêt à envoyer
DSR : data set ready : donnée prête (OUT)
DCD : data carrier detect : porteuse détectée
DTR : data terminal ready : prêt à recevoir une donnée
RI : ring indicator : plus de papier à l'imprimante.

La raison première de cette implantation logicielle de l'interface série étant le coût minimum du système, COMMODORE a réalisé une simulation complète du circuit 6551 actuellement fabriqué par MOS TECHNOLOGY, la filiale "circuits intégrés" de COMMODORE. Le circuit d'interface série 6551 possède des registres de contrôle adressables. Le logiciel de communication simule non seulement ces registres mais, de plus, il contrôle deux tampons de 256 caractères, un pour l'émission, l'autre pour la réception.

Ce sont ces tampons qui offrent au VIC une souplesse inconnue de bien d'autres micros, même dix fois plus chers. Ils permettent en effet à un programme BASIC d'envoyer une chaîne de caractères instantanément, sans se soucier de transmettre chaque caractère indépendamment. Bien plus, en réception, le Basic peut recevoir une chaîne de caractères en une seule instruction INPUT£2. Sans le tampon de réception, on devrait utiliser des instructions GET£2 dans une boucle avec l'inconvénient énorme de risquer de rater des caractères, car la vitesse du BASIC est, en général, bien insuffisante pour ce genre de tâches. Même à la vitesse classique de 300 bauds, (celle utilisée par les modems téléphoniques), l'instruction GET£2 serait trop lente, tandis que avec INPUT£2, on arrive à procéder sans problème à un dialogue normal entre deux ordinateurs.

COMMENT SE SERVIR DE L'RS-232 EN BASIC?

1.Ouvrir le canal de communication

```
OPEN2,2,0,CHR$(CTL)+CHR$(CMD)
```

2. Recevoir un caractère ou une chaîne

```
GET£2,A$
INPUT£2,A$
```

3. Emettre un caractère ou une chaîne

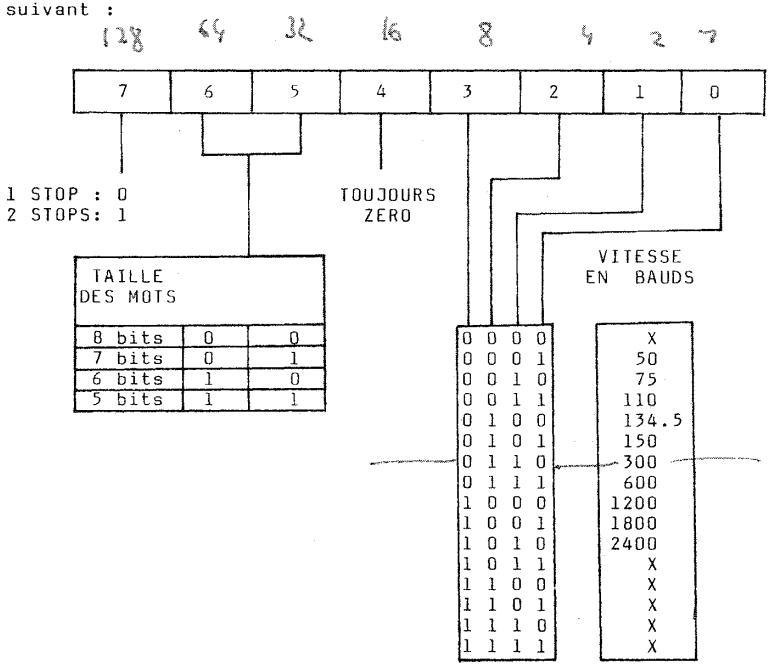
```
PRINT£2,A$
(ou CMD2:LIST)
```

4. Refermer le canal de communication

```
CLOSE2
(ou PRINT£2 : CLOSE 2)
```

A l'ouverture du canal de communication, on précise le numéro de fichier (ici 2), le numéro de périphérique (toujours 2 pour l'RS232), l'adresse secondaire, (toujours 0 pour l'RS232), la valeur à inscrire dans le registre de contrôle (CTL), la valeur à inscrire dans le registre de commande (CMD).

La valeur de l'octet CHR$(CTL) se détermine en binaire grâce au tableau suivant :



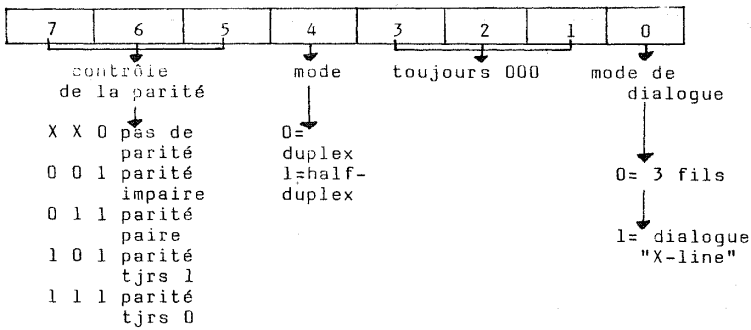
La vitesse s'exprime en nombre de bits/seconde ou BAUDS. La longueur des mots transmis est usuellement de 8 bits, parfois de 7 bits, rarement 5 ou 6 bits; dans ces derniers cas, les bits non transmis sont les bits de poids les plus forts.

Chaque mot transmis en série est constitué de 1 bit de START au niveau 1, les bits du mot à transmettre en commençant par le bit 0, éventuellement un bit de parité et un ou deux bits de STOP. Usuellement un bit de STOP est suffisant, mais parfois l'ordinateur (ou l'imprimante) relié à l'RS232 n'a pas le temps entre deux réceptions de caractères de procéder à diverses opérations internes. Un bit de STOP supplémentaire est à ce moment nécessaire pour allonger le temps mort entre deux caractères.

Par exemple, pour une transmission par MODEM, à 300 bauds, 8 bits+ 2stops, on emploiera comme premier octet de contrôle, l'octet :

10000110 soit CHR$(128+4+2)=CHR$(134)

La valeur de l'octet CHR$(CMD) s'exprime également en binaire grâce au tableau suivant :



Cet octet contrôle les modes de transmission. Le bit de parité est un bit de contrôle qui introduit dans la transmission une redondance c'est-à-dire un élément superflu mais qui permet le contrôle de la validité des données transmises. Dans le cas d'une parité paire, le bit de parité a une valeur telle que la somme de tous les bits transmis (bits de donnée + bit de parité) soit paire. Usuellement, mais pas nécessairement, il est conseillé de transmettre un bit de parité paire.

Le mode DUPLEX signifie que l'on transmet et que l'on reçoit simultanément des octets. En mode HALF-DUPLEX, la réception et l'émission sont alternées et non simultanées. Usuellement, on emploie le mode DUPLEX.

Le mode de dialogue "3 fils" signifie que les données sont émises à n'importe quel moment, le dispositif récepteur étant toujours supposé libre de reçevoir un caractère.

Avec une imprimante lente, comme par exemple la Marguerite Olympia à 17 caractères/seconde, il peut y avoir des problèmes dûs au fait que l'imprimante reçoit plus de caractères qu'elle n'est capable d'en imprimer. Dans ce cas, le mode "X-line" permet de signaler au VIC qu'il doit attendre avant de transmettre le caractère suivant.

Par exemple, pour une transmission par modem : parité paire, duplex, mode de dialogue trois fils, le second octet de commande est :

CHR$(CMD)= 01100000 =CHR$(64+32)=CHR$(96)

Dans l'exemple cité, (OPEN2,2,0, CHR$(134)+CHR$(96)), on obtient une transmission DUPLEX avec 8 bits + 2 stops et parité paire à 300 bauds, c'est-à-dire que chaque caractère est transmis comme 12 bits (1 bit de START, 8 bits de données, 1 bit de parité, 2 bits de STOP) soit une vitesse de transmission de 300/12 = 25 caractères/seconde.

L'ouverture du canal de communication implique l'attribution par le système d'une zone de 512 octets à l'usage des deux tampons d'entrée et de sortie. L'interpréteur BASIC effectue tout d'abord une instruction CLR, puis réserve 512 octets au sommet de la mémoire disponible; et ce, sans tester si cette zone n'est pas occupée par un programme. Il est donc prudent, surtout sur un VIC sans extension mémoire, de réserver les premières lignes du programme à l'ouverture du canal de communication :

```
1 IF FRE(0)<512 THEN PRINT"OUT OF MEMORY ERROR":END
2 OPEN2,2,0, CHR$(134)+CHR$(96)
```

Il convient donc de procéder à l'ouverture du canal avant d'utiliser une seule variable (pas de DIM, DEFFN, etc).

2. Réception de caractères.

La réception de caractères est réalisée par l'intermédiaire du VIA 1. L'entrée des données est reliée également à la ligne de dialogue CB1 du VIA 1, ce qui lui fait générer une interruption non masquable NMI. L'emploi du NMI se justifie par le fait qu'il ne faut pas interférer avec les routines clavier et écran qui utilisent l'autre interruption disponible, INT. C'est par ailleurs cet emploi du signal NMI qui interdit l'emploi de la cassette ou de l'interface série IEEE , qui utilisent tous deux ce signal. La routine de réception place l'octet reçu dans le tampon de réception (255 octets). réservé en sommet de mémoire au moment de l'OPEN. L'instruction Basic GET£2 enlève un caractère du tampon de réception, l'instruction INPUT£2 enlève tous les caractères jusque et y compris un RETURN = CHR$(13). Si aucune donnée n'est présente, l'instruction rend une chaîne de caractères vide. Si par contre le tampon est rempli, toutes les données reçues sont perdues. On peut s'en rendre compte par la valeur de l'octet 663 ($0297) en mémoire : le bit 2 indique que le tampon est plein :

```
IF (PEEK(663)AND4)=4 THEN PRINT"tampon rempli"
```

Cette condition peut arriver très souvent pour des vitesses supérieures à 300 bauds, limite à ne pas dépasser en Basic, en tout cas en mode "3 fils".

3. Emission de caractères.

L'émission est réalisée également par l'intermédiaire d'un tampon de 255 octets et du circuit VIA 1. L'émission est transparente; c'est-à-dire que l'utilisateur n'a pas à s'en soucier. Le print £2 transfère la chaîne à émettre dans le tampon et les routines du KERNAL s'occupent de programmer les minuteries du 6522, qui génèrent un NMI au moment de l'émission d'un caractère. La routine NMI envoie alors le caractère après l'avoir retiré du tampon d'émission. En mode "X-line" le VIC n'émet un caractère que si le fil CTS de dialogue est à l'état 1.

4.Fermeture du canal de communication.

L'instruction CLOSE 2 a plusieurs effets. Tout d'abord elle provoque la fin de la transmission. Tous les octets non encore émis du tampon d'émission vers l'extérieur ou non encore lus par le BASIC dans le tampon de réception sont perdus et les 512 octets libérés pour être ré-utilisés par le BASIC. Les fils DATA OUT et RTS repassent à l'état 1. Le vecteur de NMI est rétabli à sa

valeur par défaut, ce qui ré-autorise l'emploi de l'interface série IEEE et de la cassette. Il est conseillé d'attendre que le tampon d'émission soit vide pour fermer le canal et que le fil CTS soit à l'état 1 (bit 6 du port B VIA 1 en 37151).

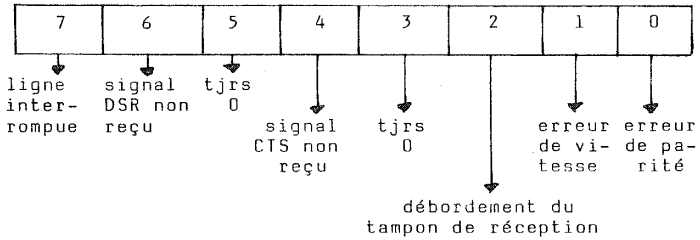
```
1000 IF ST<>0 THEN 1000 : REM attente tampon vide
1010 IF (PEEK(37151)AND64)=64 THEN 1010 : REM attend CTS
1020 CLOSE 2
```

Le mode de dialogue "X-line"

En mode "3 fils", il n'y a pas de problème : seuls sont connectés les fils DATA IN, DATA OUT et GROUND (référence). En mode "X-line", les choses se compliquent. Un des cas les plus fréquents d'usage de l'RS232 est la connexion d'une imprimante. Or pour beaucoup d'imprimantes série, il n'est pas possible de reçevoir des caractères pendant le retour de chariot, d'où la nécessité d'un dialogue avec le VIC. Usuellement le fil 4 du connecteur DB-25 (correspondant à D sur la sortie VIC) est utilisé pour cela. RTS signifie "REQUEST TO SEND" ou "demande d'envoi". Par ce fil, l'imprimante signale au VIC qu'il peut envoyer un caractère. Si dans votre cas, il s'agit d'un autre fil (parfois le 11 du connecteur DB-25), il convient de déplacer le signal vers la broche convenable. Le fil RTS est à l'état 1 si l'imprimante ne peut pas recevoir de caractères. (état 1 en RS232 = de -12 à -3 V). ATTENTION : Malheureusement, dans le mode "X-line", une erreur existe dans la ROM KERNAL qui provoque parfois le blocage complet de l'ordinateur:même les touches STOP-RESTORE ne peuvent le redémarrer. En fait, le KERNAL attend un changement de niveau d'une des lignes de dialogue à un moment où celui-ci ne peut se produire. Il existe un moyen de contourner ce problème en modifiant la routine défectueuse. (Voir ci-dessous la correction de l'erreur X-line).

La gestion des erreurs.

Après chaque transfert RS232, on peut si on le désire, tester que tout s'est bien passé : le registre de contrôle d'un 6551 est situé en mémoire à l'adresse 663 ($0297). On peut en tester les huit bits; ils indiquent les erreurs suivantes (1 = erreur, 0 = pas d'erreur) :



Des erreurs de débordement de tampon peuvent arriver à cause des fréquentes concaténations de chaînes nécessaires pour recevoir avec GET £2.

Exemple: GET£2, A$: B$=B$+A$). Les concaténations de chaînes provoquent régulièrement des "ramassage d'ordures", ou GARBAGE COLLECTION , c'est-à-dire des réorganisations de mémoire qui prennent un temps non négligeable.

L'emploi d'un lecteur de disque ou de cassette utilisant donc le signal NMI provoque une perte de caractères sur l'RS232.

Comment se servir de l'RS232 en langage machine?

Le langage machine, bien plus rapide, est en fait la seule méthode efficace pour les vitesses supérieures à 300 bauds. Les routines KERNAL les plus utiles sont les suivantes :

$FFC0 Ouverture d'un fichier de communication.
$FFC6 Ouverture d'un fichier de communication en entrée en mode "X-line". Ceci gère les lignes de dialogue RTS, CTS et DCD, le VIC étant considéré comme le terminal.
$FFC9 Ouverture d'un canal de communication en sortie (voir ci-dessus).
$FFE4 Réception d'un caractère (caractère reçu dans A).
$FFD2 Envoi d'un caractère.
$FFC3 Fermeture d'un fichier de communication.

Pour le détail de fonctionnement de ces routines, voir le chapitre KERNAL.

Les adresses utiles en RAM sont les suivantes :

$A7	Stokage temporaire bit reçu
$A8	Compteur de bits reçus
$A9	DRAPEAU : start bit reçu
$AA	Octet reçu en cours d'assemblage à la réception
$AB	Stokage du bit de parité reçu
$B4	Compteur de bits envoyés
$B5	Prochain bit à envoyer
$B6	Octet en cours d'émission
$F7-$F8	Pointeur vers début de tampon de réception
$F9-$FA	Pointeur vers début de tampon d'émission
$0298	Nombre de bits à recevoir/ à transmettre
$0299-$029A	Temps de transmission d'un bit en nombre de tops d'horloge du 6502
$029B	Pointeur fin de zone utile du tampon de réception
$029C	Pointeur début de zone utile du tampon de réception
$029D	Pointeur début de zone utile du tampon d'émission
$029E	Pointeur fin de zone utile du tampon d'émission

Les routines ROM du KERNAL peuvent être utiles. On peut les repérer dans la **CARTE MEMOIRE** générale. Elles sont situées dans la zone mémoire de $EFA3 à $F160. Bien qu'elles puissent être utiles, il est toutefois conseillé d'employer les routines habituelles OPEN , CLOSE, etc par souci de compatibilité avec de futures machines de chez CBM comme le COMMODORE 64, par exemple.

Correction de l'erreur de dialogue X-line du KERNAL.

1. Réserver une place en mémoire (tampon cassette, ligne REM en BASIC, sommet de mémoire, RAM/ROM entre $A000 et $BFFF).
2. Y installer la routine RSPATCH (Voir page suivante).
3. Modifier le pointeur de NMI avec JSR MODIF ou SYS MODIF en BASIC. L'adresse MODIF est définie dans le programme RSPATCH.

```
0:RSPATCH.....PAGE 0001

LINE# LOC   CODE          LINE


0001  0000              ;CORRECTION DE L' RS-232 VIC-20
0002  0000                                ;
0003  0000              BITCNT = $B4
0004  0000              DEVCUR = $BA        ;PERIPHERIQUE EN COURS DE TRANSFERT
0005  0000              VECNMI = $0318
0006  0000              PORTB1 = $9110
0007  0000              DDRB1  = $9112
0008  0000              NMIGO  = $FEAD      ; ADRESSE DE ROUTINE NMI
0009  0000                                ;
0010  0000                    * = $033C     ;TAMPON DE CASSETTE
0011  033C                                ;
0012  033C  48          PATCH  PHA
0013  033D  A5 BA               LDA DEVCUR
0014  033F  C9 02               CMP #$02    ;EST-CE BIEN L' RS-232 ?
0015  0341  D0 13               BNE END
0016  0343  AD 12 91           LDA DDRB1   ;SI OUI,PROGRAMME
0017  0346  29 FD               AND #$FD    ;LE FIL RTS EN ENTREE
0018  0348  8D 12 91           STA DDRB1
0019  034B                                ;
0020  034B  AD 10 91    ATTEND LDA PORTB1    ;LIT L'ETAT DE RTS
0021  034E  29 02               AND #$02    ;TANT QUE RTS VAUT 1
0022  0350  D0 04               BNE END     ;ATTENDRE QUE L'OCTET
0023  0352  A5 B4               LDA BITCNT  ;EN COURS SOIT ENVOYE
0024  0354  F0 F5               BEQ ATTEND
0025  0356  68          END    PLA
0026  0357  4C AD FE           JMP NMIGO   ;RETOUR A LA ROUTINE NMI
0027  035A                                ;
002B  035A  78          MODIF  SEI          ;ACCES PAR SYS858
0029  035B  A9 03               LDA #>PATCH
0030  035D  8D 19 03           STA VECNMI+1
0031  0360  A9 3C               LDA #<PATCH
0032  0362  8D 18 03           STA VECNMI
0033  0365  58                 CLI
0034  0366  60                 RTS
0035  0367                     .END


ERRORS = 0000


SYMBOL TABLE

SYMBOL  VALUE
 ATTEND   034B    BITCNT   00B4    DDRB1    9112    DEVCUR   00BA
 END      0356    MODIF    035A    NMIGO    FEAD    PATCH    033C
 PORTB1   9110    VECNMI   0318

END OF ASSEMBLY
```

POIGNEES DE JEU.

Les manettes de jeu analogiques COMMODORE ou ATARI sont identiques et comprennent chacune un bouton de tir et un potentiomètre. La manette gauche utilise l'entrée POT X du 6561 et le bit d'entrée JOY 2, qui sert de bouton gauche quand on utilise le manche de commande. De même, la droite utilise POT Y et JOY 3.

Bouton gauche :

```
BG = -((PEEK (37151) AND 16)=0)
```

Valeur gauche :

```
VG = PEEK (36872)
```

Bouton droit :

```
POKE 37154,127
BD = PEEK (37152) AND 128
POKE   37154,255          (Voir paragraphe manche de commande )
```

Valeur droite

```
VD = PEEK (36873)
```

VD et VG prennent des valeurs entre 0 et 255 inclus.

On notera que la course utile du potentiomètre occupe environ 90 des 300 degrés de rotation possible. Ceci est dû à la grande imprécision de fabrication des potentiomètres ordinaires : les concepteurs du VIC ont mis en place des sécurités vis-à-vis des variations de résistance possible.

A la grande différence du manche de commande, on peut déplacer un mobile à une vitesse variable et non pas-à-pas : pour obtenir des valeurs de coordonnées X et Y avec les deux poignées, on peut ramener les valeurs VD et VG entre 0 et XM par la règle de trois classique :

```
X = INT (VG*XM/255)
ou  X = XM - INT(VG*XM/255)
```

on choisira l'une ou l'autre de ces solutions suivant la direction de rotation désirée pour obtenir des X croissants.

Question : Peut-on utiliser des manches de commandes analogiques à deux axes ?

Réponse : OUI

Explication : un manche peut se raccorder aux deux entrées POT X et POT Y. Si on désire deux manches à deux axes, il faut les lire en séquence. Pour ce faire, on utilise la technique du multiplexage : un relais à double inverseur raccorde l'un ou l'autre manche de commande à POT X et POT Y, le relais lui-même pouvant être commandé par le bit JOY 0 ou le bit JOY 1 (bits 2 et 3 du PORT A du VIA 1). Pour cela, il faut mettre le bit 2 (par exemple) en sortie par :

```
POKE 37139, PEEK (37139) OR 4
```

et réaliser la commutation par :

```
POKE 37151, PEEK (37151) AND 4
```

ou

```
POKE 37151, PEEK (37151) AND 4 OR 4
```

Il faut attendre entre commutation et lecture environ 5 centièmes de seconde . Attention en réalisant ce montage, un bit de sortie du 6522 ne peut commander directement un relais, il convient d'employer un circuit amplificateur (disponible dans les boutiques électroniques). Les modèles XR-2203 ou TANDY 276-9223 conviennent bien.

CHAPITRE 2

LES PROGRAMMES INTERNES DU VIC

L'INTERPRETEUR BASIC

A l'achat d'un VIC, 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 VIC. A un point tel d'ailleurs que bien des utilisateurs de VIC ne savent même pas que **BASIC** est un programme. On cite en général BASIC en tant que langage. Or aucun ordinateur, pas même le VIC, ne peut comprendre ce langage. La langue maternelle du 6502, cerveau du VIC, 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 6502. Le fait que le BASIC est avant tout un programme et non avant tout un langage est extrèmement 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 lent comme procédé mais c'est simple et efficace.

Le programme BASIC est remarquablement court au vu de son efficacité. Dans le VIC, le BASIC occupe la mémoire ROM aux adresses $C000 à $E500, soit 9472 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 VIC, 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** (mais pas un SHIFT-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 dans le VIC 76 mots-clés. 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 $C09E ou 49310), 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 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 (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 VIC, 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 $C09E. 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 VIC avec la zone texte débutant en $1000 pour la version de base et les versions avec mémoire 8 K et plus. (En fait, pour les versions ayant de la mémoire RAM de $1000 à $ 2100 au minimum). La version avec extension 3 K voit sa zone texte débuter en $0400. Détaillons le contenu d'une version de base contenant les lignes BASIC suivantes :

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

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

Adresses	Valeur	
$1000	$00	Le premier caractère du texte BASIC est toujours $00
$1001	$08	$1008 : adresse de la ligne BASIC suivante.
$1002	$10	
$1003	$0A	$000A =10 : numéro de ligne
$1004	$00	
$1005	$99	$80 + $19 : 25ème mot-clé de la table = PRINT
$1006	$41	= "A" en ASCII
$1007	$00	signifie fin de ligne Basic.
$1008	$10	$1010 adresse de la ligne BASIC suivante
$1009	$10	
$100A	$14	$0014= 20 : numéro de ligne
$100B	$00	
$100C	$89	$89= $80+$9 : 9me mot clé de la table = GOTO
$100D	$31	"1" en ASCII
$100E	$30	'0' en ASCII
$100F	$00	signifie fin de ligne BASIC
$1010	$00	Adresse de la ligne BASIC suivante. La valeur $0000 signifie fin de programme.
$1011	$00	

En annexe, on trouvera le programme **SUPERLIST** qui donne les numéros et les adresses des lignes BASIC en mémoire après sauvegarde sur cassette par le programme **PROGDATA.**

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 **CHARGOT** qui est situé en mémoire RAM en page zéro (de $73 à $8A). Voir en annexe le texte de la routine CHARGOT. CHARGOT a deux fonctions : prendre un caractère en mémoire, ce qui implique que CHARGOT contienne un pointeur de caractère courant qui doit se modifier à chaque appel. De plus, si le caractère est un espace, ($20) , CHARGOT l'ignore et prend le caractère suivant. Deuxièmement, le flag CARRY du 6502 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 CHARGOT lui renvoie est alphabétique. CHARGOT est de très loin la routine la plus employée de tout le VIC, ce qui explique le soin extrême apporté à sa réalisation. La routine CHARGOT 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) 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 CHARGOT, 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 VIC 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 $C00C. Soit le mot-clé PRINT (code $99 ou 153) : le numéro du mot-clé est 153-128 = 25. Chaque élément de la table ayant deux octets de long, l'adresse de la routine PRINT est située en ($C00C+1+2*25 = $C03E) et ($C00C+1+2*25+1=$C03F). La valeur lue en $C03E est $9F et en $C03F, on trouve $CA. L'interpréteur ayant trouvé ainsi l' adresse $CA9F, 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 $C00C 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 6502 : 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 "BYTES FREE",message présent en mémoire en $E429, on effectue le programme suivant :

```
LDA £$29
LDY £$E4
JSR $CB1E      ;voir détail de cette rou-
RTS            ;tine 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 VIC, 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 1 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 carte-mémoire du VIC.

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	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

Comment construire ou 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

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 vaut 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 FLP1 et ACCU FLP 2). 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 : 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.

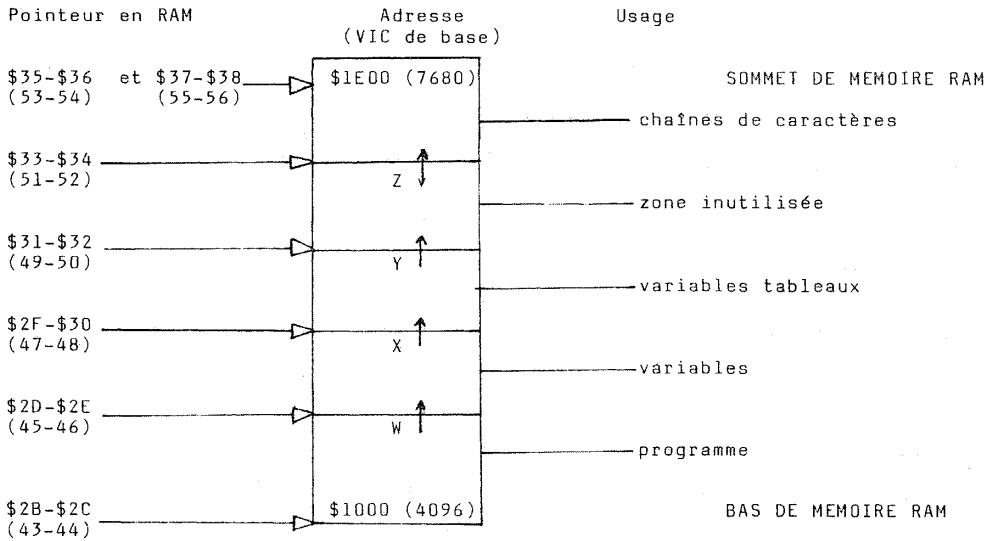
Le langage BASIC autorise l'emploi de variables entières de valeurs comprises entre -32767 et +32768 mais, contrairement à une rumeur répandue, l'emploi de ces variables ralentit l'exécution du BASIC car elles nécessitent à chaque usage une conversion flottant-entier. L'intérêt de ces variables est, 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 VIC 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 modifient et imposent 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 flèches indiquent le sens d'évolution des débuts de zones pendant l'exécution d'un programme.

Détails de l'usage de la mémoire 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 FAMILLE).

1.Variable flottante

1er caractère du nom (bit7 = 0)	2me caractère du nom (bit7 = 0)	Valeur en flottant sur 5 octets				
		MSB				LSB

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.

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

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.

1er caractère du nom (bit7=0)	2me caractère du nom (bit7=1)	longueur de la chaîne	LSB (bas) adresse de la chaîne	MSB (haut)	0	0
					2 OCTETS INUTILISES	

4.Les fonctions FN

Les fonctions FN sont définies par l'instruction DEF.
Exemple :
DEF FNAA(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 $0001 et $0002 s'obtient par PRINT FNAA(1). 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=FN(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.

1er caractère du nom de la fonction (bit7=1)	2me caractère du nom de la fonction (bit7=0)	LSB pointeur vers définition de la fonction (bas)	MSB (haut)	LSB pointeur vers la variable locale +$02 (bas)	MSB (haut)	premier caractère 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 représenter un gain de temps 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 **SUPERVARIABLES**. 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 63509, 63518, 63526 et 63531 qui peuvent être réutilisés à d'autres fins : ils réalisent la conversion des données du format binaire propre au VIC en caractères lisibles par tous. On remarquera le sous-programme 63507 qui affiche les deux caractères du nom d'une variable. La conversion des valeurs flottantes de binaire en décimal s'effectue à la ligne 63509. L'exposant ZE est calculé en 63511 et la MANTISSE ZJ EN 63512. Le résultat décimal se retrouve dans ZJ en 63515.

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 codée sur deux octets (bas,haut) du tableau-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 : éléments 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

Octet haut(MSB)				octet bas(LSB)	Voir la définition des nombres flottants ci-dessus

Entier

partie haute	partie basse	le bit7 du premier octet représente le signe

Chaîne

Longueur de la chaîne	Adresse de la chaîne	
	(bas)	(haut)

Pour rendre plus clair l'emploi de la zone tableaux , il est possible d'utiliser le programme **"SUPERTABLEAUX"** en annexe. Ce programme doit resider en mémoire en même temps que SUPERVARIABLES et eventuellement un autre programme à analyser.
Notez bien que toutes les variables utilisées dans SUPERVARIABLES et SUPERTABLEAUX doivent être déclarées avant d'entrer dans le programme ,sinon les pointeurs des différentes zones se modifient sans cesse.

SUPERTABLEAUX

63543 création des variables.

63544 ZB est l'adresse de début de zone tableaux.

63545 ZM est l'adresse de fin de zone tableaux.
ZN est l'adresse de début d'un tableau.

63546 teste si fin de zone tableaux.

63547 ZO est le nombre d'indices du tableau.

63548 calcul des valeur maxi des indices.

63549 idem.

63550 ZD est le type de tableau (0 = flottant, 1 = chaîne, 3 =entier)

63551 ZB$ est le nom du tableau

63552 fin de nom, si chaîne.

63553 fin de nom si entier.

63554 affichage du nom du tableau avec ses indices maxi comme lors de DIM.

63555 ZL est le nombre d'éléments du tableau.

63556 initialisation des indices.

63557 ZC : paramètre d'adresse pour appel.

63558 début de la boucle principale.

63559 affichage du nom de l'élément.

63560 à 63562 calcul ZC suivant le type de variable.

63564 à 63568 calcul des indices du tableau.

63569 attente d'une frappe au clavier.

63571 calcul de l'adresse du tableau suivant.

63572 affichage de chaîne.

63575 affichage du nom de l'élément.

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 $C00C. 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 $C00C.

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 **CHARGOT** qui a décelé que le premier caractère qui suit END n'est pas alphabétique. La routine d'erreur qui se situe en $C43A 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 $C19E.

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 $C474.

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 CHARGOT. 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 ait ni commis d'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 DENEW). 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 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.LES COMMANDES BASIC AVEC ARGUMENTS 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(3,3,3) : 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 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 créées 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 crée aussi sa propre variable qui occupera 7 octets dans la zone VARIABLES . 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 problème 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 6502. 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 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 LIST0 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 **$C09E** pour le transcodage. Une version BASIC de LIST est disponible en annexe (SUPERLIST). Jumelé au programme PROGDATA qui sauve sur cassette un programme sous forme de fichier de données, SUPERLIST exécute les mêmes opérations de transcodage que LIST. On voit 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 CHARGOT 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 VIC car le BASIC MICROSOFT 2.0 était ainsi fait. Il s'agit de la version de l'interpréteur BASIC installée dans les PET et CBM 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 6502 lui-même : c'est le registre **SP.** Le pointeur est mis 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 VIC. Mais ils nécessitent quasi toujours la transmission de paramètres dans les registres A,X et Y du 6502. 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 6502 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 6502 (au moment du RTS qui rend le contrôle au BASIC) par PEEK(780),...,PEEK(783).

Détaillons ici un point souvent resté OBSCUR : les **DRAPEAUX** du 6502 se positionnent par POKE783,FL

DRAPEAUX DU 6502

N	V	.	B	D	I	Z	C
128	64		16	8	4	2	1

la valeur

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
```

L'usage des différents drapeaux est décrit au chapitre 1.

Pour appeler un sous-programme en langage machine avec transmission d'un paramètre en flottant, 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-utilisés par RETURN.

De même, RETURN est un GOTO avec le paramètre lu dans la pile 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'

1er octet	2me	3me	4me	5me
$80	numéro de ligne BASIC du GOSUB (bas)	(haut)	pointeur du premier caractère après le GOSUB (bas)	(haut)

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 continue l'interprétation.

4.LES BRANCHEMENTS CONDITIONNELS 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 VIC 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 est nul 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 <u>valeur</u> 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 VIC, 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 variable = limite inf. TO limite sup. STEP incrément
...
...
NEXT variable (variable, 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 :

(variable d'indice) > limite fin de boucle si l'incrément est positif
ou (variable d'indice) < limite fin de boucle si l'incrément est 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. Notamment à 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 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 X OR C (OU EXCLUSIF avec le troisième argument) suivi de AND B (ET avec le deuxième argument qui vaut 0 s'il n'est pas cité explicitement).

Par exemple, 10 WAIT 37164,48,16 est semblable à :

```
10 IF ( PEEK (37164) AND 48 OR 16) 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 48 signifie : je ne m'intéresse qu'aux bits 4 et 5 car 48=0011000 en binaire et seuls les bits 4 et 5 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 5 et 4 doivent valoir 0 et 1 respectivement, on mettra le bit 5 à 0 et le bit 4 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 en binaire	Resultats en binaire
1er cas : WAIT 37164,48,16 10 IF PEEK(37164) AND 48 XOR 16 THEN 10	01011110 (94) 00110000 (48) 00010000 (16)	01011110 (94) 00010000 (16) 00000000 (0) = 0 : on n'attend pas
2me cas : WAIT 37164,48,16 10 IF PEEK(37164) AND 48 XOR 16 THEN 10	01111110 (126) 00110000 (48) 00010000 (16)	01111110 (126) 00110000 (48) 00100000 (32) <>0 : on attend.

L'usage de l'instruction WAIT est en pratique limitée à l'attente de modification d'un bit d'entrée-sortie. (Exemple : attente de CB2=1 dans un programme qui gère des dialogues entre deux VIC reliés par le USER'S PORT).

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 VIC 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 : l'interpréteur BASIC 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 VIC 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

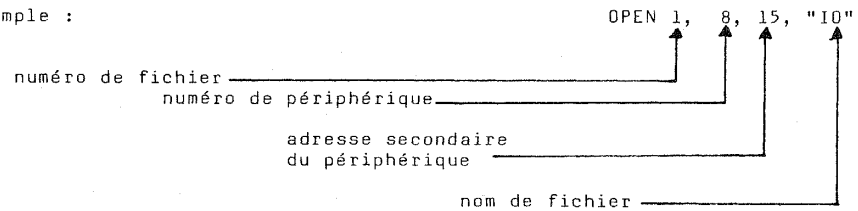
3. Les autres périphériques sont supposés se trouver sur le BUS IEEE série où les numéros autorisés vont de 4 à 31. Usuellement les imprimantes portent le numéro 4, le lecteur de disquettes le 8 (mais il est modifiable par programme), le modem téléphonique le 9. Nous avons ajouté l'imprimante parallèle en 6 (voir annexe 3).

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.

Exemple :



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 VIC 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 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: 1. L'instruction INPUT de la ROM contient une erreur. Si INPUT est utilisé avec un paramètre 'question' comme dans :

```
INPUT"VOTRE NOM";A$
```

Le fonctionnement du VIC est incorrect si la réponse (ici A$) déborde sur la ligne suivante de l'écran . BASIC attribue à A$ non seulement la réponse mais aussi la question ! Solution : effectuer OPEN 3,3:CMD 3:CLOSE 3 en début de programme et utiliser PRINT "QUESTION";:INPUT A$
Notez que dans ce cas, on doit explicitement mettre le '?' que BASIC n'envoie pas. En effet, comme CMD 3 n'est pas suivi de PRINT£3, le BASIC considère l'ecran comme un périphérique.

2. L'instruction INPUT£ se bloque irrémédiablement si elle ne rencontre pas un caractère "RETURN" (CHR$(13)) avant 88 carac-

tères ! 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 $CEF1. Exemple :

```
A
A$
A+10^4
MID$(A$,2,1)
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, 11me, 23me 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 22 c'est-à-dire le reste de la division entière de X par 22. Après avoir ainsi ramené la valeur entre 0 et 21, 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(10); "?"
PRINT "?"; TAB(10); "?"
```

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 imprimante :

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

Exemple : lister sur 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 VIC 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 ou 2. 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 l'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 VIC au moment du LOAD. Cette dernière méthode conduit à des erreurs si le VIC au moment du LOAD ne contient pas de mémoire aux adresses où se trouvait le programme lors du SAVE. Mais elle est très utile pour sauver et recharger des images-écran, des générateurs de caractères, des programmes en langage machine.

Plus astucieux encore, si la méthode absolue est employée lors du SAVE, un LOAD en mode NORMAL chargera quand même le programme en absolu.

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

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

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

3. SAVE "TOTO", 8, 1
   LOAD "TOTO", 8, 1

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**, qui signifie 'fin de bande'. En conséquence tout LOAD, SAVE ou VERIFY qui découvre ce bloc E.O.T. arrête sa recherche, la fin de bande étant atteinte.

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 $CEF1. 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 FLP.**

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 valeurs des constantes dans la carte-mémoire). 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 $E040. La méthode de calcul de **COS** et **TAN** est indirecte : en fait, BASIC n'évalue que le polynôme de sinus. Pour le cosinus, la routine **COS**, après évaluation de l'expression, y ajoute la valeur pi/2, stockée en **$E2DD** 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 : 11, 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 $01 et $02. 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 $0033 et le pointeur fin de tableaux en $0031.

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

```
1 CLR
2 A$="COUCOU"
3 A$=A$
4 A$=A$+"!"
5 A=FRE(0)
6 END
```

Contenu de la zone chaînes

```
                        TOP
                        TOP
                COUCOU  TOP
COUCOU !        COUCOU  TOP
                COUCOU !TOP
                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 ($00D3). A noter que PEEK(214) 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 22 caractères mais bien ligne BASIC qui , suivant les cas, occupe 1, 2, 3 ou 4 lignes de 22 caractères : une ligne BASIC peut avoir jusqu'à 88 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, 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 VIC. 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 VIC (comptage en micro-secondes en **VIA2-6 et VIA2-7).**

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 au moins, 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%
```

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.

Exemples de chaînes pouvant convenir ;

```
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.

+,-,*,/,^,- 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.

Les opérateurs logiques **AND,OR** et **NOT** ont un double but : les manipula-

tions 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 +32768 et -32767, 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 ou d'un VIA par exemple), dont on désire positionner le bit3 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 VIC.

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 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))
```

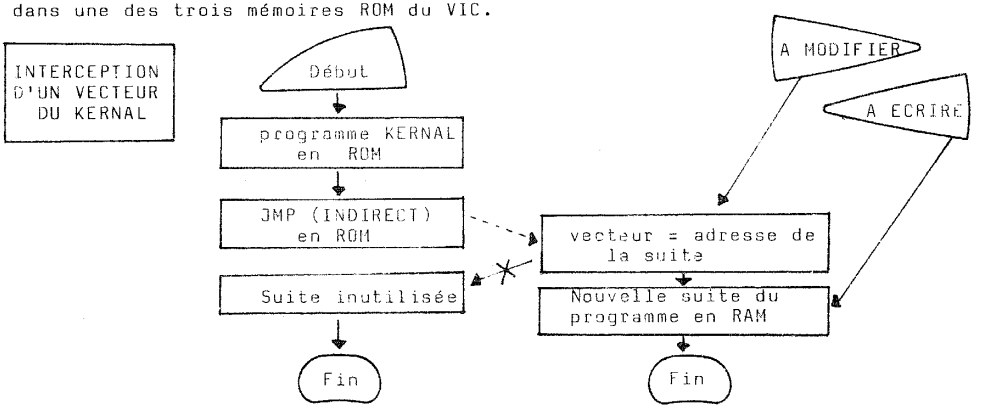
Faire plus avec BASIC

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

Pour ajouter vous-même des fonctions au VIC, ou simplement pour mieux comprendre son fonctionnement, il est utile de connaître les routines du BASIC. On en trouvera une description à la fin du chapitre. 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 VIC, 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 VIC.



Les programmes internes 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 VIC mais ils sont modifiables très aisément par le programmeur.

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 $FF84 à $FFF3 : 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 rou-

tines du KERNAL est cependant particulière : la routine CHARGOT 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 extrèmement 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 les interpréteurs de langage VIGIL (langage graphique) et PIPER (langage musical) ou des programmes ajoutant des commandes supplémentaires au BASIC (SIMON'S BASIC, SUPEREXPANDER, VICKIT, etc) utilisent tous des versions propres de la routine CHARGOT.

AUTO-START : Au démarrage, le KERNAL exécute une série d'opérations d'initialisation. L'adresse de la routine de RESET est toujours, avec le 6502, située en $FFFC et $FFFD. Pour le VIC, le contenu de cette adresse est $FD22. 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 $A000 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 VIC, la suite de caractères $41, $30, $C3, $C2, $CD (A, 0, C, B, M en ASCII - C, B, M avec le bit 7 =1) à l'adresse $A004. 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 $C000, pour initialiser l'interpréteur BASIC, ce que l'on peut toujours faire é la main en exécutant le code JSR $FD3C ou SYS(64828). Pour redémarrer un programme AUTO-START dont l'effet a été ainsi annulé, effecuez un JSR $FD2C ou SYS(64812) qui contient le code JMP(A000).

Les ROM AUTO-START contiennent en $A000 et $A001 l'adresse du début de programme (COLD START). Les adresses $A002 et $A003 contiennent une adresse de redémarrage du même programme (WARM START) utilisée quand on emploie les touches STOP-RESTORE. Pour démarrer le VIC avec une ROM auto-start sans qu'elle ne s'exécute, voir le chapitre 'Les extensions matérielles'.

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 VIC procure au programmeur d'importantes facilités. En effet, les arguments qui doivent être présents dans les registres du 6502 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 DU BASIC

$0073 CHARGOT ACQUIERT UN CARACTERE
Routine en ram qui acquiert un caractère dans le texte basic en cours d'interpretation.

$C408 RAMTST TEST MEMOIRE
Recherche le sommet réel de mémoire RAM à partir de l'adresse stockée dans les registres A (bas) et Y (haut) par test non-destructif de la mémoire.

$C435 ERRS ECRIT MESSAGE D'ERREUR
Passe par un vecteur en RAM en $0300 et revient normalement en $C43A = NERROR.En entrée,numéro de message dans le registre X. Prend l'adresse du message dans la table des adresses de messages en $C326 puis imprime le message suivi de "ERROR".

$C447 ERRS2 ECRIT MESSAGE D'ERREUR
Ecrit le message dont l'adresse est en $22, $23. Le dernier caractère du message doit avoir son bit 7 à 1. Ecrit ensuite "ERROR". Referme les canaux d'entrée/sortie sauf clavier/écran .Retour à BASIC READY.

$C474 READY BASIC READY
Redémarrage de BASIC en mode direct comme par END, STOP ou fin de programme. Appelle $CB1E pour écrire le message. Redémarre BASIC par le vecteur en RAM en $0302 qui renvoie en $C483.

$C483 NMAIN INTERPRETEUR DE COMMANDES
Attend commande BASIC au clavier et l'exécute.

$C560 GETBAS ACQUIERT COMMANDE BASIC
Attend une ligne BASIC au clavier et la transfère dans le tampon d'entrée en $0200.

$C57C NCRNCH ENCODE BASIC
Reduit la taille mémoire d'une ligne BASIC dans le tampon d'entrée en $0200 en encodant les mots-clés (voir table en $C09E).

$C642 NEW — ANNULE PROGRAMME BASIC
Entrée en $C642 avec registre A = 0 ou entrée en SCRTCH $C644 sans paramètres. Ecrit 0 dans les 2 premiers octets du texte BASIC en mémoire ce qui signifie : Fin de programme. Retablit les valeurs debut et fin de programme en $2B, $2C, $2D, $2E. Execute CLR et referme les fichiers. Execute RESTORE. Retablit le pointeur de pile à $FA en sauvegardant l'adresse de retour qui est dans la pile. Retour à la routine appellante.

$C7E1 EXEC — EXECUTE UNE COMMANDE BASIC
Passe par le vecteur en RAM en $0308 pour revenir en $C7E4 .Appelle CHARGOT. Appelle EXEC2 qui exécute la commande. A la fin de la commande, retour à BASIC READY.

$C7ED EXEC2 — EXECUTE COMMANDE BASIC
Recherche adresse de la routine à partir de la valeur TOKEN de la commande. Prend cette adresse dans la table en $C00C et la pousse dans la pile. Execute un saut vers CHARGOT en $0073 qui, par son RTS, renvoie à la routine à exécuter.

$CB1E STROUT — ECRIT CHAINE
Envoie une chaîne de caractères sur la sortie par défaut (normalement l'écran). La fin de la chaîne se signale par un caractère $00. En entrée, A et Y pointent vers le premier caractère de la chaîne (Y = haut, A = bas).

$CE83 EVALU — EVALUE UNE EXPRESSION
Effectue un JMP ($030A) (vecteur en RAM). puis NEVAL, en $CE86, evalue l'expression située à l'adresse contenue dans le pointeur en $7A,$7B. Resultat dans l'ACCU FLP1.

$CFE6 ORFAC — OU DES ACCUS FLP
Execute l'opération OU (sur 16 bits) des valeurs contenues dans les 2 ACCUS FLP. En sortie, résultat dans l'ACCU FLP 1.

$CFEB ANDFAC — ET DES ACCUS FLP
Exécute le ET logique sur 16 bits des valeurs contenues dans les 2 ACCUS FLP. En sortie, le résultat est dans l'ACCU FLP1.

$D1AA FLPINT — CONVERSION FLOTTANT-ENTIER
Le nombre situé dans l'ACCU FLP1 est converti en entier sur 16 bits. En sortie, la valeur entière est dans l'ACCU FLP 1, partie basse en $64, partie haute en $65.

$D37D FRE — EXECUTE FRE(x).
En sortie, le résultat est dans l'ACCU FLP1.

$D391 INTFLP — CONVERSION ENTIER-FLOTTANT
En entrée, la valeur entière sur 16 bits est dans Y (bas) et A (haut). En sortie, la valeur convertie en flottant est dans l'ACCU FLP1.

$D77C LEN — LONGUEUR D'UNE CHAINE
En entrée, $64 et $65 contiennent les 2 caractères du nom de la variable chaîne. En sortie, la longueur de la chaîne est dans le registre X

$D850 SUBTRA — SOUSTRACTION
En entrée, 2 valeurs dans les 2 ACCUS FLP, l'octet en $6F doit contenir la valeur OU EXCLUSIF des signes des ACCUS FLP ($66 et $6E). Le registre A doit contenir la valeur de l'octet en $61. Soustrait la valeur de l'ACCU FLP 2 de celle de l'ACCU FLP 1. En sortie, le résultat est dans l' ACCU FLP 1.

$D9EA LOG — LOGARITHME NATUREL
En entrée, paramètre dans l'ACCU FLP 1. Calcule le logarithme naturel (népérien). En sortie, résultat dans l'ACCU FLP 1.

$DA30 MPLM — MULTIPLICATION AVEC LA MEMOIRE
En entrée, un des opérandes est dans l'ACCU FLP 1, l'autre en mémoire au format flottant (usuellement dans une variable). Le registre Y (haut) et le registre A (bas) contiennent l'adrese du second opérande. En sortie, le résultat est dans l'ACCU FLP 1.

$DA33 MPLY — MULTIPLICATION
En entrée, 2 opérandes dans les 2 AC FLP, le OU EXCLUSIF des signes doit être en $6F. Le registre A contient la valeur en $61. En sortie, le résultat est dans l'ACCU FLP 1.

$DA8C MOV1 — TRANSFERT
En entrée, le registre Y (bas) et le registre A (haut) contiennent l'adresse d' une valeur flottante. La valeur est transférée dans l'ACCU FLP 2. Le OU EXCLUSIF des signes des ACCUS FLP est rangé en $6F et, le contenu de l' adresse $61 est rangé dans le registre A.

$DAE2 MPLY10 — MULTIPLIE PAR 10
En entrée, une valeur flottante dans l'ACCU FLP 1. En sortie,l'ACCU FLP 1 est multiplié par 10.

$DAFE DIVD10 — DIVISE PAR 10
En entrée, une valeur dans l'ACCU FLP 1. En sortie, l'ACCU FLP 1 contient la valeur divisée par 10.

$DB0F DIVD — DIVISION
En entrée, 2 valeurs dans les ACCUS FLP. Le OU EXCLUSIF des signes des ACCUS FLP doit être en $6F. La valeur en $61 doit être dans le registre A. Division du contenu de l'ACCU FLP 2 par l'ACCU FLP 1. En sortie, résultat dans l'ACCU FLP 1.

$DBA2 MOV2 — TRANSFERT
En entrée, une valeur flottante en mémoire. L'adresse de la valeur est dans les registres A (bas) et Y (haut). Transfère la valeur dans l'ACCU FLP 1, effectue le OU EXCLUSIF des signes et le range en $6F. Le contenu de $61 est mis dans le registre A.

$DBC7 MOV3 TRANSFERT
En entrée, valeur flottante dans l'ACCU FLP1. Adresse de zone mémoire dans le registre X (bas) et le registre Y (haut). Transfère la valeur flottante en mémoire avec le bit 31 de la mantisse représentant le signe situé en $66 pour l'ACCU FLP 1.

$DBFC MOV4 TRANSFERT
Transfère la valeur de l'ACCU FLP 2 dans l'ACCU FLP 1. Ne modifie pas la valeur de l'ACCU FLP 2.

$DC0C MOV5 TRANSFERT
Recopie le contenu de l'ACCU FLP 1 dans l'ACCU FLP 2 en effectuant l'arrondi.

$DC0F MOV6 TRANSFERT
Transfère l'ACCU FLP 1 dans l'ACCU FLP 2 sans arrondi.

$DC39 SGN SIGNE
En entrée, valeur dans l'ACCU FLP 1. En sortie, FLP 1 vaut :

0	si valeur = 0,
-1	si valeur négative,
+1	si valeur positive.

$DC58 ABS VALEUR ABSOLUE
En entrée, valeur dans l'ACCU FLP 1. En sortie, valeur absolue dans l'ACCU FLP 1.

$DC5B COMPAR COMPARAISON
En entrée, une valeur dans l'ACCU FLP 1 et une valeur flottante en mémoire dont l'adresse est dans le registre A (bas) et le registre Y (haut). En sortie,l'ACCU FLP 1 vaut 0 si les 2 valeurs sont égales; sinon, il vaut $FF.

$DCCC INT PARTIE ENTIERE
Conversion d'un nombre flottant dans l'ACCU FLP 1 en entier puis à nouveau en flottant: Seule reste la partie entière. En sortie, résultat dans l'ACCU FLP 1.

$DDCD LINPRT ECRIT UN NOMBRE ENTIER
En entrée, une valeur sur 16 bits est dans le registre A (haut) et le registre X (bas). Cette valeur est convertie en chaîne de caractères et envoyée au périphérique de sortie par défaut. Est utilisé par BASIC pour imprimer les numéros de ligne.

$DF71 SQR RACINE CARREE
En entrée, valeur dans l'ACCU FLP 1. Calcul de la racine carrée. En sortie, valeur dans l'ACCU FLP 1.

$DF78 POWER EXPONENTIATION
En entrée, 2 valeurs dans les 2 ACCUS FLP. En sortie, l'ACCU FLP 1 contient (valeur dans ACCU FLP 2) exposant (valeur dans l'ACCU FLP 1).

$DFED EXP ANTILOGARITHME
En entrée, valeur dans l'ACCU FLP 1. En sortie,(E) exposant (valeur) dans l'ACCU FLP 1.
Note: E est la base des logarithmes naturels.

$E094 RND NOMBRE AU HASARD
En entrée, $8B à $8F contiennent la valeur RND précédente, l'ACCU FLP 1 contient l'argument (voir **RND** en BASIC). Calcul du nombre pseudo-aléatoire suivant. En sortie, valeur pseudo-aléatoire dans l'ACCU FLP 1 et en $8B à $8F.

$E261 COS COSINUS
En entrée, une valeur flottante (en radians) est dans l'ACCU FLP 1. En sortie, le résultat est dans l'ACCU FLP 1.

$E268 SIN SINUS
Comme COS.

$E2B1 TAN TANGENTE
Comme COS.

$E30B ATN ARCTANGENTE
Comme COS.

$E378 INIBAS INITIALISE BASIC
Initialise les vecteurs en RAM et les adresses utiles au BASIC en page 0, 2, et 3. Se subdivise en 3 parties: INI1, INI2,INI3. Le registre pointeur de pile (SP) est remis à la valeur $FB et le contrôle rendu à READY en $C474.

$E387 CHRGOT COPIE DE CHARGOT
Routine non exécutable en $E387. Est recopiée en RAM en $0073 à l'initialisation. 24 octets de $E387 à $E39E recopiés en $0073-$008A. (Voir le texte de la routine **CHARGOT** en annexe).

$E3A4 INI2 INITIALISE CHARGOT ET PAGE 0
Initialise les adresses utiles au BASIC:$54,$00 à $06,CHARGOT en $73 à $8A, $53, $68, $13, $18, $01FD, $01FC, $16, $2B, $2C, $33, $34, $37, $38.

$E404 INI3 TESTE MEMOIRE RAM
Teste la mémoire RAM disponible et affiche la valeur suivie du message "BYTES FREE". Exécute NEW.

$E45B INI1 INITIALISE VECTEURS BASIC
Initialise la table des vecteurs en RAM de $0300 à $030B.

LES ROUTINES DU KERNAL

$E4A0 DATAHI ROUTINE IEEE SERIE
Met à 1 la ligne de données de l'IEEE série.

$E4A9 DATALO ROUTINE IEEE SERIE
Met à 0 la ligne de données de l'IEEE série.

$E505 SCRSIZ DONNE LA TAILLE DE L'ECRAN
Donne en sortie dans X et Y, la taille de l'écran par défaut:
(X)= 22 colonnes = $16,
(Y)= 23 lignes = $17.

$E518 SETSCR MODIFIE L'ADRESSE DE L'ECRAN + CURSEUR
Necessite 8 emplacements dans la pile. En entrée, la partie haute de la nouvelle adresse écran doit être mise en $0288. En sortie, les adresses $9002 et $9005 sont modifiées en conséquence. Emploie X,Y,A. Appelle la routine $EA8D qui initialise les VIA. En sortie, l'écran est effacé, la couleur d'écriture bleue,le curseur "HOME", visible et clignotant et le tampon de clavier est ré-initialisé. Cette routine initialise les adresses RAM $0291, $CF, $028F, $0290, $0289, $028C, $0286, $0288, $CC et $CD, ainsi que la table des adresses des lignes-écran en RAM en $D9 à $F1.

$E587 CURSOR GESTION DU CURSEUR
Gère le clignotement du **CURSEUR**. La position X,Y du curseur doit être définie (Y en $D3, X en $D6) avant d'appeller CURSOR. Très utilisé en BASIC pour positionner le curseur :
POKE 214,X:POKE211,Y:SYS58759

$E5C3 INIVIC INITIALISE LES REGISTRES DU 6561
Pas d'argument en entrée. Ne nécessite pas d'emplacement dans la pile. Utilise A et X. En sortie, la table des valeurs située en ROM en $EDE3 est recopiée dans les 16 registres du 6561 (V.I.C). Voir les valeurs de cette table sous la rubrique VICTBL ($EDE3).

$E5CF PRENDS CARACTERE AU CLAVIER
Prends un caractère dans le tampon clavier. En sortie, caractère dans le registre A.

$E64F INPT ENTRE UNE LIGNE AU CLAVIER
Entre des caractères au clavier jusqu'à la frappe d'un RETURN. En sortie, caractères dans le tampon d'entrée en

RETURN. En sortie, caractères dans le tampon d'entrée en $0200.

$E742 DSPLCH AFFICHE CARACTERE ECRAN
En entrée, caractère dans A. Le caractère est écrit à l'écran à l'endroit du curseur et le curseur avance d'une case.

$E800 GERE TOUCHES SHIFT/LOGO
Partie de la routine clavier.

$EABF CINV ROUTINE INTERRUPTION VECTORISEE
Entrée dans la routine d'interruption IRQ par le vecteur en RAM en $0314.

$EB1E KBSCN BALAYAGE CLAVIER
Lecture du clavier et traduction du code obtenu à l'aide d'une des 4 tables de décodage (Voir ci-dessous). En sortie, l'octet lu est mis dans le tampon de clavier.

$EBDC LOGIQUE DES SHIFT
Gestion des touches SHIFT, LOGO, CTRL pour lecture CLAVIER.

$EC5E KBDTB1 TABLE DE DECODAGE CLAVIER NORMAL
Taille de la table = 64 octets + un octet $FF (fin de table)

$EC9F KBDTB2 TABLE DECODAGE CLAVIER SHIFT

$ECE0 KBDTB3 TABLE DECODAGE CLAVIER LOGO

$ED72 KBDTB4 TABLE DECODAGE CLAVIER CTRL

$EDE3 VICTBL VALEURS INITIALES DU 6561
Table utilisée par la routine INIVIC en $E5C3. Contenu hexa:

FF 0C 26 16 2E 00 C0 00
00 00 00 00 00 00 00 00

$EE14 STALK COMMANDE IEEE
Envoie la commande TALK (parlez) sur le bus IEEE SERIE.

$EE17 SLSTN COMMANDE IEEE
Envoie la commande LISTEN (ecoutez) sur le bus IEEE série.

$EE40 SOUT COMMANDE IEEE
Envoie un octet sur le bus IEEE série.

$EE6F ROUTINE IEEE
Prépare à envoyer une donnée sur IEEE série.

$EEC0 SNDSEC COMMANDE IEEE
Envoie une adresse secondaire sur IEEE série après LISTEN.

$EEC5 ATNLO — COMMANDE IEEE
Remet à 0 la ligne ATTENTION du bus IEEE série, après LISTEN.

$EECE — COMMANDE IEEE
Envoie une adresse secondaire sur bus IEEE série.

$EED3 — COMMANDE IEEE
Met ATTENTION à 1 avant TALK sur le bus IEEE série.

$EEE4 — COMMANDE IEEE
Sortie avec tamponnage d'un octet sur bus IEEE série.

$EEF6 — COMMANDE IEEE
Envoie UNTALK sur bus IEEE série (fin de parole).

$EF04 — COMMANDE IEEE
Envoie UNLISTEN sur bus IEEE série (fin d'écoute).

$EF19 SINP — COMMANDE IEEE
Entre un octet en provenance du bus IEEE série.

$EF84 CLKLO — COMMANDE IEEE
Ne nécessite pas d'emplacement dans la pile. Remet à 0 le fil 'HORLOGE' de l'IEEE série.

$EF8D CLKHI — COMMANDE IEEE
Remet à 1 le fil 'HORLOGE' de l'IEEE série.

$EF96 DLY1MS — DELAI
Boucle d'attente d'une durée de 1 milliseconde.

$F1F5 IGETIN — LECTURE VECTORISEE DU CLAVIER
Lit un caractère dans le tampon de clavier. Point d'entrée après vecteur en RAM en $032A.Fait partie de la routine **GETIN** en $FFE4. En sortie, le caractère lu est dans le registre A. Utilise KBDBUF ($0277), KBDMAX ($0289), KBDPTR ($00C6).

$F20E IBASIN — ENTREE VECTORISEE D'UN CARACTERE
Accepte un caractère du canal d'entrée. Point d'entrée après le vecteur en RAM en $0324. Fait partie de la routine **CHRIN** en $FFCF.

$F27A IBSOUT — SORTIE VECTORISEE D'UN CARACTERE
Envoie un caractère sur le canal de sortie. Point d'entrée après vecteur en RAM en $0326. Fait partie de la routine **CHROUT** en $FFD2.

$F2C7 ICHKIN — ENTREE VECTORISEE CANAL D'ENTREE
Alloue le canal d'entrée à un fichier logique. Point d'entrée après le vecteur en RAM en $031E. Fait partie de la routine **CKKIN** en $FFC6.

Alloue le canal de sortie à un fichier logique. Point d'entrée après le vecteur en RAM en $0320. Fait partie de la routine **CHKOUT** en $FFC9.

$F34A ICLOSE ENTREE VECTORISEE CLOSE
Fermeture d'un fichier logique. Point d'entrée après le vecteur en RAM en $031C. Fait partie de la routine **CLOSE** en $FFC3.

$F3EF ICLALL FERMETURE VECTORISEE DES FICHIERS
Referme tous les fichiers logiques. Point d'entrée après le vecteur en RAM en $032C. Fait partie de la routine **CLALL** en $FFE7.

$F3F3 ICLRCH RESTAURE CANAUX D'ENTREE/SORTIE (VECTORISE)
Restaure les canaux d'entrée/sortie par défaut (clavier et écran). Point d'entrée après le vecteur en RAM en $0322. Fait partie de la routine **CLRCHN** en $FFCC.

$F40A IOPEN ENTREE VECTORISEE ROUTINE OPEN
Ouverture d'un fichier logique. Point d'entrée après le vecteur en RAM en $031A. Fait partie de la routine **OPEN** en $FFC0.

$F542 LOAD EXECUTE LOAD/VERIFY
En entrée, le numéro de périphérique est en $BA, le registre A contient 0 pour LOAD ou 1 pour VERIFY, l'adresse secondaire est en $B9. En sortie, X (bas) et Y (haut) contiennent l'adresse de fin de zone lue + 1.

$F549 ILOAD CHARGEMENT VECTORISE D'UNE ZONE
Charge une zone mémoire par le canal d'entrée. Point d'entrée après le vecteur en RAM en $0330. Fait partie de la routine **LOAD** en $FFD5.

$F675 SAVE EXECUTE SAVE
En entrée, $BA contient le numéro de périphérique, $B9 l'adresse secondaire, le registre A contient l'adresse en page 0 du pointeur vers le premier octet à sauver. L'adresse du dernier octet à sauver + 1 est dans X (bas) et Y (haut).

$F685 ISAVE SAUVETAGE VECTORISE D'UNE ZONE
Sauve une zone mémoire sur le canal de sortie. Point d'entrée après le vecteur en RAM en $0332. Fait partie de la routine **SAVE** en $FFD8.

$F770 ISTOP TEST VECTORISE DE LA TOUCHE STOP
Teste la touche STOP. Point d'entrée après le vecteur en RAM en $0328. Fait partie de la routine **STOP** en $FFE1.

$F7AF FHEAD RECHERCHE UN BLOC TITRE
Recherche un bloc titre sur cassette (**HEADER** bloc). S'arrête après avoir trouvé un bloc titre "PROGRAMME" ou un bloc titre "DONNEES". En sortie, le drapeau CARRY = 0 si trouvé, le registre A = 0 si **STOP** a été enfoncé au clavier.

$F8AB SENSE LIT INTERRUPTEUR CASSETTE

$F8B7 RECPL ROUTINE CASSETTE
Attend l'enfoncement de "RECORD & PLAY".

$F8C0 RHEAD LIT BLOC TITRE
Partie de la routine de lecture cassette.

$F8C9 RLOAD LIT BLOC "PROGRAMME"
Partie de la routine LOAD sur cassette.

$F8E3 WHEAD ECRIT BLOC TITRE
Partie de la routine SAVE sur cassette.

$F8F4 TAPE ROUTINE CASSETTE
Partie des commandes cassettes.

$F95D SETTO ROUTINE CASSETTE
Prépare le délai d'attente entre deux bits sur cassette.

$F98E CREAD LIT OCTET CASSETTE
Lit un octet en provenance de la cassette. En sortie, l'octet lu est en $BF .

$FABD BYHAND GESTION D'OCTET APRES LECTURE CASSETTE
En entrée, l'octet lu sur la cassette est en $BD. En sortie, l'octet $A8 <>0 si l'octet lu est faux;$A9<>0 si l'octet lu est 0; Le bit 7 de $AA indique s'il vaut 1 qu'il faut ignorer les octets lus jusqu'à ce que $A9 soit <>0; Sinon $AA est utilisé en décompteur après synchronisation de la cassette;Si $93<>0,effectue vérification et pas LOAD; $BE indique premier ou second bloc; $9E est l'index dans la table des erreurs pour le premier bloc; $9F est l'index dans la table d'erreurs pour le second bloc.

$FBEA WCASS ECRITURE SUR CASSETTE
Ecrit un bloc sur cassette. $BE est le compteur de blocs. Si $BE = 2 : bloc titre; Si $BE = 0 : second bloc.

$FEAD NMINV ENTREE VECTORISEE DE ROUTINE NMI
Gestion de l'interruption NMI après le vecteur en RAM en $0318.

$FED2 CBINV ENTREE VECTORISEE DE LA ROUTINE DE BREAK
Entrée dans la routine de BREAK par le vecteur en RAM en $0316. Egalement entrée pour le vecteur en RAM en $032E de la commande BASIC USR par défaut.

$FF22 RESET ROUTINE DE DEMARRAGE DU VIC
Réinitialise tout le VIC en testant la présence d'une **ROM AUTO-START** en $A000. Ne rend jamais le contrôle au programme appelant !

$FD3F AUTOTS TESTE LA PRESENCE DE ROM AUTO-START
Appel par JSR $FD3F. En sortie,le drapeau Z vaut 1 si une ROM AUTO-START est présente.

$FD8D MEMTST TEST MEMOIRE
Première routine appelée à l'initialisation du VIC. Remet à zéro toute la page 0, ainsi que les octets de $0200 à $03FF. Teste la présence de RAM par **RAMTST** à partir de $0400, puis initialise les pointeurs de bas et haut de mémoire RAM disponible. S'il n'y a pas de RAM aux adresses $2000 à $2100, l'écran est positionné en $1E00.

$FDF9 INIVIA INITIALISE LES VIA
Nécessite 6 emplacements dans la pile. Initialise les VIA : registres VIA1-2,3,8,C,E,F et VIA2-2,3,4,5,8,C,E. Appelle $EF8D.

$FE91 RAMTST TESTE SI UN OCTET EST EN RAM
En entrée,l'adresse de l'octet à tester doit être dans ($C1, $C2) (bas, haut). En sortie, l'octet en mémoire est inchangé, s'il est en RAM. Dans ce cas, il y a $18 dans le registre A et le carry est à 0; si l'octet est en ROM ou inexistant, le carry vaut 1. Le test s'effectue par sauvegarde de la valeur, écriture et vérification des octets $AA et $55 ,puis réécriture de la valeur correcte.

$FEA9 NMI ROUTINE D'INTERRUPTION NON MASQUABLE
Interdit les interruptions puis effectue un saut indirect à l'adresse en **VECNMI** ($0318). Normalement, la routine continue en $FEAD. Suivant les cas, elle gère soit le bus série IEEE ou l'RS232, soit elle effectue un redémarrage à chaud de la ROM AUTO-START ou de la ROM BASIC par JMP ($A002) ou JMP ($C002). Ce redémarrage n'a lieu que lors de l'enfoncement de **STOP** et **RESTORE** au clavier.

$FEC4 WARMAO JMP ($A002)
Effectue le redémarrage à chaud (warm start) de la ROM AUTO-START en $A000.

$FF56 RPULL FIN DE ROUTINE D'INTERRUPTION
Enlève de la pile (dans l'ordre), les contenus de Y, X, A puis execute un RTI. Est appelé en fin de routine NMI.

$FF72 BRKO ROUTINE DE BREAK/INTERRUPTION
Routine exécutée lors d'un BRK (opcode de l'octet 0) ou lors d'une interruption. Lors d'une interruption, un JMP indirect en RAM est exécuté en **VECINT** en $0314. Lors d'un BREAK, un JMP indirect en RAM est exécuté en $0316 (**VECBRK**). Emploie 3 emplacements dans la pile avant d'exécuter le JMP indirect pour y sauver dans l'ordre A, X, Y.

$FF8A OLDIO RETABLIT VECTEURS EN RAM PAR DEFAUT

Nécessite 2 emplacements dans la pile. Ne modifie aucun registre. Appel par JSR $FF8A. Rétablit les vecteurs en RAM originaux comme à la mise sous tension (Voir les valeurs des vecteurs dans la carte mémoire. Ils sont situés entre $0314 et $0333).

$FF8D SETVEC LIT OU IMPOSE LES VECTEURS EN RAM

Nécessite 2 emplacements dans la pile. Emploie X et Y. En entrée, si le bit CARRY est à un, SETVEC lit la valeur des vecteurs en RAM en $0314 à $0335 et les écrit dans une table à l'adresse (Y, X) (bas, haut). En entrée, si le bit CARRY vaut 0, SETVEC écrit en RAM en $0314 à $0335 les vecteurs qui sont dans la table à l'adresse (X,Y).

$FF90 SETMSG MODE 'DIRECT' OU MODE 'RUN'

Emploie le registre A. Nécessite 2 emplacements dans la pile. En entrée, le contenu de A est stocké à l'adresse $9D. Si le bit 7 = 1, ceci indique l'on est en mode direct, sinon on est en mode RUN : Les messages comme : "PRESS PLAY",etc.. sont différents dans les 2 modes. En sortie, le registre A et les drapeaux contiennent la valeur de l'octet ST-STATUS en $90. La valeur 0 indique qu'il n'y a pas d'erreur.

$FF93 SECOND ENVOIE ADRESSE SECONDAIRE

Nécessite 8 emplacements dans la pile. En entrée, le registre A indique l'ADRESSE SECONDAIRE à envoyer (0-31). A employer après LISTEN en $FFB1. A ne pas employer après TALK en $FFB4. En sortie, l'octet ST-STATUS contient l'erreur éventuelle en $90.

$FF96 STALK ENVOIE ADRESSE SECONDAIRE APRES TALK

Nécessite 8 emplacements dans la pile. En entrée, le registre A contient l'adresse secondaire à envoyer (0-31) sur le bus série IEEE. Doit être précédé de 'TALK' (JSR $FFB4). En sortie, l'octet ST-STATUS contient l'erreur éventuelle en $90.

$FF99 MEMTOP LIT OU DEFINIT SOMMET DE MEMOIRE

Nécessite 2 emplacements dans la pile. Emploie X et Y. En entrée, si le bit CARRY = 1, le pointeur de sommet de mémoire est lu en mémoire en $37 et $38 et revient dans X et et y (bas,haut). Si le bit CARRY = 0, c'est l'inverse.

$FF9C MEMBOT LIT OU DEFINIT BAS DE MEMOIRE

Nécessite 2 emplacements dans la pile. Emploie X et Y. En entrée, si le bit CARRY = 1, le pointeur de bas de mémoire en $2B et $2C est lu et se retrouve en sortie dans X et Y (bas, haut). Si le bit CARRY = 0, c'est l'inverse.

$FF9F SCNKEY LECTURE CLAVIER

Nécessite 4 emplacements dans la pile. En sortie, si une touche est enfoncée, sa valeur ASCII est rangée dans le tampon de clavier KBDBUF en $0277-$0280 dont la taille est en KBDMAX ($0389). Le pointeur KBDPTR ($C6) est est alors incrémenté. Cette routine est exécutée 60 fois par seconde lors des interruptions INT.

Usage : Si la routine d'interruption est supprimée, on peut lire le clavier par :

```
WAIT  JSR SCNKEY
      JSR CHRIN
      CMP £$00
      BEQ WAIT
      RTS
```

qui attendra une frappe au clavier avant de rendre le contrôle au programme appellant.

$FFA2 SETTMO ETABLIT LE DELAI IEEE (TIME-OUT)

Nécessite 2 emplacements dans la pile. En entrée, si le registre A a son bit 7 à 1, le délai est supprimé. Si le bit 7 est à 0, le délai est pris en compte. Le délai de TIME-OUT est le délai pendant lequel le VIC attend une réponse d'un périphérique IEEE. Ce délai dépassé, le VIC considère que le périphérique est absent. Par exemple, le lecteur de disquettes VIC-1540 laisse passer ce délai sans réagir si un fichier demandé n'est pas trouvé : Le VIC affiche alors FILE NOT FOUND. Le délai est de 64 millisecondes.

$FFA5 ACPTR ENTREE D'UN OCTET SUR IEEE SERIE

Nécessite 13 emplacements dans la pile. Affecte l'octet ST-STATUS (Voir détail en routine $FFB7). Doit être précédé d'appels à $FFB4 et $FF96 (TALK et adresse secondaire). En sortie, l'octet lu est dans le registre A.

$FFA8 CIOUT ENVOIE UN OCTET SUR IEEE SERIE

Nécessite 8 emplacements dans la pile. Doit être précédé d'appels à LISTEN et éventuellement à SECOND. En entrée, le caractère à transmettre est dans le registre A. En sortie, l'octet ST-STATUS contient l'erreur éventuelle. Attention, la routine CIOUT bloque un caractère. A chaque appel, elle envoie le caractère reçu la fois précédente. Le dernier caractère est envoyé lors de l'appel de UNLSTN en fin de transmission.

$FFAB UNTLK ENVOIE UNTALK SUR L' IEEE SERIE

Nécessite 6 emplacements dans la pile. Pas d'erreur signalée en sortie.

$FFAE UNLSTN ENVOIE UNLISTEN SUR L'IEEE SERIE

Nécessite 6 emplacements dans la pile. Pas d'erreur signalée en sortie.

$FFB1 LISTEN ENVOIE LISTEN SUR L'IEEE SERIE

Nécessite 8 emplacements dans la pile. En entrée,le registre A contient le numéro de périphérique (0-31). En sortie, l'octet ST-STATUS contient l'erreur éventuelle.

$FFB4 TALK ENVOIE TALK SUR L'IEEE SERIE

Nécessite 4 emplacements dans la pile. En entrée, le registre A contient le numéro de périphérique (0-31). En sortie, l'octet ST-STATUS contient l'erreur éventuelle.

$FFB7 READST LECTURE DE L'OCTET ST-STATUS

Nécessite 2 emplacements dans la pile. En sortie, le registre A contient la valeur du STATUS en $90. Le STATUS est le descripteur de l'état actuel des entrées/sorties. Il faut vérifier qu'il n'indique aucune erreur (STATUS=0), surtout après avoir ouvert un nouveau canal de communication (RS-232,IEEE série, cassette).
Les erreurs sont indiquées par la valeur 1 d'un des bits du STATUS.

En **lecture cassette**,les erreurs possibles sont :

BIT 2 : Bloc lu trop court
BIT 3 : Bloc lu trop long
BIT 4 : Erreur irrécupérable (LOAD ERROR)
BIT 5 : Erreur de somme de contôle (CHECKSUM ERROR)
BIT 6 : Fin de fichier rencontrée (END OF FILE)
BIT 7 : Fin de bande rencontrée (END OF FILE)

En **IEEE SERIE**, les erreurs possibles sont :

BIT 0 : Délai dépassé en ecriture (TIME-OUT)
BIT 1 : Délai dépassé en lecture (TIME-OUT)
BIT 6 : Signal EOI reçu (END OR IDENTIFY) : On a reçu le dernier octet d'un fichier.
BIT 7 : Périphérique non connecté (DEVICE NOT PRESENT)

$FFBA SETLFS ETABLIT NUMERO DE FICHIER, ADRESSE PRIMAIRE ET SECONDAIRE

Nécessite 2 emplacements dans la pile. Emploie les registres A,X et Y. En entrée, le numéro de fichier est dans le A (0-255), le numéro de périphérique dans X (0-31), l'adresse secondaire dans Y (255 si pas d'adresse,0 pour SAVE normal sur cassette). Il ne peut y avoir plus de 15 fichiers ouverts, et un seul vers l'RS232.

Les numéros de périphériques sont :

0 = CLAVIER
1 = CASSETTE
2 = RS232
3 = ECRAN
4 = IMPRIMANTE COMMODORE
8,9,10,11 = DISQUETTE 1540
9 = MODEM

Les numéros de périphériques supérieurs à 3 sont d'office considérés comme périphériques IEEE. Doit être précédée de la routine OPEN.

$FFBD SETNAM ETABLIT LE NOM D'UN FICHIER

Nécessite 4 emplacements dans la pile. Emploie les registres A, X et Y. Utilisé pour ouvrir un fichier avec un nom (cassette,disque, contrôle RS232). En entrée, le registre A contient la longueur du nom du fichier (ou sinon 0), les registres X et Y (bas, haut) l'adresse de la chaîne de caractères du nom. Doit être employée avant OPEN.

$FFC0 OPEN OUVRE UN FICHIER D'ENTREE/SORTIE

La routine OPEN passe par un vecteur en RAM en ($031A) où on peut l'intercepter. Elle continue en $F40A. Nécessite 8 emplacements dans la pile. Doit être précédée de SETLFS et SETNAM.

Exemple: pour initialiser une disquette sur le lecteur 1540:

```
        ; open 15,8,15,"IO"
INI     LDA £$2
        LDX £>NAME
        LDY £<NAME
        JSR SETNAM
        LDA £15
        LDX £8
        LDY £15
        JSR SETLFS
        JSR OPEN
        RTS
NAME    .BYTE 'I','O'
```

$FFC3 CLOSE FERME UN FICHIER D'ENTREE/SORTIE

Passe par un vecteur en RAM en $031C, puis continue en $F34C. Nécessite 6 emplacements dans la pile. En entrée, le registre A contient le numéro du fichier à fermer. Attention, toute donnée non envoyée ou non reçue sur le canal refermé est perdue.

$FFC6 CHKIN ATTRIBUE LE CANAL D'ENTREE A UN FICHIER OUVERT

Passe par le vecteur en RAM en $031E, puis continue en $F2C7. Nécessite 8 emplacements dans la pile. Emploie le registre X. En entrée, X contient le numéro de fichier. Employer cette routine avant chaque accès ou groupe d'accès à un périphéri-

que différent du clavier. Comme il n'existe qu'une routine d'entrée de caractères, à un moment donné, un seul périphérique peut envoyer un caractère : celui dont le canal est ouvert. Par défaut, c'est le clavier qui est choisi comme entrée. Pour lire un caractère, sur un autre périphérique, il faut préciser ses caractéristiques par un OPEN, puis ouvrir le canal par CHKIN et lire les données; Si des caractères doivent arriver d'un autre périphérique, refermer le canal pour le périphérique en cours par CLRCHN et réouvrir le canal pour l'autre périphérique. Quand tous les accès sont terminés pour un fichier, on le referme par CLOSE.

$FFC9 CHKOUT ATTRIBUE LE CANAL DE SORTIE A UN FICHIER OUVERT

Passe par le vecteur en RAM en $0320, puis revient en $F303. Nécessite 8 emplacements dans la pile. En entrée, le registre X contient le numéro de fichier. Employer cette routine avant les accès, refermer le canal par CLRCHN et éventuellement le fichier par CLOSE.

$FFCC CLRCHN REFERME LES CANAUX D'ENTREE/SORTIE

Nécessite 6 emplacements dans la pile. En entrée, pas de paramètres. En sortie, le périphérique d'entrée est à nouveau le clavier et celui de sortie l'écran. Cette routine envoie UNTALK et UNLISTEN sur le bus séie IEEE. Si on omet volontairement d'appeler cette routine après avoir ouvert un canal d'entrée disque, et avant d'ouvrir un canal de sortie imprimante, les données du disque iront directement à l'imprimante. (SPOOLING)

$FFCF CHRIN ACQUIERT UN CARACTERE SUR CANAL D'ENTREE

Passe par le vecteur en RAM en $0324, puis continue en $F20E. Nécessite 8 emplacements dans la pile. En sortie, le caractère est dans A et les erreurs éventuelles dans l'octet ST-STATUS en $90. Si on n'a pas précédemment utilisé CHKIN, le caractère vient du clavier. Au premier appel à CHRIN, le curseur apparaît et reste visible jusqu'à la frappe d'un RETURN. Les appels successifs rendent les caractères suivants de la ligne. Lors du dernier appel, le RETURN est rendu dans le registre A et le curseur disparaît.

$FFD2 CHROUT ENVOIE UN CARACTERE SUR CANAL DE SORTIE

Passe par le vecteur en RAM en $0326, puis reprend en $F27A. Est une des routines les plus employées de toutes celles du KERNAL. Nécessite 8 emplacements dans la pile. En entrée, le caractère à transmettre est dans le registre A. Si on n'a pas précédemment utilisé CHKOUT, le caractère va à l'endroit du curseur, qui avance d'une case et éventuellement décale l'écran vers le haut. Si plusieurs canaux de sortie ont été successivement ouverts vers divers périphériques , seul le dernier reçevra le caractère envoyé.

$FFD5 LOAD CHARGE UN BLOC DE DONNEES

Emploie 10 emplacements dans la pile. Passe par le vecteur en RAM en $0330 après avoir déposé en RAM en $C3 et en $C4 le contenu des registres X et Y (adresse de chargement bas, haut). En entrée, si le le registre A = 0, il y a chargement en mémoire ;si A = 1, vérification. Les registres X et Y contiennent l'adresse de chargement (bas, haut) si l'adresse secondaire spécifiée est 3. Dans le cas où l'adresse secondaire est 1 (comme dans LOAD "X",1,1 en BASIC), les données sont chargées à l'adresse précisée dans le bloc de donnés lui-même (cassette ou disque). Si l'adresse secondaire est différente de 1, le KERNAL charge les données à l'adresse qui se trouve dans X,Y qui est en général le début de zone BASIC (pointeur en RAM en $002B). En sortie, les erreurs sont dans l'octet ST-STATUS. (vérifier la valeur du ST-STATUS pour être certain d'un chargement correct)

$FFD8 SAVE ENVOIE BLOC DE DONNEES

Nécessite 8 emplacements dans la pile (10 si cassette). En entrée, X et Y (bas,haut) indiquent la fin de zone à sauver. Le pointeur de début de zone à sauver est en $2B,$2C. On peut le modifier par MEMBOT au préalable . Doit être précédé de SETLFS et (éventuellement) SETNAM. En sortie, les erreurs éventuelles sont dans l'octet ST-STATUS.
Si SETLFS a installé une adresse secondaire 1 ou 3 pour un sauver un programme sur cassette, le bloc de préambule(BLOC TITRE) contiendra un drapeau indiquant que le fichier doit toujours être relu (par LOAD) à l'adresse absolue dont il provient.

$FFDB SETTIM MISE A L'HEURE HORLOGE

Ne nécessite pas d'emplacement dans la pile. En entrée, A contient le poids fort, X le suivant et Y le poids faible de la valeur à charger dans l'horloge (variables BASIC **TI** et **TI$**). Les valeurs sont écrites en $A0 à $A2 et représentent la valeur du temps en 1/60 ème de seconde. La précision est de 1 seconde par jour environ, mais dépend en fait des périodes où les interruptions sont interdites. Si elles sont très fréquentes et de longue durée (accès cassette par exemple), l'horloge retarde notablement. Il est nécessaire d'employer cette routine et non d'écrire directement la valeur du temps en $A0 car ces adresses sont en permanences testées et modifiées par une routine d'interruption. Sans passer par SETTIM, on peut ainsi obtenir une heure tout à fait fausse, par suite d'interférence entre le programme principal et le programme d'interruption.

$FFDE RDTIM LIT L'HORLOGE

Ne nécessite pas d'emplacement dans la pile. En sortie A, X et Y contiennent l'heure en 1/60 ème de seconde (voir SETTIM). La valeur lue est toujours comprise entre 0 et 5184000, soit 24 heures.

$FFE1 STOP TESTE LA TOUCHE STOP

Passe par le vecteur en RAM en $0328,puis continue en $F770. En sortie, le drapeau Z vaut 1 si la touche STOP est enfoncée et le registre A permet de tester l'enfoncement d'une ou plusieurs autres touches de la même rangée du clavier. Un bit à 0 signifie que la touche est enfoncée. Cette valeur est rangée en $91 (145).
A noter : si STOP est enfoncé, le pointeur du tampon clavier est remis à 0.

BIT 0 : STOP REGISTRE A = PEEK (145)
BIT 1 : SHIFT GAUCHE
BIT 2 : X
BIT 3 : V
BIT 4 : N
BIT 5 : < ,
BIT 6 : ? /
BIT 7 : CURSEUR EN BAS

Par exemple, pour tester le ? : IF PEEK(145) = 191 THEN...

$FFE4 GETIN LECTURE CLAVIER

Nécessite 8 emplacements dans la pile. Passe par le vecteur en RAM en $032A, puis continue en $F1F5. En sortie, le caractère lu est dans A; S'il n'y avait pas de caractère dans le tampon, A = 0. Les caractères sont déposés dans le tampon par la routine SCNKEY. La taille maxi du tampon est définie par la valeur en $0289 qui vaut normalement 10. Cette valeur peut être modifiée à volonté entre 1 et 15 (0 = plus de clavier, 16 et plus = on écrase des pointeurs indispensables au KERNAL).

$FFE7 CLALL REFERME TOUS LES FICHIERS

Nécessite 11 emplacements dans la pile. Pas d'erreur indiquées en sortie. Referme les fichiers et rétablit les canaux d'entrée/sortie par défaut (clavier/écran). Passe par le vecteur en RAM en $032C, puis continue en $F3EF.

$FFEA UPCLK INCREMENTE L'HORLOGE

Nécessite 2 emplacements dans la pile. Est appelé 60 fois par seconde lors des interruptions. A appeler régulièrement quand on supprime les interruptions pour tester la touche STOP et maintenir une certaine précision de l'horloge.

$FFED SCREEN LIT LA TAILLE DE L'ECRAN

Nécessite 2 emplacements dans la pile. En sortie, le registre X contient le nombre de colonnes et Y le nombre de lignes actuelles de l'écran. (En effet, la taille de l'écran est variable, voir au chapitre 1).

$FFF0 PLOT LIT OU IMPOSE POSITION CURSEUR

Nécessite 2 emplacements dans la pile. Si en entrée le bit CARRY vaut 1 , les coordonnées horizontale et verticale du curseur (à partir du coin supérieur gauche) sont en sortie dans X et Y. Si en entrée, le bit CARRY vaut 0 , les valeurs X et Y deviennent en sortie les nouvelles coordonnées du curseur (X en RAM en $D6, Y en $D3), par appel de CURSOR en $E587.

Exemple: POKE 214,X : POKE 211,Y : SYS 65520

$FFF3 IOBASE REND L'ADRESSE DES VIA

Met dans X et Y l'adresse du VIA numéro 1, soit $9110.

```
IOBASE LDX £$10
       LDY £$91
       RTS
```

CHAPITRE 3

EXTENSIONS MATERIELLES.

Le VIC est muni de nombreux connecteurs d'extensions, mais celui qui retient le plus l'attention est le connecteur d'expansion BUS, qui permet d'ajouter au VIC de très nombreux dispositifs : extensions mémoires, périphériques, etc. Outre les cartouches de programmes ROM et de mémoires RAM vendues par COMMODORE même, il existe une grande variété d'extensions sur le marché. On en trouvera un aperçu en annexe. Nous allons décrire à titre exemplatif trois cartes d'extensions mémoire relativement simples à réaliser soi-même. Le tirage des circuits imprimés (C.I.) à partir du livre nécessite du matériel photographique. Si vous ne disposez pas d'un tel matériel, vous pouvez vous procurer les C.I. seuls ou les ensembles de composants chez BCM. Les circuits imprimés sont, pour la facilité de réalisation, conçus en double façe sans métallisation des trous. C'est-à-dire que les connexions indispensables entre les deux faces du circuit doivent être réalisées manuellement en soudant un petit morceau de fil de cuivre nu dans tous les trous du C.I. qui ne servent pas à l'insertion de composants. Les connexions entre faces sont expressément séparées des broches des composants pour permettre le test de la carte seule avant l'insertion des divers éléments.

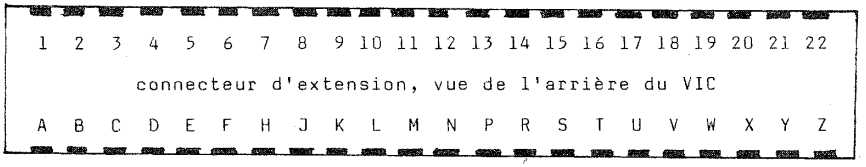
Procédure générale

1. Se procurer ou réaliser le circuit imprimé double face.
2. Se procurer les composants.
3. Percer tous les trous de la carte au diamètre 0,8 mm.
4. Agrandir les trous pour les interrupteurs et boutons pousssoirs à 1,5 mm.
5. Tester que toutes les pistes conduisent le courant sur toute leur longueur et sont isolées entre elles.
6. Réaliser toutes les connexions entre les 2 faces du circuit et retester le tout comme au point 5.
7. Essayer la carte seule sur le VIC : il doit toujours fonctionner normalement.
8. Installer les composants en utilisant des supports de bonne qualité au moins pour les mémoires ROM. Faire attention à l'orientation des circuits intégrés et des condensateurs polarisés.
9. Isoler par une feuille de plastique la face inférieure de la carte, puis la tester dans le VIC.
10. Si un problème quelconque se pose, revérifier tout - c'est bien plus facile si les composants sont montés sur support - et reprendre au point 9.
11. Rendre définitive la protection du dessous du circuit.

Pour le montage, on utilisera un fer à souder à pointe fine d'une puissance de 20 Watts environ pour ne pas surchauffer les circuits intégrés.

Le connecteur d'extension.

Le connecteur d'extension donne accès à tous les bus du VIC 20 : adresses, données et contrôle. Le bus d'adresse n'est pas complètement présent sur le connecteur : seuls y arrivent les fils A0 à A13. A14 et A15 ne sont pas présents mais ce n'est cependant pas gênant car le décodage partiel des adresses a déjà été effectué dans le VIC et les sorties du décodeur d'adresses sont disponibles sur le connecteur.



FACE SUPERIEURE			FACE INFERIEURE		
Broche	Fonction		Broche	Fonction	
1	masse		A	masse	
2	D0	bus de données à 8 bits	B	A0	bus d'adresses à 16 bits dont 14 disponibles
3	D1		C	A1	
4	D2		D	A2	
5	D3		E	A3	
6	D4		F	A4	
7	D5		H	A5	
8	D6		J	A6	
9	D7		K	A7	
10	BLK 1		L	A8	
11	BLK 2		M	A9	
12	BLK 3		N	A10	
13	BLK 5		P	A11	
14	RAM 1		R	A12	
15	RAM 2		S	A13	
16	RAM 3		T	I/O 2	
17	VR/W		U	I/O 3	
18	CR/W		V	PHI 2	
19	IRQ		W	NMI	
20	rien		X	RESET	
21	+5 V		Y	rien	
22	masse		Z	masse	

Les signaux D0 à D7, A0 à A13, + 5V et masse représentent données, adresses et alimentation. Attention, sur n'importe quelle carte d'extension, il faut au moins un condensateur d'environ 0,2 microfarads entre +5 V et masse pour régulariser la tension d'alimentation qui peut être perturbée par la résistance des contacts au niveau du connecteur.

Les signaux BLK 1, BLK 2, BLK 3, BLK 5, RAM 1, RAM 2, RAM 3, I/O 2 et I/O 3 sont des sorties du décodeur d'adresses du VIC. Parmi ces signaux, un seul à la fois peut valoir 0, ce qui correspond à la sélection d'une zone mémoire. Les zones mémoires concernées et leur usage sont décrits dans la carte-mémoire au chapitre 5.

Signal	Si = 0, sélectionne la zone :			Taille
BLK 1	$2000	à	$3FFF	8 K
BLK 2	$4000	à	$5FFF	8 K
BLK 3	$6000	à	$7FFF	8 K
BLK 5	$A000	à	$BFFF	8 K
RAM 1	$0400	à	$07FF	1 K
RAM 2	$0800	à	$0BFF	1 K
RAM 3	$0C00	à	$0FFF	1 K
I/O 2	$9800	à	$9BFF	1 K
I/O 3	$9C00	à	$9FFF	1 K

On voit que, grâce à ces signaux, on peut accéder à n'importe quelle zone de mémoire. La sélection d'une adresse précise dans un de ces blocs se réalise grâce aux 14 fils d'adresses disponibles.

Les fils IRQ, NMI et RESET sont les trois entrées d'interruption, interruption non masquable et redémarrage du 6502. Ces lignes sont actives à l'état 0, c'est-à-dire que normalement ces fils sont à l'état + 5V. Pour provoquer un **RESET** par exemple, il faut raccorder momentanément le fil **RESET** à la masse. Comme il existe dans le système plusieurs circuits susceptibles de mettre à 0 l'un de ces fils, on ne peut donc utiliser pour cela que des interrupteurs manuels (exemple : poussoir RESET) ou des circuits dits "à collecteur ouvert" tels le 74LS05 : ce sont des circuits qui peuvent imposer à leur sortie un niveau 0 mais pas un niveau 1. Ce dernier est dû à une résistance (d'environ 4K) câblée dans le VIC entre chacun des 3 fils considérés et le + 5 Volts.

Le signal PHI 2 est l'horloge du microprocesseur à 1,1 MHz. Quand ce signal vaut 1, le 6502 accède à la mémoire.

Les signaux CR/W et VR/W indiquent le sens du dialogue entre 6502 et mémoire. L'état 1 signifie lecture c'est-à-dire mémoire --> 6502, l'état 0 signifie écriture c'est-à-dire 6502 --> mémoire. La différence entre ces deux signaux est la suivante : le 6502 génère le signal CR/W, mais on sait qu'il n'accède à la mémoire que quand PHI 2 est à 1. Or on constate que le signal CR/W est déjà disponible avant que PHI 2 n'arrive à l'état 1. Donc, pour éviter des phénomènes transitoires indésirables, le VIC recrée un signal VR/W qui, lui, ne varie qu'en synchronisme avec PHI 2. En pratique, on n'utilise

que VR/W pour contrôler la lecture ou l'écriture en mémoire.

LA CARTE ROM.

Cette carte est la plus simple des trois que nous décrirons ici. On peut l'utiliser par exemple avec la ROM AUTOBASIC ou le moniteur 6502, ou avec une des mémoires VICKIT de STACK ou tout autre programme que vous écrirez vous-même en ROM (2 ou 4 K) ou même en RAM (s'il ne dépasse pas 2 K).

Les fonctions de la carte ROM sont :

1. poussoir de RESET

2. 2K ou 4K de mémoire dans un des blocs 1, 2, 3 ou 5. La sélection du bloc se fait grâce à une connection à câbler sur la carte (sur la face inférieure).

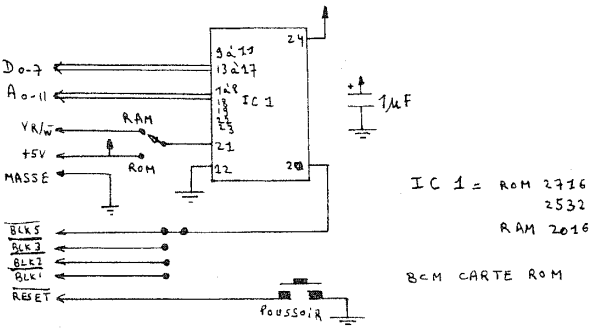
3. 2K de mémoire RAM dans un des blocs 1, 2, 3 ou 5. Il est possible de transformer la carte ROM en carte RAM en modifiant une connection sur la carte (sur la face inférieure).

On peut voir sur le schéma de principe comme sur le dessin du circuit les connexions optionnelles pour le choix de l'adresse et du type de mémoire.

Notes de construction :

- Il y a 7 trous de diamètre 1,5 mm pour le poussoir et l'inverseur ROM/RAM.
- L'emplacement du condensateur de découplage est noté + et -.
- Le condensateur sera de préférence au tantale, tension de service 12 volts minimum.
- Il y a 13 trous servant de liaisons entre les 2 faces du circuit.
- La broche 1 de la mémoire ROM ou RAM est indiquée par le chiffre 1 sur la face supérieure de la carte.
- Le connecteur s'enfoncera correctement si la carte est découpée en suivant l'intérieur du trait qui entoure la face supérieure. Toutes les broches du connecteur sont reliées entre elles sur le cliché. Ceci pour permettre la dorure éventuelle du connecteur pour diminuer la résistance de contact. La dorure étant un procédé électrolytique, il est nécessaire d'avoir une électrode commune.

l'échelle des clichés ne peut être donnée de façon précise à cause des procédés de reproduction en imprimerie. On peut vérifier l'échelle grâce au pas du connecteur de bord de la carte. De centre à centre de 2 connexions successives, il y a exactement 3,96 mm. De plus, entre le bord gauche de la première connexion et le bord droit de la dernière on mesure **exactement 85 mm.**



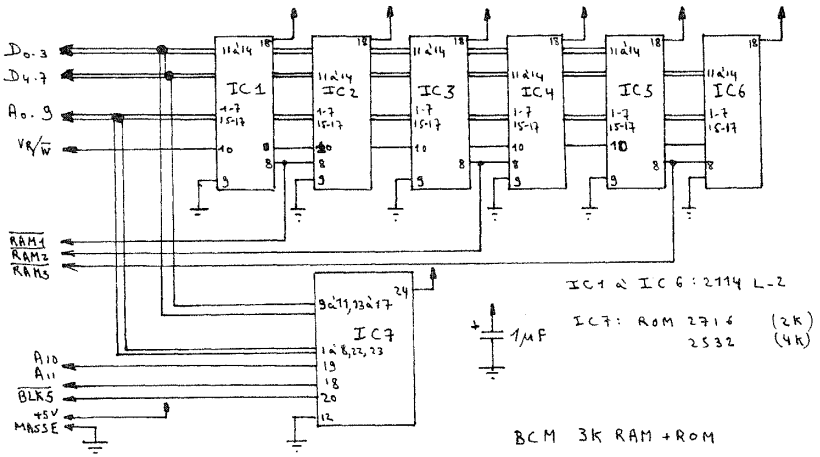
LA CARTE 3K RAM/ROM.

La carte 3K RAM/ROM est nettement plus complexe que la précédente, mais son principe est aisé à comprendre. Elle comprend une mémoire ROM de 2 ou 4K dans le bloc 5, c'est-à-dire que cette mémoire est vue du 6502 indifférement comme étant en $A000 à $AFFF ou en $B000 à $BFFF. Cette carte peut donc être utilisée pour le moniteur 6502 en $A000 ou pour le VICKIT II en $8000, par exemple. Les 3K de mémoire RAM occupent les blocs RAM 1, RAM 2 et RAM 3. Tous les décodages d'adresses sont déjà effectués dans le VIC. Les 3 K RAM se répartissent dans 6 circuits 2114-L2, chacun contenant 1K*4 bits.

Le montage se fera en respectant la procédure décrite ci-dessus. Il faut apporter un soin tout particulier au test de la carte après avoir connecté les deux faces à tous les emplacements (trous) raccordés à des pistes cuivrées sur la face supérieure. (Il y en a 66). Il faut tester aussi bien la continuité de chaque piste que les isolations entre pistes voisines.

Notes de construction :

- Le condensateur de découplage peut avoir une valeur comprise entre 0,5 et 5 microfarads. On utilisera de préférence un modèle au tantale avec une tension de service de 12 volts minimum. Le modèle au tantale étant polarisé, on vérifiera bien que le côté + soit orienté vers le connecteur.
- Les broches 1 des 2114 (l'encoche dans le boîtier) doivent être orientées à l'opposé du connecteur.
- La broche 1 du circuit ROM 24 broches est repérée par un 0 sur la face supérieure de la carte.
- Ne pas oublier d'isoler la face inférieure de la carte. En effet, la carte se trouve à quelques millimètres du circuit imprimé principal du VIC.
- L'échelle des clichés est identique à celle de la carte ROM.
- La découpe de la carte aura lieu en suivant l'intérieur du trait entourant la face supérieure.



La carte 8K RAM/ROM + 'WATCHDOG'.

La carte 8 K est une carte à double fonction. Deux circuits complètement indépendants sont présents sur la carte : une mémoire 8 K et un circuit de redémarrage automatique ou chien de garde ('WATCHDOG'). Décrivons tout d'abord ceui-ci : une minuterie monostable est déclenchée à l'allumage du VIC. Après environ 35 secondes, le délai étant écoulé, la minuterie envoie une impulsion sur le fil 'RESET' , ce qui redémarre le VIC automatiquement comme à la mise sous tension. Une différence est toutefois notable : la mémoire RAM n'est pas effacée par cette procédure; seuls les pointeurs du BASIC et du KERNAL sont réinitialisés. Ce n'est pas tout : si le 6502 envoie expressément un signal sur le fil I/03 (par POKE 40000,0, par exemple), la minuterie redémarre à zéro.

Si le but de la manoeuvre ne vous paraît pas clair, il est cependant extrêmement pratique : le VIC contrôle tout seul le fonctionnement correct des programmes qu'il exécute. En effet, si le programme que le VIC exécute envoie régulièrement le signal de redémarrage à la minuterie, rien ne se passe et tout va bien. Si, pour une raison quelconque, le programme cale et cesse de s'exécuter correctement, le VIC redémarre la procédure à zéro au plus tard une quarantaine de secondes après l'erreur. On voit tout de suite l'avantage de ce système : avec ce dispositif et un programme à chargement automatique (AUTO-BASIC en BASIC ou AUTOSTART en langage machine), le VIC devient un contrôleur de processus industriel parfaitement fiable et d'un prix de revient sans concurrence !

Le but premier de la carte 8K est donc le contrôle de processus. La mémoire adjointe est configurable de différentes façons :

- Soit 8K RAM en $2000 pour donner 11775 BYTES FREE en BASIC.
- Soit 8K ROM ou RAM en $A000 pour des programmes à démarrage automatique comme par exemple le moniteur 6502.
- Soit 6K RAM en $2000 et 2K ROM (ou RAM) en $A000 pour la ROM AUTOBASIC. Cette configuration retient tout particulièrement l'attention pour ses possibilités de contrôle de processus.
- Soit 6K ROM en $A000 et 2K RAM en $AC00. Egalement pour le contrôle de processus mais on bénéficie de 2K de mémoire RAM que le BASIC ne risque pas de modifier par erreur: Les 6k ROM peuvent contenir un programme BASIC retransféré en RAM pour exécution lors du RESET.

La carte 8K est également munie d'un poussoir de RESET, ce qui permet la mise au point aisée de programmes AUTOSTART. La mémoire RAM en $A000 peut éventuellement être commutée en $2000 pour sauvegarder un programme BASIC normal et un programme en langage machine en $A000 d'un seul tenant sur disque ou cassette.

Les options de la carte se choisissent à l'aide de pontages définitifs ou si on le désire, à l'aide d'inverseurs. C'est cette dernière méthode que nous décrirons. Il y a sur la carte 6 inverseurs à 3 positions : ON-REPOS-OFF. La position repos laisse les 3 broches de l'inverseur indépendantes. La position ON (levier basculé vers l'arrière de la carte) connecte le point milieu de l'inverseur avec le point le plus proche du VIC (donc à l'opposé du levier).

Exemple : l'inverseur SW1 doit être en position OFF pour mettre le "watchdog" en service : on connecte à ce moment la broche centrale et la broche arrière de cet inverseur.

Description du schéma

1. Le circuit de RESET (WATCHDOG).

Le circuit IC5-74LS123 est un double monostable. Le premier monostable occupe les broches 6, 7, 10, 12 et son délai de 35 secondes environ est fixé par R1 et C2+C3. Le temps est proportionnel à la valeur du produit RC. Le redémarrage est déclenché par le signal I/O 3, c'est-à-dire par n'importe quelle lecture ou écriture en mémoire dans le bloc I/O 3. La sortie du premier monostable commande le déclenchement du second circuit monostable (broches 2, 13, 14, 15) qui génère l'impulsion de RESET. Cette impulsion est envoyée à IC7, circuit inverseur avec sortie à COLLECTEUR OUVERT, avant d'être envoyée au 6502 par le fil RESET du connecteur.

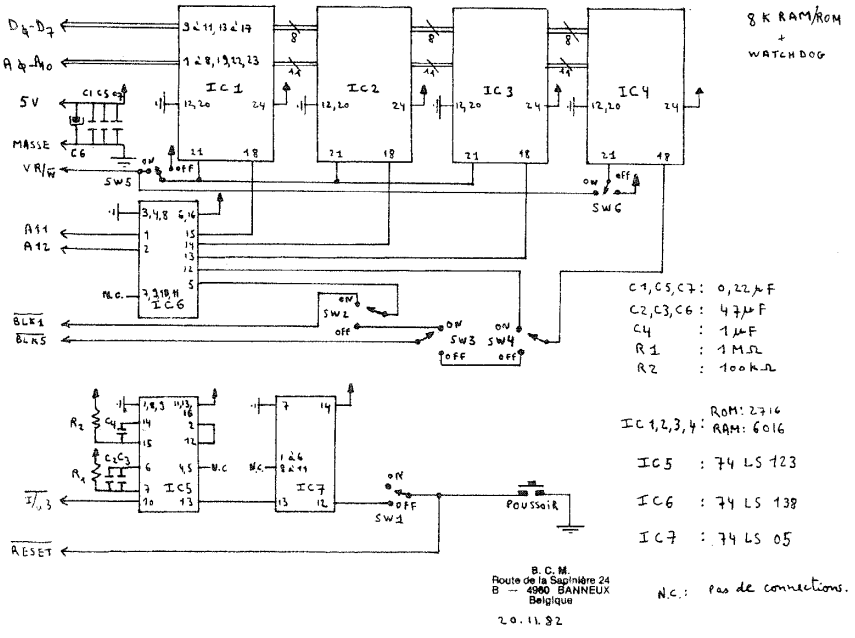
2. Le décodage des adresses

Il est réalisé par IC6 qui reçoit A11 et A12 et, suivant les 4 combinaisons possibles des états logiques de ces 2 adresses, porte à l'état logique 0 une et une seule des sorties 12, 13, 14 ou 15. Ces sorties sont reliées aux entrées de sélection des mémoires MEM1 et MEM4. On notera que la sélection de la mémoire MEM4 peut être aiguillée par SW4 vers la sélection BLK5 (mémoire en $A000) si les trois autres circuits sont en mémoire dans le BLOC 1 (BLK1 = $2000). Le circuit IC6 lui-même est sélectionné au choix

par le fil BLK1 ou le fil BLK5 à l'aide de l'inverseur SW2.

3. Les mémoires.

Elles sont raccordées très simplement aux bus d'adresses et de données à l'exception des broches 18 (entrée de sélection- voir ci-dessus) et des broches 21. La broche 21 doit être raccordée au fil VR/W pour un circuit RAM et au +5V pour un circuit ROM. Ceci est réalisé à l'aide des inverseurs SW5 pour les 3 mémoires MEM1 à MEM3 et SW6 pour la mémoire MEM4.



Fonctions des inverseurs.

Inverseur	Position ON	Position REPOS	Position OFF
Pontage	coté VIC	SANS	cote oppose
SW 1	watchdog hors service	watchdog hors service	watchdog en service
SW 2	SI MEM 1 en $2000	PAS DE MEMOIRE	SI MEM 1 en $A000
SW 3	MEM 1,2,3 et 4 en $A000	pas de mémoire en $A000	MEM 1,2,3 en $2000 MEM 4 en $A000
SW 4	MEM 1,2,3 et 4 dans le même bloc $2000 ou $A000	pas de mémoire MEM 4	si MEM 4 est la seule à être en $A000
SW 5	MEM 1, 2 et 3 = RAM	jamais	MEM 1, 2 et 3 = ROM
SW 6	MEM 4 = RAM	jamais	MEM 4 = ROM

Les mémoires doivent toutes être au standard "BYTEWIDE" 24 broches. Citons les ROM 2316 et EPROM 2716 ainsi que les RAM 2016 TOSHIBA et 6116 (NEC, FUJITSU, etc).

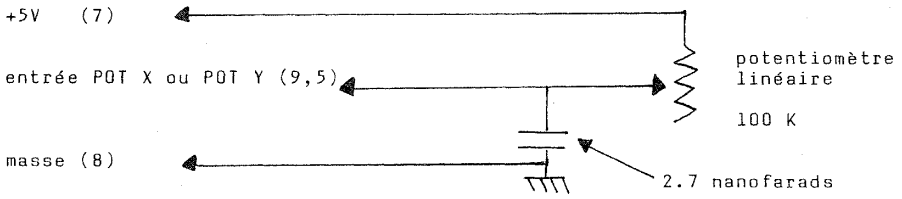
Il est évident que les amateurs de mémoire qui ne s'intéressent pas au contrôle de processus peuvent ne pas câbler sur la carte les composants correspondants :IC5, IC7, R1, R2, C2, C3, C4 et SW 1).

Notes de construction

- Les notes de construction des deux cartes précédentes restent d'actualité : vérifier l'orientation des composants, circuits intégrés et condensateurs.
- Utiliser pour les mémoires des support de bonne qualité ; ils peuvent alors être utilisés sans risque. Des supports bon marché ne résistent pas à plus de quelques insertions de composants.
- Insistons encore sur le soin tout particulier qu'il faut apporter au montage et aux tests de la carte.
- Les connections de passage entre les deux faces de la carte sont au nombre de 39.
- Le condensateur de découplage C6 doit avoir sa broche positive soudée par le dessus de la carte. On constate d'ailleurs que la pastille inférieure n'est pas raccordée.
- Ne pas oublier d'isoler la face inférieure.
- L'échelle des clichés est identique à celle des deux autres cartes.
- La découpe aura lieu en suivant l'intérieur du trait entourant la face supérieure.

Les entrées analogiques du VIC.

Les entrées POT X et POT Y du connecteur de jeu sont, on l'a vu, sensibles à la variation d'une résistance externe. Les valeurs lues en 36872 et 36873 indiquent l'état de ces entrées. Une poignée de jeu ou un capteur résistif quelconque peuvent être raccordés comme ceci :



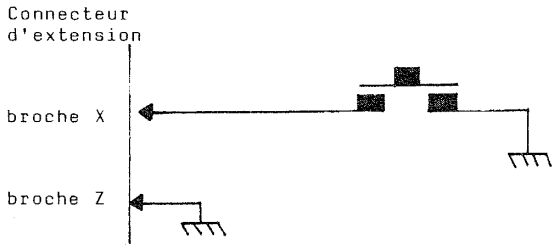
Les chiffres entre () sont les numéros des broches du connecteur de jeu. Le potentiomètre peut être remplacé par une résistance CTN 100K pour mesurer des températures ou par une photorésistance LDR pour réaliser des mesures d'éclairements (voir en annexe le programme thermomètre).

Dans le schéma ci-dessus, on voit un condensateur. Sa valeur peut être augmentée si la lecture du capteur potentiométrique ne monte pas jusque 255. La présence du condensateur augmente la sensibilité du VIC aux variations de résistance du capteur.

Modifications des cartouches existantes.

Les cartouches COMMODORE de 3K RAM sont aisément modifiables en cartouches de 4 K ROM. Il faut installer à l'un des emplacements à 24 broches un support pour EPROM 2532. La sélection de l'adresse est visible sur la carte : on connectera la broche 20 de la 2532 à la broche 13 = BLK5 du connecteur. De même, les cartouches de ROM de toutes provenances peuvent être modifiées:la sélection d'adresse peut toujours être déviée de la broche 13 vers une des broches 11 ou 12 (bloc 3 ou bloc 4) soit de façon permanente, soit à l'aide d'un interrupteur : il est ainsi possible de laisser simultanément plusieurs cartouches raccordées. On veillera à ne jamais commuter l'adresse d'une cartouche de mémoire à des moments où le 6502 y accède. Pour redémarrer une mémoire AUTOSTART que l'on vient de remettre en $A000, il suffit d'actionner le bouton poussoir de RESET, comme sur la carte 4K ROM décrite ci-dessus.

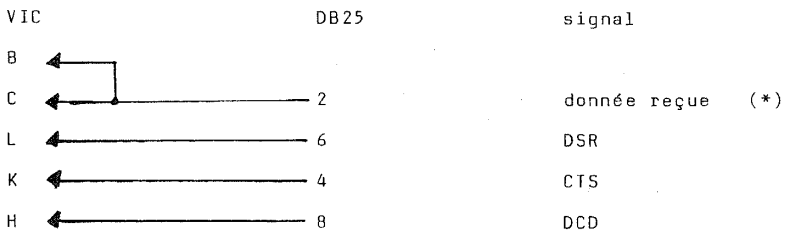
Poussoir de RESET



L'INTERFACE RS 232

Le VIC contient tout le logiciel nécessaire pour communiquer en RS 232. Seul problème : le connecteur n'est pas compatible et les niveaux logiques non plus. Pour le connecteur, il suffit de se procurer un modèle subminiature DB25 (femelle). Le point critique reste la conversion des niveaux logiques. Les tensions de sorties nécessaires étant de 3 à 12 volts, on utilisera pour la sortie positive, le + 5 volts du VIC et pour la sortie négative une pile plate de 4,5 volts ou toute autre source de tension continue de bonne qualité. Une alimentation simple convenant à cet usage est décrite ci-dessous . Elle utilise le 9 Volts alternatif présent sur le connecteur. La conversion de niveaux doit se faire dans les deux sens : vers l'extérieur et vers le VIC. Cette dernière est la plus simple.

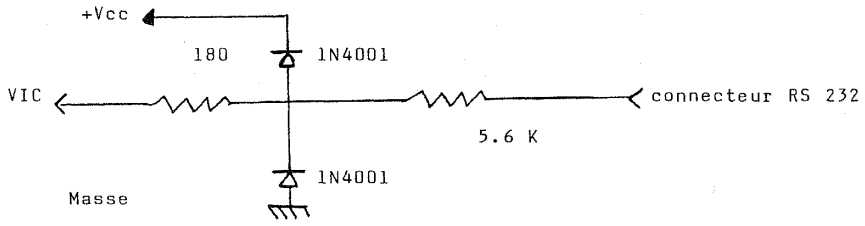
Connection d'entrée en RS-232.



(*) Le signal données est le seul nécessaire en mode 3 fils.

SHEMA D'ENTREE RS-232

Pour chacun des trois signaux ci-dessus, le shéma de conversion que voici est valable :



La résistance de 180 Ohms sert de protection pour le cas où le VIC programmerait le bit correspondant en sortie. Les diodes limitent les tensions à des valeurs entre 0 et 5 volts.

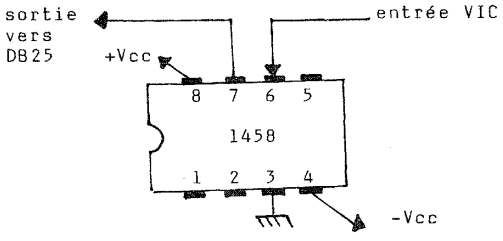
CONNECTION DE SORTIE EN RS 232

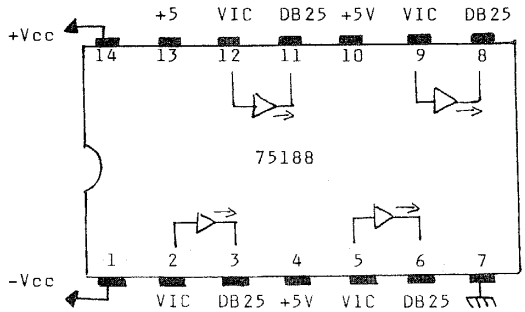
VIC	DB25	signal
M ⟶	3	donnée envoyée (*)
D ⟶	5	RTS
E ⟶	20	DTR

(*) Le signal donnée est le seul utilisé en mode 3 fils.

SHEMA DE SORTIE RS-232

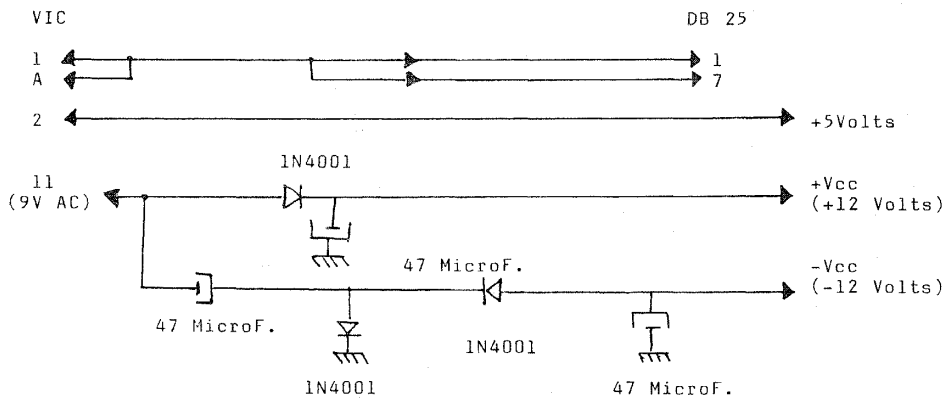
Pour les 3 signaux ci-dessus, une des deux conversions que voici convient:





Dans les shémas de sortie ci-dessus, on repère les fils d'alimentation : masse, +5V, +Vcc et -Vcc ; ces deux dernières tensions étant de l'ordre de 12 volts pour obtenir les niveaux corrects RS232.

SHEMA D'ALIMENTATION POUR INTERFACE RS232



LE PHOTOSTYLE

Le crayon lumineux est un dispositif simple : une cellule photosensible dirigée vers l'écran détecte le passage du point lumineux de balayage d'écran. Le VIC détecte cette impulsion et range les coordonnées du point dans les registres VIC-6 et VIC-7 ($9006, $9007 ou 36870, 36871). Le 6561 peut déterminer les coordonnées du point car il connaît à tout moment l'adresse du point concerné dans les mémoires écran et générateur de caractères.

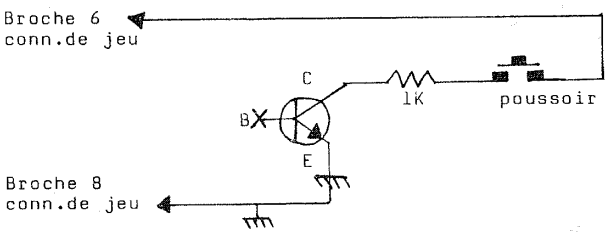
Le déplacement du point est bien plus lent en vertical qu'en horizontal : un balayage vertical dure 40 millisecondes et un balayage horizontal seulement 64 microsecondes. Il est donc évident que la coordonnée Y est bien plus précise que l'autre, car sa mesure est plus aisée.

D'un point de vue pratique, on constate que les registres VIC-6 et VIC-7 ne sont remis à jour que lors de la réception d'une impulsion. Donc, pour éviter les mesures non valables, on rajoute généralement un bouton poussoir qui ne raccorde la photocellule au VIC qu'à la demande.

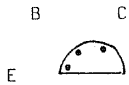
Une solution pratique consiste à mettre le poussoir juste derrière la cellule de façon à ce qu'en appuyant contre l'écran, le contact se ferme.

Comme cellule sensible, on utilisera un phototransistor à réponse rapide.

Voici un shéma qui donne satisfaction :



PHOTOTRANSISTOR
FPT 100 VU DE DESSOUS



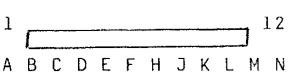
On notera que la base du phototransistor n'est connectée à rien. Le phototransistor FPT100 est disponible dans les magasins TANDY sous la référence 276-130, mais tout autre phototransistor rapide peut convenir. Avec ce montage, on observe une très bonne stabilité du registre VIC-7 (Y) tandis que la stabilité de VIC-6 (X) n'est pas idéale. (Voir programme photostyle en annexe).

Les valeurs lues varient de 45 à 141 pour X
Les valeurs lues varient de 20 à 145 pour Y.
Le cadre de 22*23 caractères, lui, donne des variations
de 55 à 135 pour X
de 35 à 122 pour Y.

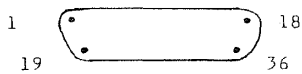
Imprimante parallèle sur port utilisateur.

De très nombreuses imprimantes existent qui se connectent sur un port parallèle grâce au standard "CENTRONICS". Nous allons voir comment ceci est possible dans le VIC. Tout d'abord, il convient de réaliser un câble de liaison.

Connecteur VIC



Connecteur CENTRONICS

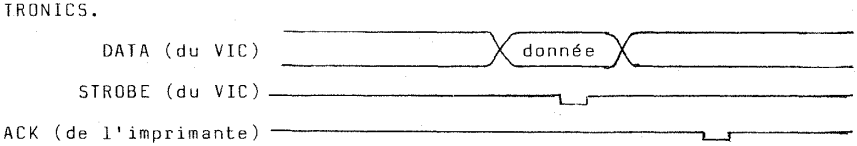


(connecteurs vus de face)

Un câble à 11 conducteurs sera nécessaire. On effectuera les connections suivantes :

VIC	CENTRONICS	NOM DU SIGNAL
A,1,12,N	19à30,36,14	référence 0 volt = GND (MASSE)
C	2	DATA 0
D	3	DATA 1
E	4	DATA 2
F	5	DATA 3 Données transmises
H	6	DATA 4 en parallèle
J	7	DATA 5
K	8	DATA 6
L	9	DATA 7
M	1	STROBE validation (=CB2)
B	10	ACK reconnaissance (=CB1)

Le diagramme temporel permet de visualiser aisément le dialogue VIC-CENTRONICS.



Le dialogue s'établit comme suit :

Le VIC dépose la donnée sur le port utilisateur, envoie une impulsion négative sur CB2 (STROBE = validation pour signaler qu'un caractère est présent. Ensuite, après "digestion" du caractère par l'imprimante, le fil CB1 (ACKNOWLEDGE = reconnaissance) reçoit une impulsion négative signalant que l'imprimante est prête pour le caractère suivant. Ce mode de transfert autorise un dialogue à des vitesses allant jusque 1200 caractères par seconde à l'aide du logiciel fourni dans ce livre. Le programme "CENTRONICS" en annexe est disponible sous deux formes : texte source en langage machine et chargeur BASIC pour utilisation directe. Actuellement, le programme occupe une centaine d'octets et est situé dans le tampon cassette. Il est cependant fort aisé de le transférer à une autre adresse. Le programme se subdivise en deux parties : initialisation et utilisation. SYS 900 modifie le vecteur de sortie pour

l'envoyer vers le programme proprement dit en $039F, programme le port utilisateur en sortie et envoie un caractère 00 à l'imprimante pour initialiser le dialogue. En $039F, le programme teste si le périphérique de sortie est bien le 6, numéro que nous avons choisi arbitrairement. Si oui, le caractère est déposé dans le port utilisateur et l'impulsion sur CB2 est générée par le programme. Ensuite, après réception de l'impulsion sur CB1, le branchement est effectué vers la routine normale.

```
SYS 900
OPEN 6,6: PRINT£6,"COUCOU"
CLOSE 6
```

Nous n'interceptons pas la routine **CLOSE** pour refermer le port parallèle explicitement. Le défaut de ceci est que après OPEN 6,6:CMD6:LIST, pour réaliser une impression de texte BASIC, lors du CLOSE 6, le message "DEVICE NOT PRESENT" est envoyé à l'écran. Etant sans conséquence pour le dialogue avec l'imprimante, ce défaut n'a pas été corrigé. Le programme peut évidemment être logé en ROM et la routine INIT (SYS 900) imbriquée dans la partie initialisation d'une RUM AUTOSTART pour éviter un SYS 900 après chaque STOP-RESTORE.

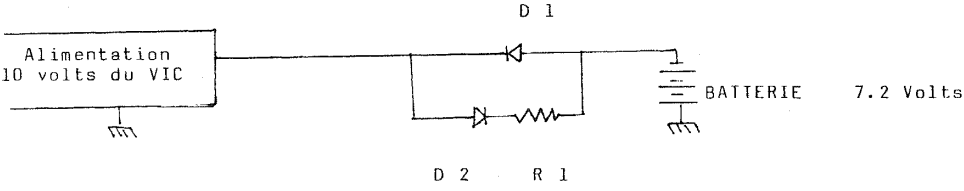
Il faut noter une particularité : le VIC n'envoyant pas le caractère 10(LINE FEED) après un RETURN, celui-ci est rajouté par le programme chaque fois que nécessaire.

L'ALIMENTATION ININTERRUPTIBLE

Il est possible de protéger un VIC et éventuellement son lecteur de disquettes des interruptions d'alimentation à l'aide de batteries d'appoint. Le dispositif ci-dessous protège le VIC contre des coupures pouvant durer de 5 à 10 minutes.

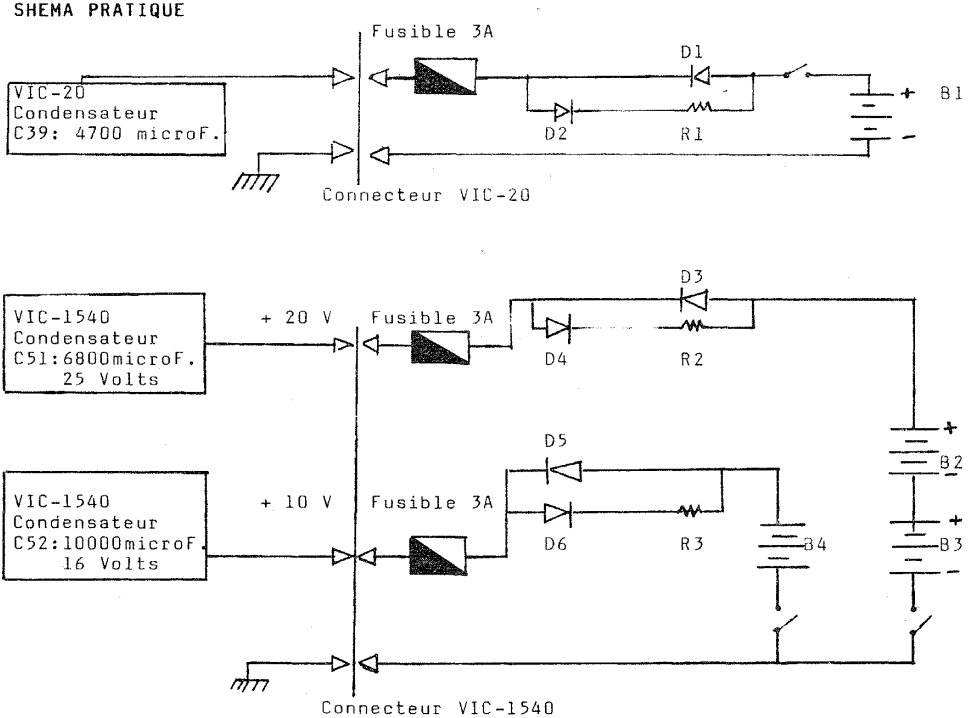
ATTENTION : COMMODORE ne couvre pas par sa garantie les VIC ayant subi cette modification !!. Il convient donc de réserver ceci aux électroniciens avertis. En effet, des connexions supplémentaires doivent être ajoutées à l'intérieur du VIC et du lecteur de disquettes VIC-1540. On installera à l'arrière du VIC-20 une prise à 2 conducteurs (type 'JACK' 3.5mm par exemple) reliée aux deux bornes du gros condensateur d'alimentation du VIC. **FAIRE TRES ATTENTION A BIEN RESPECTER LA POLARITE CORRECTE.** Pour le lecteur de disquettes, on utilisera une prise à trois conducteurs (JACK 6mm STEREO par exemple), car le lecteur a besoin de 2 tensions d'alimentation distinctes.

PRINCIPE



La diode D 1 laisse passer le courant de la batterie vers le VIC lorsque celui-ci n'est plus alimenté normalement. La diode D 2 et la résistance R 1 alimentent normalement la batterie pour la maintenir chargée. Pour les accumulateurs au CADMIUM-NICKEL, il est d'usage d'avoir un courant de charge permanent n'excédent pas le cinquantième de la capacité de la batterie en AMPERESxHEURE . Nous opterons pour des batteries CD-NI de 600 mAH, 7.2 Volts (6 éléments). Le modèle VARTA 6/600 convient bien . Un élément est nécessaire pour alimenter le VIC, trois sont nécessaires pour alimenter le lecteur VIC-1540.

SHEMA PRATIQUE



Les interrupteurs sont indispensables pour pouvoir arrêter le VIC et le 1540 Ceci est **INDISPENSABLE** quand on installe le boîtier d'alimentation.

D1 à D6 : diodes de redressement 3A (1N3408)
R1 : 220 Ohms 1/2 Watt
R2 : 470 Ohms 1/2 Watt
R3 : 330 Ohms 1/2 Watt

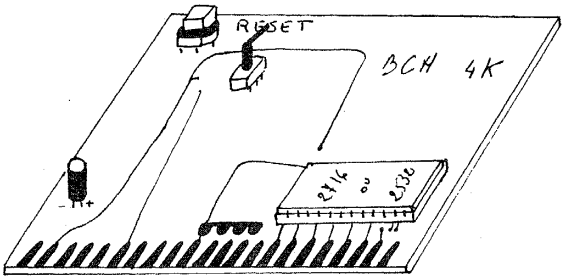
B1 à B4 : Batteries VARTA 6/600 CADMIUM/NICKEL

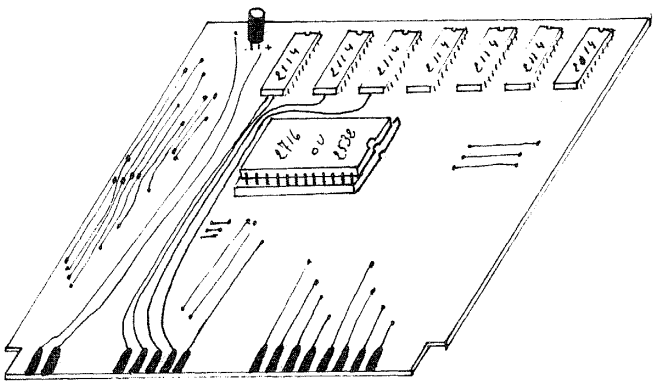
IMPLANTATION
DES COMPOSANTS

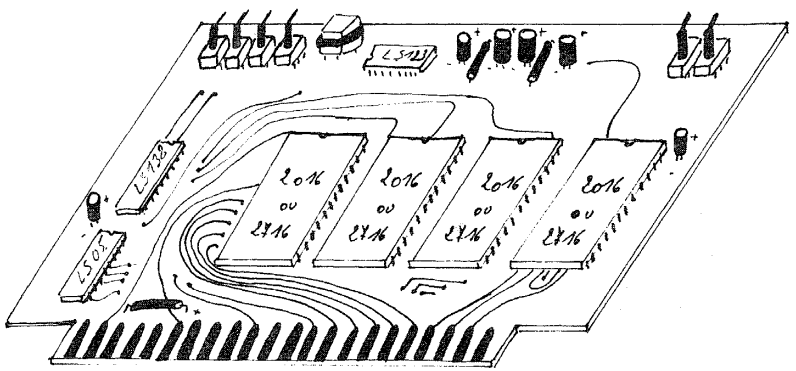
CARTE ROM

CARTE 3 K

CARTE 8 K







CHAPITRE 4.

LES EXTENSIONS LOGICIELLES.

Outre les programmes vendus ou publiés par de nombreuses firmes, ainsi que les programmes fournis avec ce livre, il est possible d'ajouter au VIC plusieurs mémoires AUTO-START ou non qui augmentent les possibilités du système ou tout au moins la facilité d'accès à diverses fonctions telles que son, haute résolution graphique, langage machine. Il existe sur le marché de nombreuses extensions de ce type. Outre le moniteur 6502 décrit ci-dessous qui donne accès au langage machine, il existe entre autres **VIC KIT I** et **VIC KIT II** de **STACK** qui comprennent des instructions graphiques (DRAW,BOX, etc) et des instructions utilitaires (RENUMBER, DELETE, TRACE, etc). On trouve également ces dernières dans la cartouche **PROGRAMMER'S AID** de COMMODORE. La mémoire d'extension la plus efficace semble être **SIMON'S BASIC** qui n'est malheureusement pas encore disponible : il fournit une centaine de commandes BASIC supplémentaires ! On trouve disponible actuellement le SUPEREXPANDER de COMMODORE qui apporte une dimension nouvelle au VIC avec des commandes sonores, graphiques et périphériques (manche de commande) d'un intérêt non négligeable. Aussi allons-nous étudier ce SUPEREXPANDER de plus près, avant de découvrir le moniteur 6502 et la ROM AUTOBASIC.

SUPEREXPANDER

Pour ceux qui ne possèdent pas le SUPEREXPANDER, citons tout d'abord ses principales fonctions :

- **Extension** mémoire de 3K RAM + 4K ROM

- **Programmation** des touches F1 à F8 avec des chaînes de caractères quelconques (longueur totale des chaînes = 128 caractères). Exemple : F8 devient "LIST<RETURN>". La commande est KEY.

- **Commandes graphiques** : de nouveaux mots-clé sont introduits : GRAPHIC, COLOR, REGION, POINT, DRAW, CIRCLE, PRINT, CHAR, SCNCLR.

La commande **GRAPHIC** donne accès à 4 modes graphiques seulement : GRAPHIC 0 est le mode normal, GRAPHIC 1 est le mode multicolore, GRAPHIC 2 le mode haute résolution statique en deux couleurs, GRAPHIC 3 est une combinaison des 2 précédents. Il n'y a pas de mode moyenne résolution, ni de mode dynamique.

Les coordonnées X, Y sont toujours comprises entre 0 et 1023, mais ne correspondent pas, bien entendu à 1023 points en horizontal ni en vertical. **COLOR** choisit les 2 ou 4 couleurs de base suivant le mode choisi, **POINT** trace un point dans une couleur donnée, **REGION** change la couleur d'écriture pour un caractère donné, **DRAW...TO** trace des lignes, **CIRCLE** trace des cercles, **PAINT** colore une zone précise de l'écran, **CHAR** trace une lettre dans un écran graphique non-multicolore. Il s'agit en fait d'une recopie de 8 octets du générateur de caractères en ROM dans le générateur en RAM utilisé à ce moment. **SCNCLR** efface un écran graphique.

- **Commandes sonores** : Les commandes sonores sont des suites de caractères inclues dans un ordre PRINT, elles sont exécutées. Le SUPEREXPANDER à cet effet intercepte la routine de sortie **PRINT** du BASIC. **SOUND** autorise certaines des 4 voix sonores (**VIC-A** à **VIC-D**) et programme le volume. **CTRL-flêche à gauche** dans une chaîne de caractères indique le début d'une commande sonore. La fin d'une commande est indiquée par la fin de la chaîne à imprimer. Les notes sont indiquées en notation américaine **:A, B, C, D, E, F, G, .** Les caractères **P** et **Q** autorisent et suppriment l'affichage des notes, **V** modifie le volume, **S** choisit une voix parmi les 4 possibles. **O** et **T** sélectionnent l'octave et le tempo. Le tempo varie de 0 à 9 soit de 900 à 14 coups/minute. R représente un silence, **£** un dièse et **$** un bémol.

-**Commandes de lectures de valeurs** : les commandes **RGR, RCOLR, RDOT, RSND, RPOT, RPEN, RJOY** permettent de lire des valeurs précises. RGR donne le numéro du mode graphique en cours, RCOLR la valeur d'une couleur de l'écran, RDOT la couleur à un point, RSND la fréquence du son, RPOT les valeurs des entrées analogiques **POT X** et **POT Y**. RPEN les valeurs des coordonnées du **photostyle**. **RJOY** indique quels interrupteurs sont fermés sur l'un ou l'autre des manches de commande.

La mémoire SUPEREXPANDER est une source très intéressante de sous-programmes langage machine. Les détails exacts de ces routines ne sont pas tous connus. Pour une meilleure compréhension, il est conseillé de transférer l'une ou l'autre routine en RAM, puis d'utiliser le moniteur pour analyser le fonctionnement du sous-programme en question. Voici une table des adresses connues utilisées par le SUPEREXPANDER.

VARIABLES EN RAM

	$0024	nombre de coordonnées ou mode ($FF=multicolore, $00=haute résolution)
	$0024-$0025	nouveau pointeur de début de BASIC
	$0026	valeur temporaire
	$0062	valeur XM du nombre de points max. horizontal(0 à 159)
	$0063	valeur XO du début de coordonnées X (0 à 159)
	$0064	différence de deux coordonnées X
	$0065	valeur YM du nombre maxi de points verticaux (0 à 159)
	$0066	valeur YM du début des coordonnées Y (0 à 159)
	$0067	différence de deux coordonnées Y-1
*	$0069	multiplicande (pour calcul d'échelles 1023 - 159)
	$006A	produit (idem)

	$006B-$006C	multiplicateur (idem)
OU	$0069	coordonnées X (pour DRAW)
	$006A	coordonnées Y (pour DRAW)
	$006B-$006C	nombre de points en Y entre deux variations de X
	$006D-$006E	nombre de points restant à tracer
OU	$0069-$006A	coordonnées du point (pour PAINT)
OU	$0069	rangée (0-19) (pour CHAR)
	$006A	colonne (0-19)
	$006B	longueur de la chaîne de caractères
	$006C-$006D	adresse de la chaîne
	$009B	index dans la table des touches programmables
	$009B-$009C	pointeur générateur de caractères
	$009D	index fin de table des touches programmables
	$009E	numéro de la touche programmable
	$009E	longueur de la chaîne programmée
	$009E-$009F	pointeur d'octet dans RAM couleur
	$00AC-$00AD	pointeur d'octet RAM écran
	$00C3	drapeau indiquant que BASIC est déplacé
	$00C3-$00C4	pointeur vers table musicale
	$00FB-$00FC	pointeur sommet de mémoire
	$02A1	nbre d'octets occupés par SUPEREXPANDER sommet mémoire
	$02A2	nbre d'octets occupés par définition des touches prog.
	$02A3	index de l'octet dans une chaîne programmée.
	$02A4	nbre d'octets restant à écrire ds une chaîne progr.
	$02A5	caractère musical précédent
	$02A6	mode musique (en marche = $80)
	$02A7	notes copiées à l'écran (OUI=$50, NON=$00)
	$02A8	voix en cours
	$02A9	index table des notes
	$02AA	durée note en cours (en 1/60 de seconde)
	$02AB	valeur du volume*2 pour la note en cours
	$02AC	valeur de l'octave en cours
	$02AD	index table des notes VOIX 1 + $80
	$02AE	décompte de la durée note en cours VOIX 1 (en 1/60)
	$02AF-$02B0	idem VOIX 2
	$02B1 $00B2	idem VOIX 3
	$02B3-$00B4	idem VOIX 4
	$02C0-$02C2	JMP vers exécution des nouvelles commandes ($A84F)
	$02C3	marge écran gauche
	$02C4	marge écran supérieure
	$02C5	nombre de colonnes écran (X max)
	$02C6	nombre de rangées écran (Y max)
	$02C7	rangée où se trouve le curseur
	$02C8	mode graphique en cours
	$02CA	couleur en cours
	$02CB	couleur écran en cours
	$02CC	couleur bord en cours
	$02CD	couleur d'écriture en cours
	$02CE	couleur auxiliaire en cours
	$02CF	index dans la table des paramètres d'une commande
	$02D0	partie haute adresse générateur de caractères en cours
	$02D1	toujours $80 (adresse générateur en ROM partie haute)
	$02D2-$02D3	ancienne valeur sommet de BASIC
	$02D4	ancienne adresse écran (haut)

	$02D5	ancienne coordonnée X (0-159)
	$02D6	ancienne coordonnée Y (0-159)
	$02D7	drapeau pour DRAW:$00=coord. dans l'écran, sinon nombre de coordonnées hors écran
	$02D8	nombre de coordonnées hors écran
	$02D9	vecteurs RAM
	$0300	=$A3FD erreurs
	$0302	=$C483 redémarrage
	$0304	=$A407 analyse d'une ligne BASIC
	$0306	=$A4BA étend les mots-clés BASIC
	$0308	=$A504 nouvelle commande BASIC
	$030A	=$A52A évalue expression
	$0314	=$A372 IRQ
	$0316	=$A2C2 BRQ
	$0324	=$A395 entrée d'un caractère
	$0326	=$A3A6 sortie d'un caractère
	$032E	=$A2C2 BRK
*	$033C-$03F	zone de sauvegarde lors de PAINT
OU	$033C	index X ou Y (pour CIRCLE)
	$0347-$0348	ancienne valeur de X pendant le tracé de CIRCLE
	$0349-$034A	ancienne valeur de Y
	$034B-$034C	nouvelle valeur de X
	$034D-$034E	nouvelle valeur de Y
	$034F-$0353	valeur en flottant coordonnées X du centre du cercle
	$0354-$0359	valeur en flottant coordonnées Y du centre du cercle
	$0400-$0FFF	extension RAM 3K inclue ds la cartouche SUPEREXPANDER SUPEREXPANDER
	$A000-$A001	=$A044 vecteur de RESET
	$A002-$A003	=$A077 vecteur de redémarrage par STOP-RESTORE
	$A009-$A010	table code ASCII des touches programmables
	$A011-$A043	table chaînes pour touches programmables
	$A044	routine de démarrage à froid
	$A077	routine de redémarrage à chaud
	$A08B	exécute KEY
	$A0BF	affiche toutes les définitions de touches
	$A132	chaîne 'KEY' (à l'envers)
	$A136	chaîne '+CHR$(13)' (à l'envers)
	$A140	chaîne 'CHR$(34)' (à l'envers)
	$A14A	supprime la programmation d'une touche (numéro dans X)
	$A17B	insère une chaîne dans la table des touches
	$A1B1	rend l'adresse de la chaîne programmée sur la touche (numéro dans X; en sortie index dans Y)
	$A1BF-$A213	table nouveaux mots-clés BASIC ASCII(TOKEN $CC à $DD)
	$A214-$A237	table des adresses des routines correspondantes
	$A238	initialise les vecteurs en page 3
	$A2A2-$A2C1	table des nouveaux vecteurs
	$A2C2	redémarrage par BRK
	$A2C8	envoie un caractère à l'écran (code dans A)
	$A318	arrête le mode musical
	$A337	interprète le décodage clavier
	$A366-$A369	code des touches programmables avant décodage
	$A36A-$A371	code touches programmables après décodage
	$A372	routine IRQ
	$A395	entrée d'un caractère (numéro du périphérique en $99)

$A3A6	sortie d'un caractère (numéro du périphérique en $9A)
$A3B4	entrée d'un caractère du tampon clavier
$A3B4	gestion de la touche <RUN>
$A3E8	gestion de la touche <RETURN>
$A3FD	écrit un message en mode graphic 0 (code erreur en A)
$A407	analyse et compacte une ligne BASIC
$A4BA	écrit un mot-clé BASIC en ASCII
$A504	exécute nvelle comm.BASIC, gère TOKEN entre $CC et $D6
$A52A	exécute une expression et gère TOKEN entre $D7 et $DD
$A597	modifie les vecteurs du BASIC au démarrage
$A5A5	sauve la table des sons en cours(adresse table ds X,Y)
$A5D1	routine d'interruption pour musique
$A602-$A625	table de conversion notes-fréquences
$A626	interprète les commandes musicales
$A629	exécute 'O' (défaut = 3)
$A644	exécute 'T' (défaut = 0)
$A65E	exécute 'S' (défaut = 4)
$A675	exécute 'V' (défaut = 7)
$A687	exécute 'R'
$A694	exécute 'P'
$A69C	exécute 'Q'
$A6A8	exécute une nouvelle note (numéro dans Y).
$A6AB	établit une nouvelle table à la fin d'une note.
$A6BA	joue les notes 'A' à 'G'
$A6CE	exécute '£'
$A6DB	exécute '$'
$A6E6-$A6EC	table de conversion code ASCII de note en no de note
$A6ED-$A6EF	table de conversion octave-fréquence
$A6F0-$A6F9	table de conversion tempo-durée
$A6FA	évalue dans le texte BASIC un paramètre 0-255 et deux paramètres 0-65535
$A6FD	évalue deux paramètres 0-65535
$A700	évalue un paramètre 0-65535
$A714	sauve l valeur ds table des paramètres (valeur dans A, index table dans Y)
$A71C	évalue deux paramètres 0-255
$A72C	acquiert les paramètres GRAPHIC
$A740	CIRCLE
$A763	DRAW
$A7A5	POINT
$A7BD	COLOR
$A7CF	REGION
$A7DE	SCNCLR
$A7DD	SOUND
$A7EA	CHAR
$A80A	PAINT
$A811	RPOT
$A818	RPEN
$A81C	RSND
$A820	RCOL
$A824	RGR
$A828	RDOT

$A848	RJOY
$A84F	JMP indirect vers les nouvelles commandes. (X (haut) et Y(bas) contiennent l'adresse de la table des paramètres, l'index dans la table de la commande à exécuter est dans A)
$A867-$A878	table des adresses nouvelles commandes (partie haute)
$A879-$A88A	table des adresses nouvelles commandes (partie basse)
$A88B	exécute GRAPHIC
$A8D4	transfère BASIC au-dessus de $2000 et exécute CLR
$A943	met l'écran en $1E00, le générateur de caract.en $1000
$A967	transfère BASIC à son adresse précéd. et exécute CLR
$A9B8	met à blanc l'écran graphique
$AA23	exécute RGR
$AA29	COLOR
$AA6B	REGION
$AA85	RCOL
$AA8C	RDOT
$AAE7	POINT
$AAF2	SCNCLR
$AB13	DRAW avec 1 paramètre
$AB23	DRAW avec trois paramètres
$AB35	SOUND
$AB55	RSND
$AB6A	RPOT
$AB77	RPEN
$AB7E	trace 1 point à partir des paramètres de la table
$ABE5	initialise pointeurs RAM couleur et générateur
$AC0B	dessine 1 ligne avec nouvelles coordonnées de départ
$AC11	dessine ligne depuis les coordonnées en cours (algorithme de BRESENHAM)
$AC93	exécute CIRCLE (algorithme d'analyse digitale différentielle)
$AD13	conversion d'angle en radians
$AD19	divise l'accu FLP 1 par 16
$AD23	calcule des nouvelles coordonnées X,Y pour le cercle
$AD6C	exécute PAINT
$AE0F	teste si 1 point est à colorier ou non (pour PAINT) (coordonnées X, Y dans les registres A et Y)
$AE57	exécute CHAR
$AEDA	exécute RJOY
$AF14	écrit les valeurs dans les registres du 6561
$AF34	sauvegarde le nombre de paramètres
$AF39	sauvegarde la valeur de couleur
$AF48	met à l'échelle les coordonnées X, Y
$AF76	met à l'échelle 1 coord. X ou Y(0-159). En entrée, A contient le nbre de colonnes ou rangées. En sortie le registre X contient ce nbre * coordonnée/128
$AFB1-$AFBA	table coordonnées Y des débuts de ligne dans la RAM couleur (partie basse).
$AFBB-$AFCE	table coordonnées X des débuts de ligne ds le générateur de caractères (partie basse)
$AFCF-$AFE2	table coordonnées X des débuts de ligne ds le générateur de caractères (partie haute)
$AFE3-$AFE5	table des bits pour RAM couleur.

$AFE7-$AFEE	table des masques de BIT pour mode haute résolution
$AFEF-$AFE6	table masques de paires de BITS pr mode multicolore
$AFF7-$AFFA	table des octets à écrire en mode multicolore
$AFFB-$AFFE	table des valeurs de conversions adresses 6502-adresse 6561.

LE MONITEUR 6502.

Le moniteur 6502 est un programme utilitaire qui permet d'écrire et de mettre au point des programmes en langage Assembleur. Ce programme, dont on trouvera le listing en annexe, a été écrit par le club COMMODORE U.S.A. qui donne à tous l'autorisation de le publier et de le reproduire. Originellement prévu pour les PET/CBM, il a été adapté au VIC et modifié en programme AUTO-START. Vous avez plusieurs possibilités . Par exemple, trouver un programmeur d'**EPROM** et y introduire le programme. Tranféré dans une mémoire 2532, il peut être utilisé avec la carte ROM ou dans deux 2716 avec la carte 8K. Il est aussi possible de l'introduire dans la carte mémoire 8K RAM, de le sauver sur cassette et de le rappeler à volonté. On peut aussi emprunter à quelqu'un le moniteur en ROM, le sauver sur cassette et le rappeler ensuite à volonté en RAM avec la carte 8K (LOAD"MONITEUR",1,1). Pour sauver un programme en langage machine, on utilise la commande S du moniteur. Mais la première fois, pour sauver le moniteur lui-même avant de l'essayer, il faut utiliser une astuce BASIC (voir le sous-programme **"MEMORYSAVE"** inclus dans le programme "CHARGEUR HEXA en annexe). La solution EPROM est cependant très conseillée car un programme en langage machine nécessite souvent lors de sa mise au point de nombreux essais et recharger 4K de programmes peut être fastidieux. BCM peut également vous fournir une copie du moniteur (précisez 2716 ou 2532). L'introduction du programme en mémoire nécessite l'utilisation du programme CHARGEUR HEXA en annexe. On le sauvera sur cassette après l'avoir vérifié avec VERIMON avant de l'exécuter !

A NOTER : **VERIMON** peut résider en mémoire en même temps que le CHARGEUR HEXA ce qui permet le contrôle sans devoir recharger de programme de la cassette.

Le texte complet du programme peut être obtenu en assembleur grâce à la commande D du moniteur qui peut ainsi se désassembler lui-même. On ne peut que conseiller l'utilisation du désassembleur pour comprendre le fonctionnement de programmes en langage machine écrits par d'autres. A l'allumage du VIC ou en appuyant sur le bouton RESET de la carte 8K, on démarre le moniteur 6502 AUTO-START. Les commandes X et E permettent de repasser au BASIC. Le retour au moniteur peut se faire à tout moment par :

Retour au moniteur : SYS40978		($A012)
Retour au BASIC :	X	(provisoirement)
	E	(complètement)

La différence entre les commandes X et E est que X laisse inchangés les vecteurs BRK, NMI et IRQ si bien que les déplacements de curseur ainsi que l'utilisation du lecteur de cassettes peuvent provoquer des anomalies. Cependant, l'exécution d'une instruction BRK ($00) en langage machine provoquera le retour automatique au moniteur. La commande E supprime de la mémoire RAM toutes les modifications introduites par le moniteur.

Attention, les commandes E et X exigent que le pointeur de pile (SP) possède la même valeur qu'au moment où on a appelé le moniteur. Vérifiez et corrigez cette valeur avec la commande R.

L'environnement BASIC est légèrement modifié par le moniteur : le tampon cassette est redéfini en $0375, les vecteurs RAM sont dérivés, etc. En général, ceci ne pose pas de problème mais il existe une solution radicale : modifier un octet pour que le programme ne soit plus AUTO-START. On ne peut plus alors appeler le moniteur que par SYS40978. Par exemple, on peut modifier l'octet $A007 et remplacer $C2 par $00. En fait, n'importe quelle valeur convient tant que l'on modifie un des octets $A004 à $A008 inclus (voir **AUTOSTART**).

Actuellement, pour une raison non élucidée à ce jour, les commandes L, S et V ne fonctionnent pas correctement avec les lecteurs de disquettes VIC-1540.

Les commandes du moniteur sont au nombre de 30. Les voici ;

A B C D E F G I J L M N O P Q R S T V W X

" £ % ( ) + - &

NOTE : chaque élément d'une ligne de commande est séparé du précédent par un espace. Il n'y a pas d'espace entre le point et le caractère de commande.

A assemble une ligne

```
.A 2000 A9 14 LDA£$14
.A 2002 ...
```

On introduit la ligne :.A 2000 LDA £$14 <RETURN> .Le VIC complète la ligne avec le code assemblé puis affiche le numéro de ligne suivante. Pour arrêter, taper <RETURN> ou <STOP>.

B break ; arrête l'exécution d'un programme.

```
.B 1600
.B 1600 0047
```

Lorsque l'on utilise la commande **Q**, le programme s'exécute jusque l'adresse spécifiée (ici $1600), où il revient au moniteur. Le second paramètre (optionnel) indique le nombre de passages par l'adresse en question avant l'arrêt. La valeur par défaut est 1.

C compare deux zones mémoire.

```
.C 0100 01FF 0700
```

Cette instruction compare tous les octets de $0100 à $01FF avec les octets à partir de $0700. Tous les octets différents voient leur adresse affichée.

D désassemble.

```
.D A009 AFFE
```

L'instruction lit les octets en mémoire et les affiche suivant le format : .,A009 20 8DFD JSR $FD8D

L'éditeur d'écran du VIC permet de corriger les octets affichés en hexa mais pas les mnémoniques. Pour modifier ceux-ci, il faut retaper un A par dessus la , en début de ligne. A noter que, comme pour de nombreuses autres commandes, les déplacements de curseur haut et bas réalisent un décalage de l'écran vers le bas ou vers le haut. On peut commander le désassemblage une ligne à la fois par la commande .D009 et passer aux lignes suivantes avec 'curseur en bas", ou passer aux lignes précédentes avec "curseur en haut" quand le curseur est en haut de l'image.

E Sortie du moniteur avec suppression des vecteurs modifiés (exit)

```
.E
```

F remplit la mémoire (FILL)

```
.F 1E00 1FFF 2A
```

Cette commande remplit la zone de $1E00 à $1FFF avec l'octet $2A : des astérisques plein l'écran !

G exécute un programme (GO)

```
.G
```

Cette commande exécute le programmme à partir de la valeur contenue dans le registre PC.
.G A009 exécute le programme à partir de l'adresse A009.

I initialise mémoire et écran.

```
.I 1000 1E00 1E
```

Cette commande définit le début de zone BASIC en $1000, le sommet de zone BASIC en $1E00 et l'écran en page $1E, c'est-à-dire en $1E00. Attention, l'écran doit être à des adresses multiples de 512 ($0200) : voir le chapitre sur le 6561. Après modification par I et retour au BASIC, exécuter NEW et STOP-RESTORE pour rétablir les pointeurs BASIC corrects.

J exécute un sous-programme (JSR)

```
.J2000
```

Le sous-programme en $2000 est exécuté. Il doit se terminer par un RTS. Les registres affichés par la commande R ne sont pas modifiés.

L charge une zone mémoire (LOAD)

```
.L2000 "PROG"01
```

charge le programme "PROG" à l'adresse $2000.Si le nom de fichier est omis pour la cassette, le premier rencontré est chargé. Le troisième paramètre est optionnel. Par défaut, il vaut 01. 01 signifie cassette, 08 signifie disquette. L'adresse de chargement normale du fichier peut être connue en inspectant le contenu du tampon cassette en $0375. L'adresse de fin de zone chargée dépend de la longueur du fichier sur le disque ou la cassette. Attention, si vous avez l'extension 3K, la commande L remet à

blanc ($20) les octets $0400 à $0438 : c'est un inconvénient dû au déplacement du tampon cassette.

M affiche le contenu de la mémoire.

.M A000 ou .M A000 A020
affiche le contenu de la mémoire : une ligne à la fois à partir de $A000, pour le premier exemple, la zone de $A000 à $A020 pour le second.

Le VIC affiche :

```
.:A000 09 A0 C7 FE 41
.:A005 30 C3 C2 CD 20
```

Les valeurs hexa peuvent être modifiées grâce à l'éditeur d'écran. On peut utiliser les curseurs haut et bas pour faire défiler la mémoire.

N recalcule les adresses d'un programme.

```
.N 1000 14FF 6000 1000 1FFF
.N 1500 1600 6000 1000 1FFF W
```

Cette commande complexe est très utile quand on déplace un programme en langage machine dans la mémoire. Après déplacement du programme par la commande T, on utilise N pour recalculer les nouvelles adresses : les arguments de toutes les instructions qui occupent 3 octets situées dans la zone $1000 à $14FF voient leur valeur augmentée de $6000 pour autant que cette valeur soit comprise entre $1000 et $1FFF. Le second exemple fait subir la même transformation à la zone $1500 à $1600 qui est alors, à cause du W, considérée comme une table d'adresses (paires d'octets-partie basse, partie haute). La commande N s'interrompt si elle rencontre un OPCODE inconnu. Attention à ceci : la commande N ne fonctionne qu'une seule fois après chaque allumage du VIC.

O calcule une valeur de saut relatif (offset)

```
.O 0500 0520 1E
```

à l'aide d'une instruction de branchement (BNE, BMI...) située en $0500, on désire effectuer un saut à l'adresse $0520. le VIC donne la valeur $1E qui est le second octet nécessaire à l'instruction de branchement.

P impression (printer)

A la frappe de .P <RETURN> , toutes les sorties de caractères vers l'écran sont dérivées vers une imprimante RS232. Le retour à la normale se fait également par la même commande. Attention, l'RS232 a besoin de 512 octets qu'il se réserve au sommet de la mémoire. Lors du retour, le tampon RS232 continue à se vider vers l'imprimante.

.P0000

referme le canal RS232 et réaffiche les sorties à l'écran.

.P UUVV

aiguille la sortie vers l'RS232 en spécifiant l'octet $UU comme octet de commande et l'octet $VV comme octet de contrôle (voir RS232).

Q exécute un programme rapidement (QUICK)

.Q 1000

exécute un programmme situé en $1000. La syntaxe est identique à la commande G, mais l'exécution s'arrête à la rencontre d'un point d'arrêt (voir commande B) ou si l'on frappe STOP et = simultanément.

R affiche le contenu des registres du 6502

.R

```
     PC   IRQ  SR   AC   XR   YR   SP
.;   UUUU UUUU UU   UU   UU   UU   UU
```

On peut modifier le contenu des registres avec l'éditeur d'écran.

S sauve le contenu d'une zone mémoire (SAVE)

.S 2000 3000 "PROG" 01

sauve la mémoire de $2000 à $2FFF (=$3000-1) sur la cassette (01 par défaut). Sous le nom "PROG" le BASIC peut recharger directement une telle zone par LOAD"PROG",1,1

T transfère le contenu d'un bloc de mémoire.

.T 1000 10FF 2000

transfère les octets de $1000 à $10ff inclus dans la zone débutant en $2000.

V vérifie un programme

.V 2000 "PROG" 01

vérifie une zone mémoire. Syntaxe identique à la commande L.

W exécute un programme pas à pas (WALK)

.W ou .W1000

affiche chaque ligne du programme en hexa et mnémoniques ainsi que le contenu des registres à chaque instruction. La commande exécute une instruction puis désassemble la suivante. S'arrête par STOP et recommence

par W <RETURN>. A tout moment, la touche J saute (JUMP) à la fin du sous-programme en cours et permet de ne tester qu'un programme principal et non toutes ses sous-routines.

X sort du moniteur .

maintient les vecteurs d'interruption aux valeurs imposées par le moniteur. Exécuter une instruction $00 (BRK) en langage machine fait retomber dans le moniteur. Sert à tester des programmes en langage machine à partir du BASIC. (Essayer SYS4096 sur un VIC de base).

" conversion ASCII

```
."A
```

affiche les valeurs du caractère ASCII "A" en décimal, hexa et binaire.

£ conversion décimale

```
.£200
```

affiche les valeurs du nombre décimal 200 en hexa, ASCII et binaire (65535 maxi)

$ conversion hexa

```
.$ 0400
```

affiche les valeurs du nombre hexa 0400 en décimal, ASCII et binaire.

% conversion binaire (maxi $FFFF)

affiche les valeurs du nombre binaire en hexa, décimal, ASCII (maxi2^16)

(autorise le son

cette commande provoque la génération d'un son à la fin de chaque commande : c'est pratique pour de longues commandes comme S,H<W. On arrête le son par <RETURN>. Il n'y a pas de son en cas d'erreur de syntaxe.

) supprime la commande son.

```
.)
```

+ addition hexa

```
.+ UUUU VVVV
```

affiche la somme des nombres hexa $UUUU et $VVVV.

- soustraction hexa.

.- UUUU VVVV

affiche le résultat hexa de la soustraction $UUUU-$VVVV

& calcule une somme de contrôle (CHECKSUM)

.& A000 AFFF

affiche la somme de contrôle sur 4 caractères hexa de la zone $A000 à $AFFF. (La valeur correcte est $57CE).

La ROM AUTOBASIC.

La ROM AUTOBASIC est une mémoire ROM qui contient un programme dont le but est de charger en mémoire un programme BASIC dès l'allumage du VIC. L'intérêt de ce programme est évident : il facilite les manipulations pour les profanes (ou les très jeunes utilisateurs).

Autre intérêt de ce programme : combiné à la carte d'extension 8K + "WATCHDOG" décrite au chapitre suivant, il transforme le VIC en un contrôleur de processus, dispositif de démonstration, etc... qui ne craint pas les coupures d'alimentation.

Ce programme AUTOBASIC exécute tout d'abord la séquence complète d'initialisation du VIC, puis affiche le message "BCM AUTOBASIC". Ensuite, il dépose dans le tampon de clavier les messages LOAD "AUTO", 8<RETURN>, RUN <RETURN> ou le message LOAD <RETURN> RUN <RETURN>. Ce dernier message est utilisé dans le cas où le programme a détecté la présence d'un lecteur de cassettes dont la touche PLAY est déjà enfoncée, Ensuite, un délai de 1 à 2 secondes est nécessaire pour laisser le temps à tous les périphériques de terminer leur processus d'initialisation et de libérer le bus IEEE série. Le contrôle est enfin rendu à l'interpréteur BASIC qui prendra les caratères du tampon clavier exactement comme si on les avait frappés au clavier.

Seule différence avec le VIC normal : le tampon clavier accepte maintenant 15 caractères au lieu de 10.

Le texte complet de la ROM AUTOBASIC est disponible en annexe. Il est aussi possible de se procurer la ROM AUTOBASIC chez BCM.
La fiabilité générale du processus peut encore être notablement améliorée en ajoutant au VIC le dispositif d'alimentation ininterruptible décrit au chapitre "EXTENSIONS MATERIELLES".

CHAPITRE 5

CARTE DES ADRESSES MEMOIRE DU VIC 20

ETIQUETTE-ADRESSE-LONGUEUR-CONTENU-COMMENTAIRES ET EXPLICATIONS

BLOC 0

PAGE 0 1: ADRESSES PROPRES AU BASIC

USER	0	$0000	3	JMP $D248	JMP VERS ROUTINE UTILISATEUR
	3	$0003	2	$D1AA	VECTEUR CONVERSION FLOTTANT-FIXE
	5	$0005	2	$D391	VECTEUR FIXE-FLOTTANT
	7	$0007	1	$22	CARACTERE PENDANT UNE RECHERCHE
	8	$0008	1	$22	
	9	$0009	1	$00	NUMERO DE COLONNE AVEC TABULATEUR
	10	$000A	1	$00	FLAG CASSETTE 0=LOAD,1=VERIFY
	11	$000B	1	$4C	POINTEUR TAMPON D'ENTREE BASIC
	12	$000C	1	$09	FLAG TABLEAU DIMENSIONNE PAR DEFAUT A 10
	13	$000D	1	$00	TYPE DE VARIABLE : 0=NUMERIQUE 255=CHAINE
	14	$000E	1	$00	TYPE DE VARIABLE : 0=FLOTTANT 128=ENTIER
	15	$000F	1	$00	FLAG "DATA"-"LIST"-"MODE GUILLEMETS"
	16	$0010	1	$00	FLAG VARIABLE INDICEE-FONCTION
	17	$0011	1	$40	FLAG TYPE D'ENTREE 0=INPUT-40=GET-98=READ
	18	$0012	1	$02	FLAG QUADRANT 2 OU 3 POUR CALCUL DE ATN FLAG COMPARAISON-EVALUATION POUR <=>,ETC
	19	$0013	1	$00	FLAG MESSAGE PRESENT POUR MOTEUR CASSETTE
	20	$0014	2		VALEUR ENTIERE
	22	$0016	1		POINTEUR PILE DE STOCKAGE PROVISOIRE
	23	$0017	2		DERNIER POINTEUR DE CHAINE PROVISOIRE
	25	$0019	9		PILE DES POINTEURS DE CHAINES PROVISOIRES
	34	$0022	4		POINTEURS DIVERS
	38	$0026	5		RESULTAT INTERMEDIAIRE POUR MULTIPLICATIONS
	43	$002B	2	$1001	POINTEUR DEBUT BASIC (BAS DE MEMOIRE)
	45	$002D	2	$1003	POINTEUR FIN DE BASIC
	47	$002F	2	$1003	POINTEUR FIN DE VARIABLES
	49	$0031	2	$1003	POINTEUR FIN DE TABLEAUX
	51	$0033	2	$1DFB	POINTEUR BAS DE ZONE CHAINES

	53	$0035	2	$1DFB	POINTEUR HAUT DE ZONE CHAINES
	55	$0037	2	$1E00	POINTEUR FIN DE MEMOIRE RAM
	57	$0039	2		NUMERO DE LIGNE BASIC EN COURS
	59	$003B	2		NUMERO DE LIGNE BASIC PRECEDENTE
	61	$003D	2		POINTEUR DE LIGNE POUR "CONT"
	63	$003F	2		NUMERO DE LIGNE DU "DATA" EN COURS
	65	$0041	2		POINTEUR VERS ZONE DE DATA EN COURS
	67	$0043	2	$0201	POINTEUR ZONE EN COURS D'INTERPRETATION
	69	$0045	2		NOM DE LA VARIABLE EN COURS
	71	$0047	2		ADRESSE DE LA VARIABLE EN COURS
	73	$0049	2		POINTEUR DE VARIABLE POUR "FOR-NEXT"
	75	$004B	2		SAUVEGARDE DE Y OU D'UN POINTEUR BASIC
	77	$004D	1	$00	SYMBOLE DE COMPARAISON EN COURS <,=,>
	78	$004E	6		VALEUR FLOTTANTE PROVISOIRE POUR TAN,..
	84	$0054	3	JMP $D80D	VECTEUR DE SAUT POUR FONCTION FN
	87	$0057	10		ZONE DE STOCKAGE POUR CALCULS
	97	$0061	1		**ACCUMULATEUR FLOTTANT 1** : EXPOSANT
	98	$0062	4		MANTISSE
	102	$0066	1		SIGNE
	103	$0067	1	$00	POINTEUR DE CONSTANTES POUR CALCUL SIN,COS
	104	$0068	1		ACCU. FLOTTANT 1 : OCTET SUPPLEMENTAIRE
	105	$0069	1		**ACCUMULATEUR FLOTTANT 2** : EXPOSANT
	105	$006A	4		MANTISSE
	109	$006E	1		SIGNE
	111	$006F	1		SIGNE DE LA DIFFERENCE DES 2 ACCU. FLOTT.
	112	$0070	1		ACCU. FLOTTANT 1 : OCTET POUR ARRONDI
	113	$0071	1	$0b	LONGUEUR DU TAMPON CASSETTE OU RS232
CHARGOT	115	$0073	24		ROUTINE D'ACQUISITION D'UN CARACTERE
	122	$007A	2		POINTEUR DE CHARGOT
	139	$008B	5	$804FC75258	VALEUR DE REFERENCE POUR "RND"

PAGE 0 **2:ADRESSES PROPRES AU KERNAL**

	144	$0090	1	$00	OCTET MESSAGE D'ERREUR **ST-STATUS**
	145	$0091	1	$FF	FLAG TOUCHE STOP,SHIFT GAUCHE,X,V,N,?
	146	$0092	1	$00	CONSTANTE DE TEMPS POUR CASSETTE
	147	$0093	1	$00	FLAG CASSETTE 1=LOAD,0=VERIFY
	148	$0094	1	$00	SORTIE RS232 : CARACTERE DIFFERE
	149	$0095	1	$00	ENTREE RS232 : CARACTERE DIFFERE
	150	$0096	1	$00	FLAG INDIQUANT FIN DE CASSETTE (EOT)
	151	$0097	1		SAUVEGARDE DE REGISTRE
	152	$0098	1	$00	NOMBRE DE FICHIERS OUVERTS
	153	$0099	1	$00	NUMERO DU PERIPHERIQUE D'ENTREE
	154	$009A	1	$03	NUMERO DU PERIPHERIQUE DE SORTIE
	155	$009B	1	$00	PARITE DE L'OCTET RECU/ENVOYE CASSETTE
	156	$009C	1	$00	FLAG OCTET RECU DE LA CASSETTE
	157	$009D	1	$80	FLAG MODE DIRECT=128,MODE RUN=0
	158	$009E	1	$00	IEEE SERIE RECHERCHE D'ERREUR
	159	$009F	1	$00	IEEE SERIE ERREUR CORRIGEE
	160	$00A0	1	$00	**HORLOGE** EN 1/60SEC. (**JIFFY**)
	163	$00A3	1	$00	COMPTEUR DE BITS SUR IEEE SERIE
	164	$00A4	1	$00	COMPTEUR DE CYCLES
	165	$00A5	1	$00	DECOMPTEUR DE BITS EN ECRITURE CASSETTE
	166	$00A6	1	$00	POINTEUR TAMPON DE CASSETTE
INBIT	167	$00A7	1	$00	RS232 VALEUR DU BIT RECU

BITCI	168	$00A8	1	$00	RS232 NOMBRE DE BITS RECUS
RINONE	169	$00A9	1	$00	RS232 FLAG TEST DU BIT START EN COURS
RIDATA	170	$00AA	1	$00	RS232 OCTET EN COURS DE RECEPTION
PIPRTY	171	$00AB	1	$00	RS232 BIT DE PARITE RECU
	172	$00AC	2	$0000	POINTEUR DANS LE TAMPON CASSETTE
	174	$00AE	2	$0000	ADRESSE FIN DE ZONE LUE SUR CASSETTE
	176	$00B0	2	$0000	CONSTANTE DE TEMPS CASSETTE
	178	$00B2	2	$033C	POINTEUR DE DEBUT DE TAMPON CASSETTE
BITTS	180	$00B4	1	$00	RS232 COMPTEUR DE BITS EN EMISSION
NXTBIT	181	$00B5	1	$00	RS232 PROCHAIN BIT A EMETTRE
RODATA	182	$00B6	1	$00	RS232 POINTEUR OCTET EN COURS D'EMISSION
	183	$00B7	1	$00	NOMBRE DE CARACTERES POUR NOM DE FICHIER
	184	$00B8	1	$00	NUMERO DU FICHIER EN COURS
	185	$00B9	1	$00	ADRESSE SECONDAIRE DU FICHIER EN COURS
DEVCUR	186	$00BA	1	$00	NUMERO DE PERIPH. DU FICHIER EN COURS
	187	$00BB	2	$0000	POINTEUR NOM DE FICHIER EN COURS
	189	$00BD	1	$00	OCTET EN COURS DE LECTURE/ECRITURE IEEE
	190	$00BE	1	$00	NOMBRE DE BLOCS RESTANTS A LIRE/ECRIRE
	191	$00BF	1	$00	STOCKAGE OCTET POUR TRANSMISSION IEEE SERIE
	192	$00C0	1	$00	FLAG CASSETTE MANUEL/PROGRAMME
	193	$00C1	2	$2000	ADRESSE DE DEBUT DES ENTREES/SORTIES
	195	$00C3	2	$FD6D	POINTEUR KERNAL
	197	$00C5	1	$40	VALEUR NON DECODEE DE LA TOUCHE ENFONCEE
KBDPTR	198	$00C6	1	$00	NOMBRE DE CARACTERES DANS TAMPON CLAVIER
	199	$00C7	1	$00	FLAG MODE **REVERSE**=18,NORMAL=00
	200	$00C8	1		POINTEUR DE FIN DE LIGNE EN ENTREE
	201	$00C9	1	$02	POSITION **CURSEUR** bas
	202	$00CA	1	$13	POSITION **CURSEUR** haut
	203	$00CB	1		VALEUR NON DECODEE TOUCHE PREC.RIEN=64
CURFLG	204	$00CC	1	$00	FLAG **CURSEUR** VISIBLE=0,INVISIBLE=1
CURDEL	205	$00CD	1		DECOMPTAGE DUREE CLIGNOTEMENT **CURSEUR**
CARCUR	206	$00CE	1		CODE DU CARACTERE SOUS LE CURSEUR
CURONF	207	$00CF	1		FLAG CURSEUR ALLUME/ETEINT
	208	$00D0	1	$00	ENTREE VENANT DE L'ECRAN OU DU CLAVIER
ADLN	209	$00D1	2	$1E84	POINTEUR DE LA LIGNE ECRAN EN COURS
CURXP	211	$00D3	1	$04	POSITION DU CURSEUR DANS LA LIGNE
	212	$00D4	1	$00	FLAG MODE GUILLEMETS=1,NORMAL=0
	213	$00D5	1	$15	LONGUEUR LIGNE ECRAN EN COURS(22,44,66,88)
	214	$00D6	1		NUMERO DE LIGNE ECRAN EN COURS
	215	$00D7	1		CODE ASCII DERNIERE TOUCHE ENFONCEE
	216	$00D8	1		NOMBRE DE BLANCS INSERES RESTANT A REMPLIR
	217	$00D9	25		POIDS FORT DES ADRESSES DE LIGNES ECRAN
	242	$00F2	1	$00	INDEX PROVISOIRE DANS LA TABLE CI-DESSUS
	243	$00F3	2	$E4,$97	COULEUR DE L'ECRAN
	245	$00F5	2	$EC5E	POINTEUR TABLE DE DECODAGE CLAVIER
	247	$00F7	2	$0000	RS232 ADRESSE TAMPON RECEPTION
	249	$00F9	2	$0000	RS232 ADRESSE TAMPON EMISSION
	251	$00FB	5		ZONE INUTILISEE

PAGE 1 **ZONE DE LA PILE**

256	$0100	255	BAS DE LA PILE LA PILE SE REMPLIT DE $01FF VERS $0100
256	$0100	10	ZONE POUR CONVERSION ASCII-FLOTTANT
256	$0100	64	STOCKAGE DES ERREURS EN LECTURE CASSETTE

				A PARTIR DE $0100
STKTOP	511	$01FF	1	PREMIER OCTET DE LA PILE

PAGE 2

	512	$0200	89	TAMPON D'ENTREE BASIC
	601	$0259	10	TABLE DES NUMEROS DE FICHIERS EN COURS
	611	$0263	10	TABLE DES NUMEROS DE PERIPH. CORRESPONDANTS
	621	$026D	10	TABLE DES ADRESSES SECONDAIRES CORRESP.
KBDBUF	631	$0277	10	TAMPON DE **CLAVIER**
	641	$0281	5	EXTENSION DU TAMPON DE CLAVIER
	646	$0286	1 $00	CODE DE COULEUR EN COURS A L'ECRAN
	647	$0286	1 $F1	CODE DE COULEUR SOUS LE CURSEUR
	648	$0288	1 $1E	NUMERO DE LA PAGE DE MEMOIRE ECRAN
KBDMAX	649	$0289	1 $0A	TAILLE MAXI DU TAMPON **CLAVIER**
	650	$028A	1 $00	FLAG REPETITION NORMAL=0,TOUT=128,RIEN=127
	651	$028B	1 $02	DELAI AVANT DEBUT **REPETITION**
	652	$028C	1 $10	DELAI ENTRE REPETITIONS
	653	$028D	1 $00	MODE CLAVIER:NORM=255,SHIFT=159,LOGO=244
	654	$028E	1 $00	MODE CLAVIER PRECEDENT
	655	$028F	2 $EBDC	VECTEUR INDIRECT BALAYAGE CLAVIER
	657	$0291	1 $00	FLAG TOUCHE **LOGO** ENFONCEE (C=)
	658	$0292	1 $00	FLAG MODE **SHIFT** AUTORISE=0,INTERDIT=1
M51CTR	659	$0293	1 $00	**RS232** PSEUDO REGISTRE DE CONTROLE 6551
M51CDR	660	$0294	1 $00	RS232 PSEUDO REGISTRE DE COMMANDE 6551
M51AJB	661	$0295	2 $0000	RS232 (DUREE D'UN BIT/2-100)
RSSTAT	663	$0297	1 $00	RS232 PSEUDO REGISTRE D'ETAT 6551
BITNUM	664	$0298	1 $00	RS232 NOMBRE DE BITS A ENVOYER/RECEVOIR
BAUDOF	665	$0299	2 $0000	RS232 NOMBRE DE TOPS HORLOGE POUR UN BIT
RIDBE	667	$029B	1 $00	RS232 INDEX FIN DE TAMPON RECEPTION
RIDBS	668	$029C	1 $00	RS232 INDEX DEBUT DE TAMPON RECEPTION
RODBS	669	$029D	1 $00	RS232 INDEX DEBUT DE TAMPON EMISSION
RODBE	670	$029E	1 $00	RS232 INDEX FIN DE TAMPON EMISSION
	671	$029F	2 $0000	STOCKAGE PROVIS. VECTEUR D'INTERRUPTION PENDANT OPERATIONS CASSETTE
	673	$02A1	94	ZONE INUTILISEE

PAGE 3 **1: ZONE DE VECTEURS INDIRECTS DU BASIC**

	768	$0300	2 $C43A	VECTEUR ROUTINE D'ERREURS
	770	$0302	2 $C483	VECTEUR REDEMARRAGE A CHAUD DU BASIC
	772	$0304	2 $C57C	VECTEUR COMPACTAGE D'UNE LIGNE BASIC
	774	$0306	2 $C71A	VECTEUR IMPRESSION D'UN CARACTERE BASIC
	776	$0308	2 $C7E4	VECTEUR DEBUT DE NOUVELLE COMMANDE BASIC
	778	$030A	2 $C786	VECTEUR EVALUATION D'EXPRESSION
	780	$030C	1 $00	VALEUR TEMPORAIRE DE **A** PENDANT **SYS**
	781	$030D	1 $00	VALEUR TEMPORAIRE DE **X** PENDANT **SYS**
	782	$030E	1 $00	VALEUR TEMPORAIRE DE **Y** PENDANT **SYS**
	783	$030F	1 $00	VALEUR TEMPORAIRE DES FLAGS PENDANT **SYS**

PAGE 3 **2:ZONE DE VECTEURS INDIRECTS DU KERNAL**

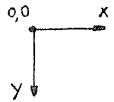
VECINT	788	$0314	2 $EABF	VECTEUR ROUTINE D'INTERRUPTIONS
VECBRK	790	$0316	2	VECTEUR DE **BRK**
VECNMI	792	$0318	2 $FEAD	VECTEUR D'INTERRUPTION **NMI**

VECOPN	794	$031A	2 $F40A	VECTEUR D'OUVERTURE D'UN FICHIER
VECCLO	796	$031C	2 $F34A	VECTEUR DE FERMETURE D'UN FICHIER
VECOPC	798	$031E	2 $F2C7	VECTEUR OUVERTURE D'UN CANAL D'ENTREE
VECCLC	800	$0320	2 $F309	VECTEUR OUVERTURE D'UN CANAL DE SORTIE
VECDEF	802	$0322	2 $F3F3	VECTEUR RETABLISSEMENT DES E/S PAR DEFAUT
VECGET	804	$0324	2 $F20E	VECTEUR LECTURE D'UN CARACTERE SUR FICHIER
VECWRT	806	$0326	2 $F27A	VECTEUR ECRITURE D'UN CARACTERE SUR FICHIER
VECSTP	808	$0328	2 $F770	VECTEUR TEST DE LA TOUCHE STOP
VECKBD	810	$032A	2 $F1F5	VECTEUR PRISE D'UN CARACTERE CLAVIER
VECCLA	812	$032C	2 $F3EF	VECTEUR FERMETURE DE TOUS LES FICHIERS
VECUSR	814	$032E	2 $FED2	VECTEUR COMMANDE BASIC **USR**
VECLOD	816	$0330	2 $F549	VECTEUR CHARGEMENT DE ZONE MEMOIRE (LOGO)
VECSAV	818	$0332	2 $F685	VECTEUR SAUVETAGE DE ZONE MEMOIRE (SAVE)
	820	$0334	8	ZONE INUTILISEE
PAGE3				**3:TAMPON CASSETTE**
	828	$033C	195	TAMPON D'ENTREE/SORTIE CASSETTE
PAGE 4				
	1024	$0400	3072	**EXTENSION RAM 3K**
				LE 6551 N'A PAS ACCES A CETTE ZONE.
PAGES 5 A F				
PAGES 10 A 1D				**MEMOIRE LIBRE POUR L'UTILISATEUR**
	4096	$1000	3583	CONTIENT USUELLEMENT: TEXTE BASIC, VARIABLES, TABLEAUX,ESPACE LIBRE, CHAINES DE CARACTERES. (PEUT AUSSI CONTENIR L'ECRAN SI RAM >8K)
PAGES 1E ET 1F				**MEMOIRE ECRAN**
	7680	$1E00	512	23 LIGNES DE 22 CARACTERES. SI L'EXTENSION MEMOIRE 8K RAM EST PRESENTE, L'ECRAN SE PLACE EN PAGE 10 ($1000)
BLOC 1				**EXTENSION MEMOIRE**
	8192	$2000	8192	EXTENSION DE 8K.SI MEMOIRE RAM, LE KERNAL MODIFIE LES POINTEURS BASIC ET ECRAN
BLOC 2				**EXTENSION MEMOIRE**
	16383	$4000	8192	
BLOC 3				**EXTENSION MEMOIRE**
	24576	$6000	8192	
BLOC 4				**1:GENERATEUR DE CARACTERES**
CARGE1	32768	$8000	1024	MEMOIRE MORTE (ROM) CONTENANT LA REPRESEN- TATION DES 256 CARACTERES EN 8 OCTETS

					CONSECUTIFS CHACUN.
CARGE2	33792	$8400	1024		MAJUSCULES ET GRAPHIQUES
CARGE3	34816	$8800	1024		MAJUSCULES ET GRAPHIQUES INVERSES
CARGE4	35840	$8C00	1024		MAJUSCULES ET MINUSCULES
					MAJUSCULES ET MINUSCULES INVERSEES
BLOC 4					**2:ENTREES/SORTIES**
I/O BLOC 0					**1:PROCESSEUR VIDEO VIC**
VIC-0	36864	$9000	1	$0C	BIT7:ENTRELACEMENT
					BITS0-6:CENTRAGE HORIZ.ECRAN
VIC-1	36865	$9001	1	$26	CENTRAGE VERTICAL ECRAN
VIC-2	36866	$9002	1	$96	BIT7:ADRESSE ECRAN BIT9
					BITS0-6:NOMBRE DE CARACTERES/LIGNE
VIC-3	36867	$9003	1		BIT7:POSITION BALAYAGE ECRAN BIT0
					BITS1-6:NOMBRE DE LIGNES/ECRAN
					BIT0:HAUTEUR CARACT.(SIMPLE=0,DOUBLE=1)
VIC-4	36868	$9004	1		POSITION BALAYAGE ECRAN BITS 1-8
VIC-5	36869	$9005	1	$F0	BITS4-7:ADRESSE MEMOIRE-ECRAN BITS10-13
					BITS0-3:ADRESSE GEN. DE. CARACT BITS10-13
VIC-6	36870	$9006	1	$00	ADRESSE HORIZONTALE **PHOTOSTYLE**
VIC-7	36871	$9007	1	$00	ADRESSE VERTICALE PHOTOSTYLE
VIC-8	36872	$9008	1	$FF	VALEUR ENTREE ANALOGIQUE X
VIC-9	36873	$9009	1	$FF	VALEUR ENTREE ANALOGIQUE Y
VIC-A	36874	$900A	1	$00	FREQUENCE DU GENERATEUR SONORE 1
VIC-B	36875	$900B	1	$00	FREQUENCE DU GENERATEUR SONORE 2
VIC-C	36876	$900C	1	$00	FREQUENCE DU GENERATEUR SONORE 3
VIC-D	36877	$900D	1	$00	FREQUENCE DU GENERATEUR DE BRUIT
VIC-E	36878	$900E	1	$00	BITS4-7:COULEUR AUXILIAIRE (MODE MULTI)
					BITS0-3:VOLUME SONORE
VIC-F	36879	$900F	1	$1B	BITS4-7:COULEUR DU FOND ECRAN
					BIT3:INVERSION COULEURS FOND/BORD
					BITS0-2:COULEUR BORD
	36880	$9010	256		ZONE INUTILISEE (ET INUTILISABLE)
I/O BLOC 0					**2: VIA 1 (NMI)**
VIA1-0	37136	$9110	1	$FF	**PORTB**: BITS0-7: **PORT UTILISATEUR**
					EGALEMENT UTILISE POUR L'RS232 (VOIR
					CE CHAPITRE)
VIA1-1	37137	$9111	1	$7E	**PORTA** (INUTILISE ,VOIR VIA1-F)
VIA1-2	37138	$9112	1	$00	**DDRB**:DIRECTION DES BITS PORT B(SORTIE=1)
VIA1-3	37139	$9113	1	$80	**DDRA**:DIRECTION DES BITS PORT A(ENTREE=0)
VIA1-4	37140	$9114	1		**T1-CL**:MINUTERIE NUMERO 1 (PARTIE BASSE)
VIA1-5	37141	$9115	1		**T1-CH**:MINUTERIE 1 (HAUT): **BAUD RATE**
					VITESSE DE TRANSMISSION RS232
VIA1-6	37142	$9116	1	$55	**T1-LL**:VERROU DE MINUT. 1(PARTIIE BASSE)
VIA1-7	37143	$9117	1	$BF	**T1-LH**:VITESSE DE TRANSMISSION CASSETTE
VIA1-8	37144	$9118	1		**T2-L**:MINUTERIE NUMERO 2 (BASSE)
VIA1-9	37145	$9119	1		**T2-H**:IDEM (HAUT) VITESSE RECEPTION RS232
VIA1-A	37146	$911A	1	$FF	**SR**:REGISTRE A DECALAGE
VIA1-B	37147	$911B	1	$40	**ACR**:REGISTRE DE COMMANDE AUXILIAIRE

```
VIA1-C 37148 $911C   1   $FE      PCR:CONTROLE DE CA1,CA2,CB1,CB2
                                   BITS5-7:CB2-EMISSION RS232
                                   BIT4:CB1
                                   BITS1-3:CA2-MOTEUR CASSETTE
                                   BIT0:CA1-TOUCHE RESTORE
VIA1-D 37149 $911D   1   $00      IFR:INDICATION DE SOURCE D'INTERRUPTIONS
                                   BIT7:NMI -NMI EST GENERE PAR LE VIA1
                                   BIT6: MINUTERIE 1
                                   BIT5: MINUTERIE 2
                                   BIT4: CB1 ENTREE RS232
                                   BIT3: CB2
                                   BIT2: REGISTRE A DECALAGE
                                   BIT1: CA1 TOUCHE RESTORE
                                   BIT0: CA2 MOTEUR CASSETTE
VIA1-E 37150 $911E   1   $82      IER:AUTORISATION D'INTERRUPTIONS
                                   CORRESPONDANT A VIA1-D
VIA1-F 37151 $911F   1   $7E      PORTA:E/S SANS AFFECTER CA1,CA2
                                   BIT7: SORTIE ATN POUR IEEE SERIE
                                   BIT6: TEST TOUCHE CASSETTE ENFONCEE
                                   BIT5: PHOTOSTYLE OU BOUTON DE TIR
                                   BIT4: MANCHE JOY2
                                   BIT3: MANCHE JOY1
                                   BIT2: MANCHE JOY0
                                   BIT1: ENTREE DONNEES IEEE SERIE
                                   BIT0: ENTREE HORLOGE IEEE SERIE

I/O BLOC 0                         3: VIA 2  (IRQ)

VIA2-0 37152 $9120   1   $F7      PORTB:BALAYAGE CLAVIER SORTIE RANGEE

                                   BIT7: EGALEMENT MANCHE JOY3
                                   BIT3: EGALEMENT ECRITURE CASSETTE
VIA2-1 37153 $9121   1   $FF      PORTA:BALAYAGE CLAVIER ENTREE COLONNE
VIA2-2 37154 $9122   1   $FF      DDRB:DIRECTION DES BITS PORT B(SORTIE=1)
VIA2-3 37155 $9123   1   $00      DDRA:DIRECTION DES BITS PORT A(ENTREE=0)
VIA2-4 37156 $9124   1            TI1-CL:MINUTERIE 1 (BAS)
VIA2-5 37157 $9125   1            TI1-CH:IDEM (H) DUREE LECTURE CASSETTE
                                          OU GENERATION DES INTERRUPTIONS
                                          AU 1/60 EME DE SECONDE
VIA2-6 37158 $9126   1   $26      TI1-LL:MINUTERIE 1-VERROU D'ECRITURE
VIA2-7 37159 $9127   1   $48      TI1-LH:IDEM (HAUT)
VIA2-8 37160 $9128   1            T2L: MINUTERIE 2 (BAS)
                                        DELAI TIMEOUT POUR IEEE SERIE
VIA2-9 37161 $9129   1            T2H:IDEM (H) SYNCHRONISATION CASSETTE
VIA2-A 37162 $912A   1   $FF      SR:REGISTRE A DECALAGE
VIA2-B 37163 $912B   1   $40      ACR
VIA2-C 37164 $912C   1   $DE      PCR:
                                   BITS5-7:CB2-SORTIE DONNEES IEEE SERIE
                                   BIT4:CB1-ENTREE SRQ DEMANDE DE SERVICE
                                   BITS1-3:CA2-SORTIE HORLOGE IEEE SERIE
                                   BIT0:CA1-LECTURE CASSETTE

VIA2-D 37165 $912D   1   $00      IFR:INDICATION SOURCE D'INTERRUPTION
                                   BIT7:IRQ GENERE PAR LE VIA2
                                   BIT6:MINUTERIE 1
```

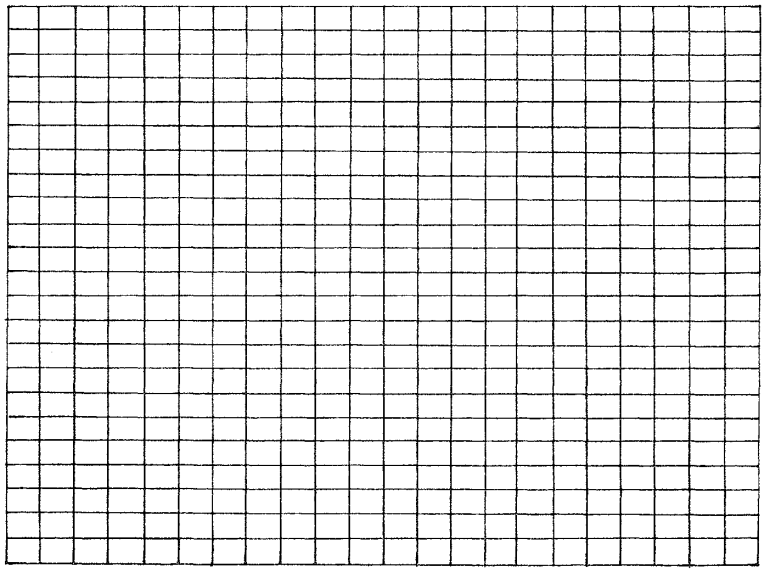
				BIT5:MINUTERIE 2 BIT4:CB1 BIT3:CB2 BIT2:REGISTRE A DECALAGE BIT1:CA1 BIT0:CA2
VIA2-E	37166	$912E	1 $C0	**IER**:AUTORISATION D'INTERRUIPTION CORESPONDANT A VIA2-D
VIA2-F	37167	$912F	1 $FF	**PORTA**:NON UTILISE (VOIR VIA2-1)
	37169	$9130	718	ZONE INUTILISEE (ET INUTILISABLE)
I/O BLOC 1				**MEMOIRE COULEUR ECRAN**
	37888	$9400	512	ATTENTION CES 1024 EMPLACEMENTS-MEMOIRE NE CONTIENNENT QUE 4 BITS (UN **NIBBLE** OU **QUARTET**):BITS 4-7 INEXISTANTS LA RAM COULEUR EST EN $9400 SI ECRAN EN PAGE PAIRE (DIVISIBLE PAR $0400).SINON, RAM COULEUR EN $9600,CAS DU VIC DE BASE.
	38800	$9600	512	
I/O BLOC 2				**EXTENSION D'ENTREES-SORTIES**
	38912	$9800	1024	NON UTILISE
I/O BLOC 3				**EXTENSION D'ENTREES/SORTIES**
	39936	$9C00	1024	NON UTILISE
BLOC 5				**EXTENSION MEMOIRE MORTE**
	40960	$A000	8192	MEMOIRE MORTE (ROM) POUVANT CONTENIR UN PROGRAMME A DEMARRAGE AUTOMATIQUE (VOIR **AUTOSTART**) CETTE ZONE PEUT CONTENIR DE LA MEMOIRE VIVE (RAM) MAIS CELLE-CI NE PEUT ETRE UTILISEE POUR DU PROGRAMME BASIC,NI COMME GENERATEUR DE CARACTERES OU MEMOIRE-ECRAN



Caractère..... 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Point..... 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168

Adresse relative	Caractère	Point
0	0	0
23	1	8
45	2	16
67	3	24
89	4	32
111	5	40
133	6	48
155	7	56
177	8	64
199	9	72
221	10	80
243	11	88
265	12	96
287	13	104
309	14	112
331	15	120
353	16	128
375	17	136
397	18	144
419	19	152
441	20	160
463	21	168
485	22	176



BLOC 6 INTERPRETEUR BASIC

49152	$C000	8192	VOIR BASIC
49152	$C000	2 $E378	ADRESSE DE DEMARRAGE (SITUEE DANS LE KERNAL)
49154	$C002	2 $E467	ADRESSE DE REDEMARRAGE (IDEM)
49156	$C004	8	"CBMBASIC"
			ADRESSES DES ROUTINES BASIC
			L'ADRESSE EST CELLE QUI PRECEDE LE DEBUT DE LA ROUTINE EN QUESTION PARCE QU' ON Y ACCEDE VIA UN RTS
49164	$C00C	2 $C830	END
49166	$C00E	2 $C741	FOR
49168	$C010	2 $CD1D	NEXT
49170	$C012	2 $C8F7	DATA
49172	$C014	2 $CBA4	INPUT£
49174	$C016	2 $CBBE	INPUT
49176	$C018	2 $D080	DIM
49178	$C01A	2 $CC05	READ
49180	$C01C	2 $C9A4	LET
49182	$C01E	2 $C89F	GOTO
49184	$C020	2 $C870	RUN
49186	$C022	2 $C927	IF
49188	$C024	2 $C81C	RESTORE
49190	$C026	2 $C882	GOSUB
49192	$C028	2 $C8D1	RETURN
49194	$C02A	2 $C93A	REM
49196	$C02C	2 $C82E	STOP
49198	$C02E	2 $C94A	ON
49200	$C030	2 $D82C	WAIT
49202	$C032	2 $E164	LOAD
49204	$C034	2 $E152	SAVE
49206	$C036	2 $E161	VERIFY
49208	$C038	2 $D3B2	DEF
49210	$C03A	2 $D823	POKE
49212	$C03C	2 $CA7F	PRINT£
49214	$C03E	2 $CA9F	PRINT
49216	$C040	2 $C856	CONT
49218	$C042	2 $C69B	LIST
49220	$C044	2 $C65B	CLR
49222	$C046	2 $CA85	CMD
49224	$C048	2 $E126	SYS
49226	$C04A	2 $E1BA	OPEN
49228	$C04C	2 $E163	CLOSE
49230	$C04E	2 $CB7A	GET
49232	$C050	2 $C641	NEW
			ADRESSES DES FONCTIONS BASIC
49234	$C052	2 $DC39	SGN
49236	$C054	2 $DCCC	INT
49238	$C056	2 $DC58	ABS
49240	$C058	2 $0000	USR
49242	$C05A	2 $D37D	FRE
49244	$C05C	2 $D39E	POS
49246	$C05E	2 $DF71	SQR

49248	$C060	2	$E094	RND
49250	$C062	2	$D9EA	LOG
49252	$C064	2	$DFED	EXP
49254	$C066	2	$E261	COS
49256	$C068	2	$E268	SIN
49258	$C06A	2	$E2B1	TAN
49260	$C06C	2	$E30B	ATN
49262	$C06E	2	$D80D	PEEK
49264	$C070	2	$D77C	LEN
49266	$C072	2	$D465	STR$
49268	$C074	2	$D7AD	VAL
49270	$C076	2	$D78B	ASC
49272	$C078	2	$D6EC	CHR$
49274	$C07A	2	$D700	LEFT$
49276	$C07C	2	$D72C	RIGHT$
49278	$C07E	2	$D737	MID$
49280	$C080	2	$D869	+ DYADIQUE
49282	$C082	2	$D852	-
49284	$C084	2	$DA2A	*
49286	$C086	2	$DB11	/
49288	$C088	2	$DF7A	^
49290	$C08A	2	$CFE8	AND
49292	$C08C	2	$CFE5	OR
49294	$C08E	2	$DFB3	- MONADIQUE
49296	$C090	2	$CED3	NOT MONADIQUE
49298	$C092	2	$D015	>=<

TABLE DES MOTS-CLE BASIC

LE BIT7 DU DERNIER CARACTERE DE CHAQUE MOT-CLE VAUT 1.

				TOKEN	
49310	$C09E	3	"END	$80	128
49313	$C0A1	3	"FOR	$81	129
49316	$C0A4	4	"NEXT	$82	130
49320	$C0A8	4	"DATA	$83	131
49324	$C0AC	6	"INPUT£	$84	132
49330	$C0B2	5	"INPUT	$85	133
49335	$C0B7	3	"DIM	$86	134
49338	$C0BA	4	"READ	$87	135
49342	$C0BE	3	"LET	$88	136
49345	$C0C1	4	"GOTO	$89	137
43349	$C0C5	3	"RUN	$8A	138
49352	$C0C8	2	"IF	$8B	139
49354	$C0CA	7	"RESTORE	$8C	140
49361	$C0D1	5	"GOSUB	$8D	141
49366	$C0D6	6	"RETURN	$8E	142
49372	$C0DC	3	"REM	$8F	143
49375	$C0DF	4	"STOP	$90	144
49379	$C0E3	2	"ON	$91	145
49381	$C0E5	4	"WAIT	$92	146
49385	$C0E9	4	"LOAD	$93	147
49389	$C0ED	4	"SAVE	$94	148
49393	$C0F1	6	"VERIFY	$95	149
49399	$C0F7	3	"DEF	$96	150
49402	$C0FA	4	"POKE	$97	151
49406	$C0FE	6	"PRINT£	$98	152
49412	$C105	5	"PRINT	$99	153

49417	$C109	4	"CONT	$9A	154
49421	$C10D	4	"LIST	$9B	155
49425	$C111	3	"CLR	$9C	156
49428	$C114	3	"CMD	$9D	157
49431	$C117	3	"SYS	$9E	158
49434	$C11A	4	"OPEN	$9F	159
49438	$C11E	5	"CLOSE	$A0	160
49443	$C123	3	"GET	$A1	161
49446	$C126	3	"NEW	$A2	162

TABLE DES NOMS DE FONCTIONS BASIC

49449	$C129	4	"TAB(	$A3	163
49453	$C12D	2	"TO	$A4	164
49455	$C12F	2	"FN	$A5	165
49457	$C131	4	"SPC(	$A6	166
49461	$C135	4	"THEN	$A7	167
49465	$C139	3	"NOT	$A8	168
49468	$C13C	4	"STEP	$A9	169
49472	$C140	5	"+,"-,"*,"/,"^	$AA,$AB,$AC,$AD,$AE	
49477	$C145	3	"AND	$AF	175
49480	$C148	2	"OR	$B0	176
49482	$C14A	3	">=<	$B1,$B2,$B3	
49485	$C14D	3	"SGN	$B4	180
49488	$C150	3	"INT	$B5	181
49491	$C153	3	"ABS	$B6	182
49494	$C156	3	"USR	$B7	183
49497	$C159	3	"FRE	$B8	184
49500	$C15C	3	"POS	$B9	185
49503	$C15F	3	"SQR	$BA	186
49506	$C162	3	"RND	$BB	187
49509	$C165	3	"LOG	$BC	188
49512	$C168	3	"EXP	$BD	189
49515	$C16B	3	"COS	$BE	190
49518	$C16E	3	"SIN	$BF	191
49521	$C171	3	"TAN	$C0	192
49524	$C174	3	"ATN	$C1	193
49527	$C177	4	"PEEK	$C2	194
49531	$C17B	3	"LEN	$C3	195
49534	$C17E	4	"STR$	$C4	196
49537	$C182	3	"VAL	$C5	197
49541	$C185	3	"ASC	$C6	198
49544	$C188	4	"CHR$	$C7	199
49548	$C18C	5	"LEFT$	$C8	200
49553	$C191	6	"RIGHT$	$C9	201
49559	$C197	4	"MID$	$CA	202
49563	$C19B	2	"GO (POUR GO TO)	$CB	203

MESSAGES D'ERREUR DU BASIC

			LE BIT 7 DU DERNIER CARACTERE DE CHAQUE MESSAGE VAUT 1.	NUMERO DE MESSAGE
49566	$C19E	14	"TOO MANY FILES	0
49580	$C1AC	9	"FILE OPEN	1
49589	$C1B5	13	"FILE NOT OPEN	2
49452	$C1C2	14	"FILE NOT FOUND	3
49616	$C1D0	18	"DEVICE NOT PRESENT	4

```
        49634 $C1E2  14          "NOT INPUT FILE                    5
        49648 $C1F0  15          "NOT OUTPUT FILE                   6
        49663 $C1FF  17          "MISSING FILENAME                  7
        49680 $C210  21          "ILLEGAL DEVICE NUMBER             8
        49701 $C225  16          "NEXT WITHOUT FOR                  9
        49717 $C235   6          "SYNTAX                           10
        49723 $C23B  20          "RETURN WITHOUT GOSUB             11
        49743 $C24F  11          "OUT OF DATA                      12
        49754 $C25A  16          "ILLEGAL QUANTITY                 13
        49770 $C26A   8          "OVERFLOW                         14
        49778 $C272  13          "OUT OF MEMORY                    15
        49471 $C27F  17          "UNDEF'D STATEMENT                16
        49808 $C290  13          "BAD SUBSCRIPT                    17
        49821 $C29D  13          "REDIM'D ARRAY                    18
        49834 $C2AA  16          "DIVISION BY ZERO                 19
        49850 $C2BA  14          "ILLEGAL DIRECT                   20
        49864 $C2C8  13          "TYPE MISMATCH                    21
        49877 $C2D5  15          "STRING TOO LONG                  22
        49892 $C2E4   9          "FILE DATA                        23
        49890 $C2ED  19          "FORMULA TOO COMPLEX              24
        49920 $C300  15          "CAN'T CONTINUE                   25
        49934 $C30E  16          "UNDEF'D FUNCTION                 26
        49950 $C31E   6          "VERIFY                           27
        49956 $C324   4          "LOAD                             28

        50038 $C376  11          "READY
        50049 $C381   8          "BREAK

        50057 $C389              RECHERCHE BLOC FOR DANS LA PILE
        50104 $C3B8              FAIT DE LA PLACE POUR INSERTION LIGNE
        50171 $C3FB              TEST SI PLACE SUFFISANTE DANS LA PILE
RAMTST  50184 $C408              TESTE MEMOIRE DISPONIBLE
ERRS    50229 $C435              ENVOIE UN MESSAGE D'ERREUR SUIVI DE "ERROR"
ERRS2   50247 $C447              ECRIT MESSAGE D'ERREUR QUELCONQUE
READY   50292 $C474              RETOUR A BASIC READY AVEC IMPRESSION DE
                                  MESSAGE. (MODE DIRECT)
NMAIN   50307 $C483              TRAITEMENT D'UNE LIGNE BASIC AU CLAVIER
        50483 $C533              RECONSTRUIT LES POINTEURS DE LIGNES BASIC
GETBAS  50528 $C560              RECOIT UNE LIGNE BASIC AU CLAVIER
NCRNCH  50556 $C57C              ENCODE LES MOTS-CLES BASIC (TOKEN)
        50707 $C613              RECHERCHE UNE LIGNE DE NUMERO CONNU
NEW     50754 $C642              EXECUTE NEW PUIS:
        50784 $C660              EXECUTE CLR
        50830 $C68E              EXECUTE LE PROGRAMME A PARTIR DU DEBUT
        50844 $C69C              EXECUTE LIST
        51010 $C742              EXECUTE FOR
EXEC    51169 $C7E1              EXECUTE UNE COMMANDE BASIC
EXEC2   51181 $C7ED              EXECUTE UNE COMMANDE BASIC (SUITE)
        51229 $C81D              EXECUTE RESTORE
        51244 $C82C              EXECUTE STOP OU END
        51287 $C857              EXECUTE CONT
        51313 $C871              EXECUTE RUN
        51331 $C883              EXECUTE GOSUB
        51360 $C8A0              EXECUTE GOTO ET GO TO
        51410 $C8D2              EXECUTE RETURN (IGNORE FIN DE LIGNE)
```

```
        51435 $C8EB                    IGNORE L'INSTRUCTION DATA
        51462 $C906                    RECHERCHE LE MOT-CLE BASIC SUIVANT
        51465 $C909                    RECHERCHE LA LIGNE BASIC SUIVANTE
        51496 $C928                    EXECUTE IF
        51575 $C93B                    IGNORE UNE LIGNE SI REM
        51636 $C94B                    EXECUTE ON
        51563 $C96B                    ACQUIERT UN NOMBRE ENTIER DANS TEXTE BASIC
        51621 $C9A5                    EXECUTE LET
        51741 $CA1D                    AJOUTE UN CHIFFRE EN ASCII A L'ACC.FL 1
        51756 $CA2C                    SUITE DE LET
        51840 $CA80                    EXECUTE PRINT£
        51846 $CA86                    EXECUTE CMD
        51866 $CA9A                    EXECUTE PRINT
STROUT  51998 $CB1E                    ECRIT UNE CHAINE DE CARACTERES EN MEMOIRE
        52027 $CB3B                    ECRIT UN CARACTERE A L'ECRAN
        52045 $CB4D                    GESTION DES ERREURS DANS INPUT
        52091 $CB7B                    EXECUTE GET
        52133 $CBA5                    EXECUTE INPUT£
        52159 $CBBF                    EXECUTE INPUT
        52217 $CBF9                    IMPRIME MESSAGE ET ATTEND REPONSE
        52230 $CC06                    EXECUTE READ (UTILISE PAR INPUT)
        52476 $CCFC                    ECRIT EXTRA IGNORED OU
                                        REDO FROM START (ERREURS D'INPUT)
        52510 $CD1E                    EXECUTE NEXT
        52600 $CD78                    VERIFIE TYPE DE DONNEES (TYPE MISMATCH)
        52638 $CD9E                    ENTRE ET EVALUE UNE EXPRESSION
EVALU   52867 $CE83                    EVALUE UNE EXPRESSION
EVAL    52977 $CEF1                    SUITE DE L'EVALUATION
        52983 $CEF7                    TESTE PRESENCE DE )
        52986 $CEFA                    TESTE PRESENCE DE (
        52989 $CEFD                    TESTE PRESENCE D'UNE VIRGULE
        53000 $CF08                    ECRIT SYNTAX ERROR
        53005 $CF0D                    ETABLIT UNE FONCTION
        53012 $CF14                    RECHERCHE D'UNE VARIABLE PAR NOM
        53159 $CFA7                    IDENTIFIE LES REFERENCES A UNE FONCTION
ORFAC   53222 $CFE6                    EXECUTE OR
        53225 $CFE9                    EXECUTE AND
ANDFAC  53227 $CFEB                    AND DES DEUX ACCUS FLP
        53270 $D016                    EFFECTUE COMPARAISONS (NUM. ET ALPHA)
        53374 $D07E                    EXECUTE DIM
        53387 $D08B                    RECHERCHE L'ADRESSE D'UNE VARIABLE
        53523 $D113                    TESTE SI UN CARACTERE EST ALPHABETIQUE
        53533 $D11D                    CREE NOUVELLE VARIABLE BASIC
        53652 $D194                    SUITE DE ^ SI VARIABLE TABLEAU
        53669 $D1A5   5 $9080000000    32768 EN FLOTTANT
FLPINT  53674 $D1AA                    EVALUE EXPRESSION POSITIVE ENTIERE
        53713 $D1D1                    TROUVE OU CREE UNE VARIABLE TABLEAU
        54092 $D34C                    CALCULE LE NOMBRE D'ELEMENTS D'UN TABLEAU
FRE     54141 $D37D                    EXECUTE FRE
INTFLP  54161 $D391                    CONVERSION ENTIER-FLOTTANT
        54174 $D39E                    EXECUTE POS
        54182 $D3A6                    TEST SI MODE 'RUN'SINON,ILLEGAL DIRECT
        54195 $D3B3                    EXECUTE DEF
        54241 $D3E1                    TESTE LA SYNTAXE D'UNE FONCTION FN
        54260 $D3F4                    EVALUE UNE FONCTION FN
```

```
        54373 $D465                     EXECUTE STR$
        54389 $D475                     CALCULE UN POINTEUR DE CHAINE
        54407 $D487                     RECHERCHE OU CREE UNE CHAINE
        54516 $D4F4                     PARTIE DE CONSTRUCTION DE CHAINE
        54566 $D526                     RAMASSAGE D'ORDURES
        54717 $D5BD                     RECHERCHE D'UNE CHAINE INUTILE
        54790 $D606                     SUPPRIME UNE CHAINE INUTILE
        54845 $D63D                     CONCATENATION DE CHAINES
        54906 $D67A                     CONSTRUIT UNE CHAINE EN MEMOIRE
        54947 $D6A3                     ANNULE UNE CHAINE DEVENUE INUTILE
        55003 $D6DB                     MISE A JOUR PILE DES DESCRIPTEURS DE CHAINES
        55020 $D6EC                     EXECUTE CHR$
        55040 $D700                     EXECUTE LEFT$
        55084 $D72C                     EXECUTE RIGHT$
        55095 $D737                     EXECUTE MID$
        55137 $D761                     PREND DANS LA PILE LES PARAMETRES POUR MID$
LEN     55164 $D77C                     EXECUTE LEN
        55170 $D782                     SORT DU MODE "GUILLEMETS"
        55179 $D78B                     EXECUTE ASC
        55225 $D79B                     ENTRE UNE VALEUR ENTIERE 0-255
        55213 $D7AD                     EXECUTE VAL
        55275 $D7EB                     ACQUIERT 2 PARAM.POUR POKE ET WAIT
        55287 $D7F7                     CONVERSION FLOTTANT-ENTIER
        55309 $D80D                     EXECUTE PEEK
        55332 $D824                     EXECUTE POKE
        55341 $D82D                     EXECUTE WAIT
        55369 $D849                     AJOUTE .5 A L'ACCU FLP 1
SUBTRA  55376 $D850                     SOUSTRACTION
        55394 $D862                     ADDITION
        55623 $D947                     COMPEMENTE L'ACCU FLP 1
        55678 $D97E                     ECRIT OVERFLOW
        55683 $D983                     MULTIPLIE PAR UN ENTIER 0-255
        55740 $D9BC   5 $8100000000 1 EN FLOTTANT:VALEUR STEP PAR DEFAUT
        55745 $D9C1   1 $03           TAILLE TABLE COEFFICIENTS POLYNOME LOG
        55746 $D9C2   5 $7F5E56CB79 .434255942    (EN FLOTTANT)
        55751 $D9C7   5 $80139B0B64 .576584541
        55756 $D9CC   5 $8076389316 .961800759
        55761 $D9D1   5 $8238AA3B20 2.88539007
        55766 $D9D6   5 $803504F334 .707106781 SQR(2)/2
        55771 $D9DB   5 $813504F334 1.41421356
        55776 $D9E0   5 $8080000000 -.5
        55781 $D9E5   5 $80317217F8 .693147181 LOG(2)
LOG     55786 $D9EA                     EXECUTE LOG
MPLM    55856 $DA30                     MULTIPLICATION
MPLY    55859 $DA33                     MULTIPLICATION DES 2 ACCUS FLP
        55897 $DA59                     MULTIPLIE PAR UN BIT
MOV1    55948 $DA8C                     TRANSFERT MEMOIRE-ACCU FLP 2
        55991 $DAB7                     NORMALISE LES ACCU FLP 1 ET 2
        56020 $DAD4                     GESTION DES DEBORDEMENTS (OVER ET UNDERFLOW)
MPLY10  56034 $DAE2                     MULTIPLIE ACCU FLP 1 PAR 10
        56057 $DAF9   5 $8420000000 10 EN FLOTTANT
DIVD10  56062 $DAFE                     DIVISION PAR 10 DE L'ACCU FLP 1
        56071 $DB07                     EVALUE DIVIDENDE
DIVD    56079 $DB0F                     DIVISION DES 2 ACCUS FLP
        56082 $DB12                     EVALUE DIVISEUR
```

```
MOV2    56226 $DBA2                  TRANSFERT MEMOIRE-ACCU FLP 1
MOV3    56263 $DBC7                  TRANSFERT ACCU FLP 1-MEMOIRE
MOV4    56316 $DBFC                  TRANSFERT ACCU FLP 2-ACCU FLP1
MOV5    56332 $DC0C                  TRANSFERT ACCU FLP 1-ACCU FLP2 AVEC ARRONDI
MOV6    56335 $DC0F                  IDEM SANS ARRONDI
        56347 $DC1B                  ARRONDIT LA VALEUR DANS L'ACCU FLP 1
        56363 $DC2B                  CALCULE LE SIGNE DE L'ACCU FLP 1
SGN     56377 $DC39                  EXECUTE SGN
ABS     56408 $DC58                  EXECUTE ABS
COMPAR  56411 $DC5B                  COMPARAISON ACCU FLP 1-MEMOIRE
        56475 $DC9B                  CONVERSION FLOTTANT-ENTIER
INT     56524 $DCCC                  EXECUTE INT
        56563 $DCF3                  CONVERSION CHAINE-FLOTTANT
        56702 $DD7E                  ACQUIERT UN NOUVEAU CHIFFRE EN ASCII
        56755 $DDB3   5 $9B3EBCF1FD 99999999.9 EN FLOTTANT
        56780 $DDB8   5 $9E6E6B27FD 999999999   (VALEURS POUR CONVERSION
        56785 $DDBD   5 $9E6E6B2800 1E+9         EN NOTATION SCIENTIFIQUE)
        56770 $DDC2                  ECRIT IN NUMERO DE LIGNE EN COURS
LINPRT  56781 $DDCD                  ECRIT LE NUMERO DE LA LIGNE EN COURS
        56797 $DDDD                  CONVERSION FLOTTANT-CHAINE (POUR TI$)
        57105 $DF11   5 $8000000000 .5 EN FLOTTANT (POUR ARRONDI ET SQR)
        57110 $DF16   4 $FA0A1F00   -100000000
        57114 $DF1A   4 $00989680    +10000000 (VALEURS POUR CONVERSION
        57118 $DF1E   4 $1FF0BDC0     -1000000  FLOTTANT-CHAINE)
        57122 $DF22   4 $000186A0      +100000
        57126 $DF26   4 $FFFFD8F0       -10000
        57130 $DF2A   4 $000003E8        +1000
        57134 $DF2E   4 $FFFFFF9C         -100
        57138 $DF32   4 $0000000A          +10
        57142 $DF36   4 $FFFFFFFF           -1
        57146 $DF3A   4 $FFDF0A80     -2160000 (VALEURS POUR CONVERSION
        57150 $DF3E   4 $00034BCC      +216000  DE TI EN TI$)
        57154 $DF42   4 $FFFF7360       -36000
        57158 $DF46   4 $00000E10        +3600
        57162 $DF4A   4 $FFFFFDA8         -600
        57166 $DF4D   4 $0000003C          +60
        57170 $DF52  31              ZONE INUTILISEE
SQR     57201 $DF71                  EXECUTE SQR
POWER   57208 $DF78                  EXECUTE ^
        57268 $DFB4                  EXECUTE - MONADIQUE (NEGATION)
        57279 $DFBF   5 $8138AA3B29 1/LOG(2) EN FLOTTANT
        57280 $DFC0   1 $07          DEBUT TABLE COEFFICIENTS DU POLYNOME EXP
        57285 $DFC5   5 $7134583E56 2.14987637 E-5
        57290 $DFCA   5 $74167EB31B 1.4352314E E-4
        57295 $DFCF   5 $772FEEE385 1.34226348 E-3 (EN FLOTTANT)
        57300 $DFD4   5 $7A1D841C2A 9.61401701 E-3
        57305 $DFD9   5 $7C6359580A .0555051269
        57310 $DFDE   5 $1E75FDE7C6 .240226385
        57315 $DFE3   5 $8031721810 .693147186
        57320 $DFE8   5 $8100000000 1
EXP     57325 $DFED                  EXECUTE EXP

BLOC 7                               1:ROUTINES BASIC DIVERSES
        573E4 $E000
```

```
        57408 $E040                     EVALUATION D'UN POLYNOME
        57482 $E08A    5 $9835447A00    COEFFICIENTS POUR CALCUL DE RND
        57887 $E08F    5 $6828B14600
RND     57492 $E094                     EXECUTE RND
        57590 $E0F6                     ZONE DE CORRECTIONS POUR KERNAL (PATCHES)
COS     57953 $E261                     EXECUTE COS
SIN     57960 $E268                     EXECUTE SIN
TAN     58033 $E2B1                     EXECUTE TAN
        58077 $E2DD    5 $81490FDAA2    .5*PI  EN FLOTTANT
        58082 $E2E2    5 $83490FDAA2    2*PI
        58087 $E2E7    5 $7F00000000    .25
        58092 $E2EC    1 $05            TAILLE DE LA TABLE COEFFICIENTS SIN
        58093 $E2ED    5 $84E61A2D1B    -14.3813907
        58098 $E2F2    5 $862807FBF8    +42.0077971
        58103 $E2F7    5 $8799688901    -76.7041703  EN FLOTTANT
        58108 $E2FC    5 $872335DFE1    +81.6052237
        58113 $E301    5 $86A55DE728    -41.3417021
        58118 $E306    5 $83490FDAA2    +6.28318531
ATN     58123 $E30B                     EXECUTE ATN
        58171 $E33B    1 $0B            TAILLE DE LA TABLE COEFFICIENTS ATN
        58172 $E33C    5 $76B383BDD3    -6.84793912 E-4
        58177 $E341    5 $791EF4A6F5    +4.85094216 E-3
        58182 $E346    5 $7B83FCB010    -.0161117018
        58187 $E34B    5 $7C0C1F67CA    +.034209638
        58192 $E350    5 $7CDE53CBC1    -.0542791328
        58197 $E355    5 $7D1464704C    +.0724571965   EN FLOTTANT
        58202 $E35A    5 $7DB7EA517A    -.0898023954
        58207 $E35F    5 $7D6330887E    +.110932413
        58212 $E364    5 $7E9244993A    -.142839808
        58217 $E369    5 $7E4CCC91C7    +.19999912
        58222 $E36E    5 $7FAAAAAA13    -.33333331
        58227 $E373    5 $8100000000    1
INIBAS  58232 $E378                     INITIALISATION DES VECTEURS EN RAM
CHRGOT  58247 $E387   24                ROUTINE CHARGOT (VA EN RAM EN $73-$8A)
        58271 $E39F    5 $804FC75258    .811635157 EN FLOTTANT
INI2    58276 $E3A4                     INITIALISE PAGE 0 ET CHARGOT POUR BASIC
INI3    58372 $E404                     TESTE RAM ET ECRIT "XXXX BYTES FREE"
        58409 $E429   12                MESSAGE BYTES FREE,$0D,$00
        58421 $E435   26                MESSAGE ****CBM BASIC V2****,$0D,$00
        58447 $E44F   12                VALEURS DES VECTEURS PAGE 3
INI1    58459 $E45B                     INITIALISE LES VECTEURS RAM EN PAGE 3
        58492 $E47C                     ZONE INUTILISEE
DATAHI  58528 $E4A0                     MET A 1 LIGNE DONNEES IEEE SERIE
DATALO  58537 $E4A9                     MET A 0 LIGNE DONNEES IEEE SERIE
        58546 $E4B2                     ATTEND LECTURE STABLE SUR PORTB VIA1
        58556 $E4BC                     ROUTINES DE CORRECTION POUR KERNAL IEEE
        58586 $E4DA 38                  ZONE INUTILISEE

BLOC 7                                  2: ROUTINES KERNAL

        58624 $E500 6911                ROUTINES SYSTEME - VOIR KERNAL

        58624 $E500
SCRSIZ  58629 $E505                     DONNE LA TAILLE DE L'ECRAN
        58634 $E50A                     LIT OU ETABLIT POSITION CURSEUR
```

SETSCR	58648	$E518		MODIFIE ADRESSE ECRAN ET RE-INITIALISE
INIVIC	58819	$E5C3		INITIALISE LES REGISTRES DU **6551**
	58831	$E5CF		PRENDS CARACTERE AU CLAVIER
INPT	58959	$E64F		ENTRE LIGNE AU CLAVIER
DSPLCH	59202	$E742		AFFICHE CARACTERE A L'ECRAN
	59392	$E800		GESTION DES TOUCHES SHIFT
CINV	60095	$EABF		ROUTINE VECTORISEE D'INTERRUPTION
KBDSCN	60190	$EB1E		BALAYAGE CLAVIER
	60380	$EBDC		LOGIQUE DES SHIFT,LOGO,CTRL
KBDTB1	60510	$EC5E	65	TABLE DE DECODAGE CLAVIER **NORMAL**
KBDTB2	60575	$EC9F	65	TABLE DE DECODAGE CLAVIER **SHIFT**
KBDTB3	60640	$ECE0	65	TABLE DE DECODAGE CLAVIER **LOGO**
KBDTB4	60786	$ED72	65	TABLE DE DECODAGE CLAVIER **CTRL**
VICTBL	60899	$EDE3	16	TABLE DES VALEURS INITIALES DU 6551
STALK	60948	$EE14		ENVOIE **TALK** SUR IEEE SERIE
SLSTN	60951	$EE17		ENVOIE **LISTEN** SUR IEEE SERIE
SOUT	60992	$EE40		ENVOIE OCTET SUR IEEE SERIE
	61039	$EE6F		PREPARE POUR ENVOI SUR IEEE
SNDSEC	61120	$EEC0		ENVOIE AD.SEC. SUR IEEE SERIE
ATNLO	61125	$EEC5		**ATN** A ZERO SUR IEEE SERIE
	61134	$EECE		ENVOIE AD. SEC. SUR IEEE SERIE
	61139	$EED3		**ATN** A 1 SUR IEEE SERIE
	61156	$EEE4		SORTIE SUR IEEE SERIE AVEC TAMPON
	61174	$EEF6		ENVOIE **UNTALK** SUR IEEE SERIE
	61190	$EF06		ENVOIE **UNLISTEN** SUR IEEE SERIE
SINP	61209	$EF19		ENTREE SUR IEEE SERIE
CLKLO	61316	$EF84		MET A 0 L'HORLOGE IEEE SERIE
CLKHI	61325	$EF8D		MET A 1 L'HORLOGE IEEE SERIE
DLY1MS	61334	$EF96		DELAI 1 MS
IGETIN	61941	$F1F5		LECTURE VECTORISEE DU CLAVIER
IBASIN	61966	$F20E		ENTREE VECTORISEE D' UN CARACTERE
IBSOUT	62074	$F27A		SORTIE VECTORISEE D' UN CARACTERE
ICHKIN	62151	$F2C7		ENTREE VECTORISEE ALLOCATION CANAL D'ENTREE
ICKOUT	62217	$F309		ENTREE VECTORISEE ALLOCATION CANAL DE SORTIE
ICLOSE	62282	$F34A		FERMETURE VECTORISEE D'UN FICHIER
ICLALL	62447	$F3EF		FERMETURE VECTORISEE DE TOUS LES FICHIERS
ICLRCH	62451	$F3F3		RETABLISSEMENT VECTORISE DES CANAUX E/S
IOPEN	62474	$F40A		OUVERTURE VECTORISEE D' UN FICHIER
LOAD	62786	$F542		**LOAD/VERIFY**
ILOAD	62793	$F549		CHARGEMENT VECTORISE D' UNE ZONE MEMOIRE
SAVE	63093	$F675		**SAVE**
ISAVE	63109	$F685		SAUVETAGE VECTORISE D' UNE ZONE MEMOIRE
FHEAD	63407	$F7AF		RECHERCHE UN BLOC-TITRE SUR CASSETTE
ISTOP	63344	$F770		TEST VECTORISE DE LA TOUCHE STOP
SENSE	63659	$F8AB		LIT INTERRUPTEUR CASSETTE
RECPL	63671	$F8B7		ATTEND **RECORD AND PLAY**
RHEAD	63680	$F8C0		LIT BLOC-TITRE SUR CASSETTE
RLOAD	63689	$F8C9		LIT BLOC-PROGRAMME SUR CASSETTE
WHEAD	63715	$F8E3		ECRIT BLOC-TITRE SUR CASSETTE
TAPE	63732	$F8F4		OPERATIONS CASSETTE
SETTO	63837	$F95D		PREPARE MINUTERIE POUR CASSETTE
CREAD	63886	$F98E		LIT OCTET SUR CASSETTE
BYHAND	64189	$FABD		GESTION D'UN OCTET LU SUR CASSETTE
WCASS	64490	$FBEA		ECRITURE SUR CASSETTE
NMINV	65197	$FEAD		ROUTINE **NMI** VECTORISEE

CBINV	65234	$FED2				ROUTINE VECTORISEE DE **BREAK**
RESET	64802	$FD22				ROUTINE DE DEMARRAGE DU VIC
AUTOTS	64831	$FD3F				TESTE LA PRESENCE DE ROM **AUTOSTART**
MEMTST	64909	$FD8D				INITIALISATION ET TEST MEMOIRE
INIVIA	65017	$FDF9				INITIALISE LES 2 **VIA**
RAMTST	65169	$FE91				TESTE SI UN OCTET EST EN RAM
NMI	65193	$FEA9				ROUTINE D'INTERRUPTION NON MASQUABLE
WARMAO	65220	$FEC4				REDEMARRAGE A CHAUD D'UNE ROM **AUTOSTART**
RPULL	65366	$FF56				FIN DE ROUTINE D'INTERRUPTION
BRKO	65394	$FF72				ROUTINE D'INTERRUPTION/BREAK
						TABLE DE POINTS D'ENTREES SYSTEME
OLDIO	65418	$FF8A	3	JMP	$FD52	RETABLIT LES VECTEURS EN RAM PAR DEFAUT
SETVEC	65421	$FF8D	3	JMP	$FD57	LIT OU IMPOSE LES VECTEURS EN RAM
SETMSG	65424	$FF90	3	JMP	$FE66	ETABLIT MODE DIRECT OU MODE 'RUN'
SECOND	65427	$FF93	3	JMP	$EEC0	ENVOIE ADRESSE SECONDAIRE
STALK	65430	$FF96	3	JMP	$EECE	ENVOIE ADRESSE SEC. APRES **TALK**
MEMTOP	65433	$FF99	3	JMP	$FE73	LIT OU DEFINIT LE SOMMET DE MEMOIRE
MEMBOT	65436	$FF9C	3	JMP	$FE82	LIT OU DEFINIT LE BAS DE MEMOIRE
SCNKEY	65439	$FF9F	3	JMP	$EB1E	LECTURE **CLAVIER**
SETTMO	65442	$FFA2	3	JMP	$FE6F	ETABLIT LE DELAI **TIMEOUT** IEEE
ACPTR	65445	$FFA5	3	JMP	$EF19	ENTREE D'UN OCTET SUR **IEEE SERIE**
CIOUT	65448	$FFA8	3	JMP	$EEE4	ENVOI D'UN OCTET SUR **IEEE SERIE**
UNTLK	65451	$FFAB	3	JMP	$EEF6	ENVOI DE **UNTALK** SUR IEEE SERIE
UNLSTN	65454	$FFAE	3	JMP	$EF04	ENVOI DE **UNLISTEN** SUR IEEE SERIE
LISTEN	65457	$FFB1	3	JMP	$EE17	ENVOI DE **LISTEN** SUR IEEE SERIE
TALK	65460	$FFB4	3	JMP	$EE14	ENVOI DE **TALK** SUR IEEE SERIE
READST	65463	$FFB7	3	JMP	$FE57	LECTURE DE L'OCTET D'ERREUR **ST-STATUS**
SETLFS	65466	$FFBA	3	JMP	$FE50	ETABLIT NUMERO DE FICHIER,AD.PRIM,AD.SEC
SETNAM	65469	$FFBD	3	JMP	$FE49	ETABLIT LE NOM D'UN FICHIER
OPEN	65472	$FFC0	3	JMP($031A)		OUVRE UN FICHIER D'ENTREE/SORTIE
CLOSE	65475	$FFC3	3	JMP($031C)		FERME UN FICHIER D'ENTREE/SORTIE
CKKIN	65478	$FFC6	3	JMP($031E)		ALLOUE LE CANAL D'ENTREE A UN FICHIER
CHKOUT	65481	$FFC9	3	JMP($0320)		ALLOUE LE CANAL DE SORTIE A UN FICHIER
CLRCHN	65484	$FFCC	3	JMP($0322)		REFERME LES CANAUX D'ENTREE/SORTIE
CHRIN	65487	$FFCF	3	JMP($0324)		**LIT UN CARACTERE** SUR CANAL D'ENTREE
CHROUT	65490	$FFD2	3	JMP($0326)		**ENVOI D'UN CARACTERE** SUR CANAL SORTIE
LOAD	65493	$FFD5	3	JMP	$F542	CHARGE UN BLOC DE DONNEES
SAVE	65496	$FFD8	3	JMP	$F675	ENVOI D'UN BLOC DE DONNEES
SETTIM	65499	$FFDB	3	JMP	$F767	MET L'HORLOGE A L'HEURE
RDTIM	65502	$FFDE	3	JMP	$F760	LIT L'HORLOGE
STOP	65505	$FFE1	3	JMP($0328)		TESTE LA TOUCHE **STOP**
GETIN	65508	$FFE4	3	JMP($032A)		LIT UN CARACTERE DANS **TAMPON CLAVIER**
CLALL	65511	$FFE7	3	JMP($032C)		REFERME TOUS LES FICHIERS
UPCLK	65514	$FFEA	3	JMP	$F734	INCREMENTE L'**HORLOGE**
SCREEN	65517	$FFED	3	JMP	$E505	LIT LA TAILLE DE L'ECRAN
PLOT	65520	$FFF0	3	JMP	$E50A	LIT OU IMPOSE LA POSITION DU CURSEUR
IOBASE	65523	$FFF3	3	JMP	$E500	REND L'ADRESSE DES **VIA**
GONMI	65530	$FFFA	2	$FEA9		ADRESSE D'INTERRUPTION NON MASQUABLE
GOIRQ	65532	$FFFC	2	$FD22		ADRESSE DE RESET
GOREST	65534	$FFFE	2	$FF72		ADRESSE D'INTERRUPTION OU DE BREAK

CHAPITRE 6

DIVERS.

HEXADECIMAL

Les notations utilisées tout au long de ce livre sont de deux types : les nombres décimaux et les nombres hexadécimaux. On reconnaît un nombre hexadécimal au fait qu'il est précédé du caractère $. En hexadécimal, la base de comptage est 16 et non 10 comme en décimal. Il faut donc 16 signes différents pour les 16 chiffres. On utilise :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Les conversions décimal-hexa et réciproquement sont relativement simples. On trouvera en annexe les sous-programmes BASIC de conversion dans le programme CHARGEUR HEXA. Si vous utilisez le **MONITEUR 6502**, utilisez les commandes $ et £. Donnons quelques exemples :

Hexa	Décimal	Commentaire
$09	9	
$0A	10	
$0F	15	
$10	16	1 * 16
$100	256	(1 page)
$400	1024	(1 K)
$1000	4096	(4 K)
$2000	8192	(1 bloc) (8 K)
$9120	37152	9*4096+1*256+2*16+0
$FFFF	65535	15*4096+15*256+15*16+15

Le programme **CHARGEUR HEXA** en annexe permet, si vous n'avez pas le **moniteur 6502**, d'introduire et de relire des octets en mémoire en HEXADECIMAL. Notez que le programme **MEMORYSAVE** également en annexe permet de sauver un bloc de données quelconques sur cassette. le bloc mémoire ainsi sauvé sous la forme d'un fichier programme est relisible par LOAD"",1,1.

POINTEURS

Dans la mémoire du VIC, il y a de nombreux pointeurs, c'est-à-dire paires d'octets contenant une adresse. Les deux octets d'un pointeur contiennent chacun la moitié de l'adresse. le premier contient toujours la partie basse, le reste de la division de l'adresse par 256 ou encore les 2 chiffres de droite en notation hexadécimale (partie basse de $1000 =$00). La partie haute est donc le quotient entier de l'adresse par 256 (partie haute de $1000=$10).

En BASIC, on travaille toujours en décimal. Donc pour lire la valeur d'un pointeur, on utilisera le truc suivant (pointeur de début de BASIC)

```
PRINT PEEK(43)+256*PEEK(43+1)
```

qui affiche la valeur du pointeur situé en 43 et 44.

SOMMET DE MEMOIRE

On a souvent besoin de limiter le sommet de mémoire pour se réserver un espace inaccessible au BASIC. On peut mettre dans la zone protégée des images d'écran, des programmes langage machine, etc...

Le pointeur de sommet de BASIC et le pointeur de sommet de mémoire sont en 53 et 55. La modification s'effectue donc par :

```
POKE 53,L: POKE 54,H; POKE 55,L: POKE 56,H :CLR
```

où L et H sont les parties haute et basse du nouveau sommet de mémoire. Le CLR est indispensable pour supprimer les variables chaînes de caractères que le BASIC avait précédemment rangées sous l'ancien sommet de mémoire.

Il peut aussi être très utile de réduire les pointeurs de début et de fin de mémoire ainsi que l'adresse de l'écran pour simuler une configuration VIC de plus faible taille mémoire que la vôtre. Pour cela, on utilise une routine du KERNAL en 64824. Cette routine a besoin de 3 paramètres que l'on trouvera au tableau ci-dessous. On effectuera le chargement de taille-mémoire par :

```
POKE 641,0:POKE 642,X:POKE 643,0:POKE 644°,Y:POKE 648,Z : SYS 64824
```

Configuration VIC	X	Y	Z
de base	16	30	30
+ 3K	4	30	30
+ 8K	18	64	16
+16K	18	96	16
+24K	18	128	16

ERREURS DANS L'INTERPRETEUR BASIC.

Il y a trois erreurs connues dans l'interpréteur BASIC du VIC : une erreur RS 232 (voir ce chapitre) une erreur dans l'instruction INPUT (voir ce chapitre) et une dernière moins grave mais bien curieuse : les numéros de lignes BASIC ne peuvent dépasser 63999 dans le VIC. L'interpréteur BASIC refuse tout numéro de ligne supérieur sauf, et c'est là l'erreur, les numéros de ligne entre 350720 et 360000 environ. Exemple : tapez 350720 <RETURN>. Attention, ce truc réussit très souvent à effacer tout le contenu de la mémoire, y compris les pages 0 et 1, ce qui oblige à couper le courant pour

avoir à nouveau un fonctionnement correct.

ERREUR DU LECTEUR VIC-1540

Pour les possesseurs de lecteurs de disquettes VIC-1540, signalons une erreur gênante dans le programme "RANDOM FILE EXAMPLE" listé dans le mode d'emploi du lecteur et présent sur le disque TEST-DEMO : Le sous-programme qui calcule la position (piste-secteur FT, FS) d'un secteur de numéro donné (F=0 à 598) aux lignes 5330 à 5390 ne fonctionne pas correctement pour les valeurs de F = 471 et 580. Pour ces deux secteurs, la position calculée est incorrecte et dépend des valeurs de calcul F1, F2, F3 qui intervenaient dans le calcul de la position du secteur précédent.

Correction:

Remplacer la ligne 5340 par :

```
IF F>357AND F<472 THEN F1=357:F2=20:F3=19:GOTO 5370
```

et la ligne 5350 par :

```
IF F>471AND F<581THEN F1=471:F2=19:F3=25:GOTO 5370
```

EXPLOSION

Après une explosion de vaisseau spatial dans l'écran de votre VIC, le sol tremblera si vous effectuez le programme suivant :

```
10 FOR I = 1 TO 20
20 POKE 36865, PEEK(36865)+1
30 FORJ = 1 TO 100 : NEXT J
40 POKE 36865, PEEK(36865)-1
50 FORJ = 1 TO 100 :NEXT J
60 NEXT I
```

CERCLES EN HAUTE RESOLUTION

Ils ne seront ronds que si on adapte les valeurs de X à celles de Y en les multipliant par 0,56, faute de quoi on obtient des ellipses. En effet, un point de l'écran du VIC est plus large que haut. (rapport 1 à 0,56)

SI LA MEMOIRE DU VIC EST TROP PETITE

Ce qui arrive souvent avec un VIC de base et des programmes graphiques, par exemple. Que faire ?

1. Supprimer les REM.
2. Supprimer les blancs inutiles.
3. Si on emploie THEN GOTO, l'un des deux est inutile.
4. Derrière NEXT, on peut généralement négliger le nom de variable.
5. Noms de variables en une lettre et non deux ou plus.
6. Si 'OUT OF MEMORY' arrive lors de SAVE, sauver sans utiliser de nom de programme.
7. Réutiliser les variables qui ne servent plus ou les effacer toutes par CLR si plus aucune ne sert. <:CLR> occupe 2 octets tandis qu'une variable en occupe au moins 7.
8. Employer des numéros de lignes plus faibles : GOTO 1000 occupe 5 octets, GOTO 2 seulement 2 octets.
9. Mettre un maximum d'instructions par ligne. Une ligne de moins = 4 octets gagnés.
10. Employer un sous-programme pour les instructions que l'on doit effectuer plus de trois fois en cours de programme.
11. Utiliser un maximum de DEFFN..., ON A GOTO...
12. Remplacer de nombreux DATA par un fichier de données sur cassette (voir caractères programmables).
13. Si rien de tout cela ne suffit, acheter ou construire soi-même une extension mémoire.
14. Si cela ne suffit pas encore, acheter un IBM 370 (on en trouve d'occasion!)

LE VIC COMME TERMINAL

On peut utiliser le VIC avec un interface RS232 et un modem comme terminal d'un gros ordinateur. Il suffit d'ouvrir le canal RS 232, d'y envoyer toute frappe au clavier, et d'envoyer à l'écran tous les caractères reçus... Voir en annexe le programme "terminal".

RESET

La "remise à zéro" du VIC se fait en coupant l'alimentation, avec le bouton RESET de la carte 8K ou en entrant la ligne :

```
SYS64802
```

PROGRAMME INLISTABLE.

Un programme devient inlistable avec :

```
POKE 755,200
```

Il le redevient avec :

```
POKE 755,199
```

De plus, un programme qui contient :

```
POKE 802,0:POKE 803,0:POKE 818,165
```

devient impossible à sauver sur cassette pour autant que l'on ait exécuté les POKE en question, bien sûr! Avec ce truc, il n'y a plus non plus de STOP ou de STOP/RESTORE en fonction.

RENUMEROTER LES LIGNES BASIC.

En utilisant le programme RENUM en annexe, on renumérote le programme. Les GOTO et GOSUB sont à corriger manuellement, mais c'est moins cher que le PROGRAMMER'S AID ou le VICKIT.

CHANGEMENT DE DIAPOSITIVES.

On peut simuler bien des choses avec le VIC. Essayez donc :

```
10 FOR J=5 TO 30:POKE 36864,J:NEXT J
20 FOR J=30 TO 5 STEP-1:POKE 36864,J:NEXT J.
```

TRANSFERER DES PROGRAMMES PET DANS LE VIC

Il suffit de lire la cassette ou le disque PET avec le VIC. Attention, un PET ne peut pas écrire sur un disque formatté avec le VIC, mais il peut lire les disques écrits par le VIC.

TRANSFERER DES PROGRAMMES VIC DANS UN PET/CBM

Si le VIC a 3K d'extension ou moins :

1. Charger le programme dans le PET.
2. taper 10 REM <RETURN> (Première ligne bidon du programme)
3. taper POKE 1025,7 <RETURN>
   POKE 1026,16 <RETURN)
4. taper 10 <RETURN> pour effacer la ligne 10.
5. taper POKE 43, PEEK (43)-12 :CLR

Si le VIC a 8K d'extension ou plus, la procédure est identique, sauf les points 3 et 5 :

3. taper POKE 1025,7 <RETURN>
   POKE 1026,18 <RETURN>
5. Taper POKE 43, PEEK(43)-14:CLR

PROGRAMME DE TRI "QUICKSORT"

On trouvera en annexe le programme **"QUICKSORT"** qui est un algorithme de tri très efficace. Les chaînes de caractères A$(I) sont triées par ordre alphabétique. Le tri d'un tableau de 100 éléments dure environ une demi minute. Avec certaines méthodes moins efficaces, on atteint des durées de plusieurs dizaines de minutes ! En remplaçant A$(I) par A(I), il est possible de trier des nombres.

RECUPERER UN PROGRAMME APRES "NEW" OU "RESET".

Le bouton de RESET est très utile pour sortir des blocages complets de la machine. De même, NEW est utile quand on désire effacer un programme. Par contre, si par erreur on a exécuté l'instruction NEW, on peut à l'aide de diverses modifications de la mémoire du VIC, récupérer le programme. Les deux premiers octets du programme BASIC contenaient (voir le chapitre 2) l'adresse du début de seconde ligne du texte BASIC et NEW les a remis à zéro. Le pointeur de fin de BASIC est rendu égal à celui de début de BASIC. Le programme **DE NEW** en annexe contient 8 lignes à exécuter en mode direct (bien relire chaque ligne avant de frapper RETURN). Expliquons leurs fonctions brièvement :

DE-NEW

Ligne 1 : Modifier l'adresse de début de variable pour la mettre en sommet de mémoire. Sans cette ligne, les variables I et J créées aux lignes suivantes iraient se ranger en mémoire 'par dessus' le programme que l'on veut récupérer.

Ligne 2 : Recherche du premier 0 dans le texte BASIC. A la fin de cette ligne, I contient l'adresse du premier octet de la seconde ligne BASIC.

Ligne 3 : Cette valeur I est convertie sur deux octets et remise à sa place au début de la première ligne BASIC.

Ligne 4 : A ce moment, la commande LIST rend le texte du programme : il est revenu ! Attention, toutefois, tout essai de modification, exécution, sauvegarde de ce programme conduit à des erreurs irrécupérables.

Ligne 5 : On recherche par cette boucle l'adresse de la fin du programme BASIC qui se signale par la présence de 3 octets 00 consécutifs : le zéro de fin de ligne BASIC, suivi de 00 comme adresse de ligne suivante :

c'est la convention pour indiquer "fin de programme". A la fin de cette ligne, I a la valeur exacte qu'il faut réécrire dans le pointeur en fin de BASIC.

Ligne 6 : Comme le pointeur de fin de BASIC est aussi le pointeur de début de variables, on ne peut utiliser de variables pour le modifier. Donc cette ligne encode la valeur I sur deux octets et les sauve en mémoire en 888 et 889 dans le tampon cassette.

Ligne 7 : Les deux valeurs sont écrites dans le pointeur de fin de BASIC début de variables.

Ligne 8 : Il est obligatoire d'effectuer **CLR** ou **RUN** pour restaurer la valeur des autres pointeurs (début de tableaux, etc...) avant d'employer une seule variable. Et voilà ! Tout est comme avant le NEW !

Ce programme DENEW est prévu pour un VIC de base. Pour d'autres tailles mémoire, il faut recalculer les différentes valeurs. Exemple : en VIC de base : 4102 = 4096 + 6 = début de BASIC + 6. Donc, avec extension 3K on emploiera : 1039=1024 + 6= début de BASIC+6, et ainsi de suite.

A la ligne 5, **29** est le numéro de l'avant dernière page de mémoire RAM du VIC. (29*256=7424=sommet de mémoire BASIC-256). Modifiez cette valeur pour être sûr que l'adresse correspondante soit dans l'avant-dernière page de la mémoire RAM.

ANNEXE 1

PROGRAMMES.

Ecrans multiples.

```
10 REM ECRANS MULTIPLES     B.C.M.
20 REM POUR VIC DE BASE OU +3K
50 :
60 POKE54,24:POKE56,24:CLR:DIML(3,33),T(3),U(3),V(3)
70 FORJ=0TO3:READT(J),U(J),V(J):NEXTJ
90 PRINT"■";:FORA=0TO3:GOSUB1000:PRINT"■";:NEXTA
100 :
130 GETA$:IFA$=""GOTO130
140 A=ASC(A$):IF(A>132)AND(A<137)THENA=A-133:GOSUB1000
150 PRINTA$;:GOTO130
160 :
1000 REM COMMUTE VERS UN DES 4 ECRANS
1010 REM AVEC LES TOUCHES PROGRAMMABLES
1020 FORI=0TO23:L(E,I)=PEEK(217+I):NEXTI
1030 E=A:POKE36866,T(A):POKE36869,U(A):POKE648,V(A)
1040 FORI=0TO23:POKE217+I,L(A,I):NEXTI
1050 PRINT"■";
1060 RETURN
2000 :
2010 DATA150,240,30
2020 DATA 22,240,28
2030 DATA150,224,26
2040 DATA 22,224,24

READY.
```

Double alpha

```
10 POKE56,28:CH=32768
20 FORX=6144TO7679STEP2
30 POKEX,PEEK(CH):POKEX+1,PEEK(CH)
40 CH=CH+1:NEXT
50 POKE36867,25:POKE36869,254
60 PRINT"■":FORI=32TO127:PRINTCHR$(I);:NEXT

READY.
```

Mini orgue.

```
10 PRINT"■■■     ■■**************":PRINT"     ** MINI ORGUE *":PRINT"     ********
******■"
20 PRINT"■■■■■■■■■■■■■■DO RE MI FA SOL LA SI ";"■■ 1  2  3  4  5  6  7  ■"
30 NX(1)=225:NX(2)=228:NX(3)=231:NX(4)=232:NX(5)=235:NX(6)=237:NX(7)=239
40 Y=PEEK(60510+PEEK(203))-48
45 IFY=207THENPOKE36878,0:GOTO40
50 POKE36878,15:POKE36876,NX(Y):POKE36875,NX(Y):POKE36874,NX(Y):GOTO40

READY.
```

Poignées de jeu.

```
1 REM DESSIN AVEC POIGNEES DE JEU
2 PRINT"■":EC=PEEK(648)*256:NX=22:NY=23
3 CO=38400+512*(EC/1024=INT(EC/1024)):REM ADRESSE RAM COULEUR
4 FORI=0TONX*NY:POKECO+I,0:NEXTI      :REM COULEUR DE FOND : NOIR
5 POKEEC,160
6 X=PEEK(36872):Y=PEEK(36873)
7 X=INT(X/11.6):Y=INT(Y/11.1)
8 C=EC+Y*NX+X
9 POKEC,160:GOTO6

READY.
```

Thermomètre.

```
10 REM THERMOMETRE
20 :
30 PRINT"■■■■■SHEMA DE MONTAGE■■"
40 PRINT"   7:5V  ←——":PRINTTAB(13)"|":PRINTTAB(13)"␣"
50 PRINTTAB(13)"␣ CTN":PRINTTAB(13)"␣ 220K":PRINTTAB(13)"␣":PRINTTAB(13)"|"
60 PRINT"   9:POTX←—┤":PRINTTAB(13)"|":PRINTTAB(13)"␣"
70 PRINTTAB(13)"␣ RES":PRINTTAB(13)"␣ 220K":PRINTTAB(13)"␣":PRINTTAB(13)"|"
80 PRINT"   8:0V  ←——":PRINT:PRINT"■APPUYEZ SUR UNE TOUCHE"
90 GETA$:IFA$=""THEN90
100 PRINT"■■■TREMPEZ LE THERMOMETRE";
110 PRINT"DANS DE L'EAU AVEC DE LA GLACE PUIS POUSSEZ SUR ■E■"
120 GETA$:IFA$=""THEN120
130 IF A$<>"E"THEN120
140 ZERO=PEEK(36872)
150 PRINT"■■■VOUS POUVEZ EVITER CETETALONNAGE EN NOTANT  LA VALEUR SUIVANTE■"
160 PRINT"   ZERO="ZE
170 PRINT"■LORS D'UNE UTILISATIONULTERIEURE DU PGM IL  VOUS SUFFIRA D'INTRO-"
180 PRINT"-DUIRE CETTE VALEUR ET";:PRINT"DE FAIRE RUN 210"
190 PRINT"■■■APPUYEZ SUR UNE TOUCHE■"
200 GETA$:IFA$=""THEN200
```

```
210 EC=7680:CO=38400:I=0:J=0:K=0:DEFFNA(I)=EC+362-I*22:DEFFNB(I)=CO+362-I*22
220 DIM A(7)
230 A(0)=32:A(1)=109:A(2)=111:A(3)=121:A(4)=98:A(5)=248:A(6)=247:A(7)=227
240 PRINT"■"TAB(10)"_":FORI=1TO12:PRINT TAB(9)"▌▐":NEXTI
250 PRINT TAB(9)"▌▐":PRINT TAB(9)"▐▐":PRINT TAB(9)"▐▐":PRINT TAB(9)"▐▐"
260 PRINTTAB(9)"■■ ■■":PRINTTAB(9)"■  ■":PRINTTAB(9)"■  ■" PRINTTAB(9)"■■ ■■"
270 PRINT"■■":FORI=60 TO 10 STEP-5:PRINT TAB(6)I:NEXTI
280 PRINTTAB(8)"5":PRINT TAB(8)"■■■■■"
290 PRINT TAB(12)"■- 5":PRINT TAB(12)"-10":PRINT TAB(12)"-15" I=0
300 V=PEEK(36872):A=-25/(2E-128):B=25*2E/(2E-128)
310 T=A*V+B:H=(T+15)*8
320 H=(T+15)*8
330 IF H=K THEN 300
340 IF H>K THEN GOSUB 370
350 IF H<K THEN GOSUB 420
360 GOTO300
370 K=K+1:J=J+1
380 IF K>128 THEN K=128
390 IF J>7 THEN J=0:POKE FNA(I),160:POKE FNB(I),2:I=I+1
400 IF I>15 THEN I=15
410 GOSUB470:RETURN
420 K=K-1:J=J-1
430 IF K<0 THEN K=0
440 IF J<0 THEN J=7:I=I-1
450 IF I<0 THEN I=0
460 GOSUB470:RETURN
470 POKE FNA(I),A(J):POKE FNB(I),2:RETURN

READY.
```

Lores.

```
1 REM BASSE RESOLUTION
2 :
3 PRINT"■"
4 EC=PEEK(648)*256
5 NX=22:NY=23
6 CO=38400+512*(EC/1024=INT(EC/1024)):REM ADRESSE RAM COULEUR
7 FORI=0TONX*NY:POKECO+I,0:NEXTI      :REM COUEUR DE FOND : NOIR
8 POKEEC,160
9 :
10 GETA$
11 IFA$="■"THENX=0:Y=0
12 IFA$="■"GOTO3
13 IFA$="■"THENY=Y-1
14 IFA$="■"THENY=Y+1
15 IFA$="■"THENX=X-1
16 IFA$="■"THENX=X+1
17 IFX=NXTHENX=0
18 IFX<0THENX=NX-1
19 IFY=NYTHENY=0
20 IFY<0THENY=NY-1
21 C=EC+Y*NX+X
22 IFA$=""GOTO10
23 IFPEEK(C)=160THENPOKEC,32:GOTO10
24 POKEC,160
25 GOTO10

READY.
```

Block data.

```
10 REM BARGRAPH
20 :
30 O=8126:H=0:I=0:J=0
40 DIMA(11),B(11),C(7)
50 C(0)=32:C(1)=100:C(2)=111:C(3)=121:C(4)=98:C(5)=248:C(6)=247:C(7)=227
60 PRINT"■■■■   RENTREZ VOS DONNEES    POUR LES MOIS DE:"
70 INPUT"JAN";A(0)
80 INPUT"FEV";A(1)
90 INPUT"MAR";A(2)
100 INPUT"AVR";A(3)
110 INPUT"MAI";A(4)
120 INPUT"JUN";A(5)
130 INPUT"JUL";A(6)
140 INPUT"AOU";A(7)
150 INPUT"SEP";A(8)
160 INPUT"OCT";A(9)
170 INPUT"NOV";A(10)
180 INPUT"DEC";A(11)
190 L=A(0)
200 FORI=0TO11
```

```
210 IFA(I)>H THEN H=A(I)
220 IFA(I)<L THEN L=A(I)
230 NEXT I
240 M=(H-L)/20
250 PRINT
260 FOR I=H TO L STEP-M
270 PRINT(INT(I*10))/10
280 NEXT I
290 PRINT"QQQQQQQQQQQQQQQQQQQQQ        _______________"
300 PRINT"       JFMAMJJASOND";
310 M=M/8
320 FOR I=0TO11
330 B(I)=INT((A(I)-L)/M)
340 C=5
350 IF I=0 OR I=2 OR I=4 OR I=6 OR I=8 OR I=10 THEN C=2
360 N=INT(B(I)/8):R=INT((B(I)/8-N)*8)
370 FOR J=0 TO N-1
380 IF N=0 THEN 410
390 POKE G,160:POKE G+30720,C
400 G=G-22
410 NEXT J
420 POKE G,C(R):POKE G+30720,C
430 FORJ=0 TO N-1
440 IF N=0 THEN 460
450 G=G+22
460 NEXT J
470 G=G+1
480 NEXT I
490 GETA$:IFA$=""THEN490

READY.
```

Medres.

```
1 POKE54,28:POKE56,28:CLR:EC=28*256
2 POKE648,28:SYS58648
3 V=36864:NX=48:NY=60:REM NX ET NY PAIRS
4 POKEV,10:POKEV+1,24:POKEV+2,PEEK(V+2)AND128OR(NX/2):POKEV+3,2*INT(NY/2)
5 CO=38400+512*(EC/1024=INT(EC/1024))
6 FORI=0TO(NX*NY/4):POKECO+I,0:POKEEC+I,32:NEXT
7 DIMA(15):FORI=0TO15:READA(I):NEXT:POKEEC,126
8 GETA$
9 IFA$="■"GOTO6
10 IFA$="■"THENX=0:Y=0
11 IFA$="■"THENY=Y-1
12 IFA$="■"THENY=Y+1
13 IFA$="■"THENX=X-1
14 IFA$="■"THENX=X+1
15 IFA$<>""THENB=ASC(A$):IF(B>47)AND(B<57)THENF=1:G=B-49:REM COULEURS
16 IFX=NXTHENX=0
```

```
17 IFX<0THENX=NX-1
18 IFY=NYTHENY=0
19 IFY<0THENY=NY-1
20 S=INT(X/2):T=INT(Y/2)
21 C=EC+NX/2*T+S:CL=PEEK(C)
22 IFG=-1GOTO25
23 H=CO+NX/2*T+S:REM ADRESSE COULEUR
24 POKEA,G
25 FORI=0TO15:IFCL=A(I)THENCL=I:I=15
26 NEXTI
27 U=X-2*S:V=Y-2*T
28 CH=-(U+1)*((V=0)+4*(V=1)):REM CH=1,2,4 OU 8:NUMERO DU CARRE
29 REM                              DANS LE CARACTERE
30 IF(A$="")OR(F=1)THENF=0:GOTO8
31 IFCLANDCH THENPOKEC,A(CLAND(15-CH)):GOTO8
32 POKEC,A(CHORCL):GOTO8
33 DATA32,126,124,226,123,97,255,236,108,127,225,251,98,252,254,160

READY.
```

Caractères programmables.

```
5 REM CARACTERES PROGRAMMABLES
6 REM
7 REM PREPARE SOMMET DE MEMOIRE
10 POKE53,0:POKE54,26:POKE55,0:POKE56,26:CLR
15 REM ADRESSE ECRAN = 6656
16 EC=26:POKE648,EC:SYS58648
19 REM ADRESSE GENERATEUR = 7680
20 POKE36869,(PEEK(36869)AND240)OR15
25 REM RECOPIE CARACTERES ROM -->RAM
30 FORI=0TO1024:POKE7168+I,PEEK(32768+I):NEXTI
35 REM ADRESSE DU CARACTERE NUMERO 48
40 C=7168+48*8
45 REM PROGRAMME LE CARACTERE 48
50 FORI=0TO7:READ            :POKEC+I,A:NEXTI
60 PRINT"0123456789"
70 DATA0,62,127,99,99,127,62,0
```

Hires

```
1 POKE54,18:POKE56,18:CLR
2 V=36864:POKE648,18:SYS58648
3 FORI=0TO191:POKE38400+I,0:POKE4608+I,I:NEXT
4 POKEV,10:POKEV+2,152:POKEV+3,17:POKEV+5,205
5 FORI=5120TO8191:POKEI,0:NEXT:CLR
6 GETA$:IFA$=""THENX=0:Y=0
7 IFA$="Q"GOTO5
8 IFA$="X"THENY=Y+1
9 IFA$="M"THENX=X+1
10 IFA$="N"THENX=X-1
```

```
11 IFA$="□"THENY=Y-1
12 IFX<0THENX=191
13 IFX>191THENX=0
14 IFY<0THENY=127
15 IFY>127THENY=0
16 S=INT(X/8):T=INT(Y/16)
17 C=5120+16*(S+T*24+Y/16-T):B=7-X+S*8
18 IFA$=""GOTO6
19 IFPEEK(C)AND(2↑B)THENPOKEC,PEEK(C)AND(255-2↑B):GOTO6
20 POKEC,PEEK(C)OR(2↑B):GOTO6

READY.


FRE(0)=11
```

Dynahires.

```
10 POKE54,22:POKE56,22:CLR
15 NX=24:NY=15
16 XM=NX*8-1:YM=NY*16-1
20 X=INT(XM/2):Y=INT(YM/2):V=36864
30 POKEV,10:POKEV+1,22:POKEV+2,128+NX:POKEV+15,25
40 POKEV+3,10R(2*NY):POKEV+5,208OR14:GN=6144:EC=5632
50 FORI=0TONY*NX:POKE38400+I,U:NEXT
60 IFU<>0THENGETA$:IFA$<>"□"GOTO60
70 FORI=ECTOEC+NX*NY:POKEI,127:NEXT
80 FORI=GNTOGN+128*16:POKEI,0:NEXT
90 GETA$
100 IFA$="S"THENX=0:Y=0
110 IFA$="□"GOTO50
120 IFA$="M"THENY=Y+1
130 IFA$="□"THENY=Y-1
140 IFA$="M"THENX=X+1
150 IFA$="■"THENX=X-1
160 IFX<0THENX=XM
170 IFX>XMTHENX=0
180 IFY<0THENY=YM
190 IFY>YMTHENY=0
200 S=INT(X/8):T=INT(Y/16)
210 AD=EC+NX*T+S
230 C=PEEK(AD):IFC=127THEN C=N:N=N+1:POKEAD,C
240 IFN>126THENPOKEAD,127:U=7:GOTO50
250 OC=6144+16*(C+Y/16-T):B=7-X+S*8
270 IFA$=""GOTO90
280 IFPEEK(OC)AND(2↑B)THENPOKEOC,PEEK(OC)AND(253-2↑B):GOTO90
290 POKEOC,PEEK(OC)OR(2↑B):GOTO90
```

Multires

```
2 POKE54,20:POKE56,20:CLR:POKE648,20:SYS58648:EC=5120
4 NX=24:NY=30:XM=NX*4-1:YM=NY*8-1
5 X=INT(XM/2):Y=INT(YM/2):V=53264
8 CO=37888
10 POKEV,16:POKEV+1,22:POKEV+2,PEEK(V+2)AND120ORNX
12 POKEV+3,2*NY:POKEV+5,PEEK(V+5)AND240OR14:D=256*24
14 POKEV+14,PEEK(V+14)AND15OR96:POKEV+15,26:U=2
16 FORI=0TONX*NY:POKEEC+I,255:NEXT
18 FORI=CDTOCD+NX*NY:POKEI,8:NEXT
20 FORI=0TO6+2048:POKEI,0:NEXT
22 GETA$
24 IFA$=""THENX=0:Y=0
26 IFA$="Q"GOTO16
28 IFA$="A"THENY=Y+1
30 IFA$="Z"THENY=Y-1
32 IFA$="■"THENX=X+1
34 IFA$="■"THENX=X-1
36 IFX<0THENX=XM
38 IFX>XMTHENX=0
40 IFY<0THENY=YM
42 IFY>YMTHENY=0
44 IFA$="2"THENU=0
46 IFA$="3"THENU=1
48 IFA$="1"THENU=2
50 IFA$="7"THENU=3
52 S=INT(X/4):T=INT(Y/8):AD=EC+NX*T+S
54 C=PEEK(AD):IFC=255THENC=N:N=N+1:POKEAD,C
56 IFN>254GOTO64
58 OC=D+8*(C+Y/8-T):B=6+2*(3*4-N)
59 IFA$=""OR(A$>"0"ANDA$<"9")GOTO22
60 IFPEEK(OC)AND(3*2↑B)THENPOKEOC,PEEK(OC)AND(255-3*2↑B)OR((3-U)*2↑B):GOTO22
62 POKEOC,PEEK(OC)AND(255-3*2↑B)OR(U*2↑B):GOTO22
64 POKEV+15,RND(0)*255:FORI=0TO300:NEXT:GOTO64

READY.
```

Touches programmables.

```
5 REM                    TOUCHES PROGRAMMABLES (0 OU 3K)
10 POKE55,135 :POKE56,29:CLR
30 FORI= 828TO 974:READA:POKEI,A:NEXTI
40 FORI=7560TO7679:READA:POKEI,A:NEXTI
50 SYS828
100
828 DATA32,198,3,165,55,133,251,133:REM          VERSION POUR VIC DE BASE
836 DATA253,165,56,133,252,133,254,169:REM       OU + 3 K
844 DATA49,133,0,169,133,133,1,169
852 DATA13,32,210,255,169,70,32,210:REM          ENTRER LES CHAINES A ATTRIBUER
860 DATA255,165,0,32,210,255,234,234:REM         A CHAQUE TOUCHE
868 DATA234,234,234,169,63,32,210,255:REM        EN REMPLACANT LES 'RETURN'
876 DATA169,32,32,210,255,32,207,255:REM         PAR DES '£'
```

```
894 DATA72,160,0,165,1,145,55,104:REM
892 DATA32,198,3,201,13,240,14,201:REM
900 DATA92,208,2,169,13,145,55,32
908 DATA207,255,76,124,3,230,0,165:REM
916 DATA0,41,1,208,10,24,165,1:REM
924 DATA105,4,133,1,76,170,3,56:REM
932 DATA165,1,233,3,133,1,165,0:REM
940 DATA201,57,144,163,120,169,136,141:REM
948 DATA20,3,169,29,141,21,3,88:REM
956 DATA169,0,133,0,32,68,198,76
964 DATA116,196,166,55,208,2,198,56
972 DATA198,55,96:DATA 165,0,240,59,160,0,177,251
7568 DATA32,235,29,176,12,165,55,197,251,208,21,165,56,197,252,208
7584 DATA15,169,0,133,0,165,253,133,251,165,254,133,252,76,191,234
7600 DATA166,198,177,251,157,119,2,230,198,32,247,29,165,198,201,11
7616 DATA144,204,230,0,76,191,234,165,215,32,235,29,176,3,76,191
7632 DATA 234,165,0,41,1,208,247,160,0,177,251,197,215,208,6,32
7648 DATA247,29,76,142,29,32,247,29,76,217,29,201,133,144,6,201
7664 DATA141,176,2,56,96,24,96,166,251,208,2,198,252,198,251,96

READY.
```

LA TAILLE MAXI DU TOTAL DES 3 CHAINES EST DE 512 OCTETS

APRES STOP/RESTORE , ON PEUT OBTENIR A NOUVEAU LES FONCTIONS PROGRAMMABLES PAR SYS 941 POUR AUTANT QUE LE TAMPON CASSETTE N'AIE PAS ETE UTILISE ENTRE TEMPS.

```
10 REM                    TOUCHES PROGRAMMABLES (8 K)
20 POKE55,136 :POKE56,63:CLR
30 FORI= 828TO 974:READA:POKEI,A:NEXTI
40 FORI=16264TO16383:READA:POKEI,A:NEXTI
50 SYS828
60 :
70 DATA32,198,3,165,55,133,251,133,253,165,56,133,252,133,254,169
80 DATA49,133,0,169,133,133,1,169,13,32,210,255,169,70,32,210
90 DATA255,165,0,32,210,255,234,234,234,234,234,169,63,32,210,255
100 DATA169,32,32,210,255,32,207,255,72,160,0,165,1,145,55,104
110 DATA32,198,3,201,13,240,14,201,92,208,2,169,13,145,55,32
120 DATA207,255,76,124,3,230,0,165,0,41,1,208,10,24,165,1
130 DATA105,4,133,1,76,170,3,56,165,1,233,3,133,1,165,0
140 DATA201,57,144,163,120,169,136,141,20,3,169,63,141,21,3,88
150 DATA169,0,133,0,32,68,198,76,116,196,166,55,208,2,198,56
160 DATA198,55,96,165,0,240,59,160,0,177,251
170 DATA32,235,63,176,12,165,55,197,251,208,21,165,56,197,252,208
180 DATA15,169,0,133,0,165,253,133,251,165,254,133,252,76,191,234
190 DATA166,198,177,251,157,119,2,230,198,32,247,63,165,198,201,11
200 DATA144,204,230,0,76,191,234,165,215,32,235,63,176,3,76,191
210 DATA 234,165,8,41,1,208,247,160,0,177,251,197,215,208,6,32
220 DATA247,63,76,142,63,32,247,63,76,217,63,201,133,144,6,201
230 DATA141,176,2,56,96,24,96,166,251,208,2,198,252,198,251,96

READY.
```

Manche de commande.

```
1 REM   MANCHE DE COMMANDE
2 :
3 :
4 REM H= -1,0,+1   V=-1,0,+1
5 :
6 :
10 POKE37154,127:P=PEEK(37152):POKE37154,255
20 H=((PAND128)=0)
30 P=PEEK(37151)
40 V=((PAND4)=0)-((PAND8)=0)
50 H=((PAND16)=0)-H
60 T=1-SGN((PEEK(37151))AND32)
70 :
80 :
90 :
100 PRINT"H="H"V="V:PRINT:PRINT"BOUTON="T
110 GOTO10

READY.
```

Terminal RS 232.

```
10 REM TERMINAL RS232
20 :
30 OPEN2,2,3,CHR$(166)+CHR$(96)
40 GET#2,A$:PRINT"
50 GETB$
60 IFB$<>""THENPRINT#130,B$;
70 IFB$=CHR$(13)THENPRINT#2,CHR$(10);
80 GET#2,C$
90 PRINTB$;C$;
100 SR=ST:IFSR=0THEN50
110 :
120 PRINT"ERROR: ";
130 IF(PEEK(37151)AND64)=1THEN130
140 CLOSE2:END

READY.
```

Autoprint.

```
 10 REM AUTOPRINT
 20 REM CREE AUTOMATIQUEMENT UNE LIGNE DATA
 30 REM SUPPLEMENTAIRE
 40 REM DANS SON PROPRE PROGRAMME
 50 :
 60 REM LES DONNEES DE L'EXEMPLE SONT LES 8
 70 REM PREMIERS OCTETS DU GENERATEUR DE
 80 REM CARACTERES EN ROM
 90 :
 95 PRINT"
```

```
100 FORI=0TO7:A(I)=PEEK(32768+I):NEXT:NL=1000
110 PRINTNL"DATA";
120 FORI=0TO6:PRINTA(I)"■,";:NEXT:PRINTA(7)
130 PRINT"GOTO 200■"
140 POKE198,2:REM KBDPTR
150 POKE631,13:REM KBDBUF
160 POKE632,13
170 END :REM ICI ON SORT DU PROGRAMME
200 PRINT"■■■■SUITE DU PROGRAMME":END

READY.
```

Renumber.

```
63900 REM RENUMBER
63910 BA=4096
63920 INPUT"DEPART EN";NU
63930 INPUT"PAR PAS DE";IN
63940 IFPEEK(BA+3)=156ANDPEEK(BA+4)=249THENEND
63950 N1=INT(NU/256):N2=NU-N1*256:POKEBA+3,N2:POKEBA+4,N1
63960 IFPEEK(BA+5)THENBA=BA+1:GOTO63960
63970 NU=NU+IN:BA=BA+5:GOTO63940

READY.
```

Superlist.

```
1000 REM    SUPERLIST
1010 REM  B. M I C H E L
1020 REM
1030 REM POUR IMPRIMER:OPEN4,4:CMD4:GOTO1000
1040 :
1050 PRINT"POSITIONNEZ LA        CASSETTE"
1060 OPEN1,1,0,"LISTING":PRINT"□";
1070 TB=49310:REM TABLE DES MOTS-CLE
1080 INPUT#1,AD:AD=AD-1:INPUT#1,ZZ
1090 DIMB$(24),B(24)
1100 FORI=0TO24:READB$(I):READB(I):NEXTI
1110 XP=AD :REM ADRESSE DE DEBUT DE PROGRAMME
1120 :
1130 REM BOUCLE DE LIST
1140 IFST<>0THENCLOSE1:CLOSE4:END
1150 IFAD>ZZ-2THENCLOSE1:CLOSE4:END
1160 GOSUB1490:REM CARACTERE A LISTER
1170 REM FIN DE LIGNE?
1180 IFC=0THENQUOTE=0:GOSUB1400:GOTO1130
1190 REM ENTRE GUILLEMETS ?
1200 IF(C=34)AND(QUOTE=0)THENQUOTE=1:GOTO1230:
1210 IF(C=34)AND(QUOTE=1)THENQUOTE=0
1220 REM MOTS-CLE BASIC
1230 IF(QUOTE=0)AND(C>127)AND(C<204)THENGOSUB1310:GOTO1130
1240 IFQUOTE=0GOTO1280
1250 REM GESTION CARACTERES SPECIAUX
```

```
1260 FORI=0TO15:IFC=B(I)THENPRINTB$(I);:GOTO1290
1270 NEXTI
1280 PRINTCHR$(C);:REM IMPRESSION DU CHARACTERE
1290 GOTO1130
1300 :
1310 REM LISTE MOT-CLE BASIC
1320 T=TB:D=C-128:IFD=0GOTO1350
1330 P=PEEK(T):IFP<128THENT=T+1:GOTO1330
1340 T=T+1:D=D-1:GOTO0
1350 PRINTCHR$((PEEK(T))AND127);:T=T+1
1360 IFPEEK(T-1)>127THEN1380
1370 GOTO1350
1380 RETURN
1390 :
1400 REM DEBUT DE LIGNE
1410 X=XP:REM ADRESSE DE LA LIGNE EN COURS
1420 IFST<>0THENRETURN
1430 IFAD>ZZ-4THENRETURN
1440 GOSUB1530:XP=C:REM ADRESSE LIGNE SUIVANTE
1450 GOSUB1530:L=C:REM NUMERO DE LIGNE
1460 PRINT:PRINT"■";X;"■";XP-X;"■";L;"■";
1470 RETURN
1480 :
1490 REM ACQUIERT UNE VALEUR SUR CASSETTE
1500 INPUT#1,C:AD=AD+1
1510 RETURN
1520 :
1530 REM LIT VALEUR SUR 2 OCTETS SUR CASS.
1540 INPUT#1,A:INPUT#1,B:C=A+256*B:AD=AD+2
1550 RETURN
1560 :
1570 DATA[BAS],17,[REVERSE],18,[HOME],19
1580 DATA[DROITE],29,[NOIR],144,[HAUT],145
1590 DATA[OFF],146,[CLR],147,[GAUCHE],157
1600 DATA[BLANC],5,[ROUGE],28,[CYAN],159
1610 DATA[ROSE],156,[VERT],30,[BLEU],31
1620 DATA[JAUNE],158
1630 DATA[FLECHE],6,[F1],133,[F2],137
1640 DATA[F3],134,[F4],138,[F5],135
1650 DATA[F6],139,[F7],136,[F8],140
```

Progdata.

```
1 REM  PROGDATA
10 OPEN1,1,1,"LISTING"
20 A=PEEK(43)+256*PEEK(44)
30 B=PEEK(45)+256*PEEK(46)
40 PRINT#1,A;CHR$(13);B;CHR$(13);
50 FORI=A-1TOB:PRINT#1,PEEK(I);CHR$(13);:NEXT
60 CLOSE1

READY.
```

Supervariables.

```
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$(I-1)+"#":B%(I)=B%(I-1)+2*I:NEXTI
63500 ZA=PEEK(46)*256+PEEK(45)
63501 ZB=PEEK(48)*256+PEEK(47)
63502 FORZC=ZATOZB-1STEP7
63503 ZD=-((PEEK(ZC)AND128)=128)*2-((PEEK(ZC+1)AND128)=128)
63504 ONZD+1GOSUB63511,63520,63508,63533
63505 GETZA$:IFZA$=""THEN63505
63506 PRINT:NEXTZC:GOTO63536
63507 PRINTCHR$(PEEK(ZC)AND127)CHR$(PEEK(ZC+1)AND127);:RETURN
63508 :
63509 REM VAR.FLOTTANTES
63510 :
63511 ZE=PEEK(ZC+2)-128
63512 ZJ=.5+(PEEK(ZC+3)AND127)/256+PEEK(ZC+4)/256↑2+PEEK(ZC+5)/256↑3+PEEK(ZC+6)/
25614
63513 IFPEEK(ZC+3)>127THENZK=-1
63514 IFPEEK(ZC+3)<128THENZK=1
63515 ZJ=ZK*ZJ*(2↑ZE):IFZS<0THENRETURN
63516 GOSUB63507:PRINT"="ZJ:ZJ=0:PRINT"II":RETURN
63517 :
63518 REM CHAINES
63519 :
63520 ZE=PEEK(ZC+4)*256+PEEK(ZC+3)
63521 ZG=PEEK(ZC+2):IFZS<0THENRETURN
63522 GOSUB63507:PRINT"$:AD="ZE;"L="ZG
63523 GOSUB63507:PRINT"=";
63524 FORZF=ZETOZE+ZG-1:PRINTCHR$(PEEK(ZF));:NEXT:RETURN
63525 :
63526 REM FONCTIONS
63527 :
63528 PRINT"FN";:GOSUB63507:PRINT" :DEF="(PEEK(ZC+2)+PEEK(ZC+3)*256)
63529 PRINT"FN";:GOSUB63507:PRINT" :VAR="(PEEK(ZC+4)+PEEK(ZC+5)*256):PRINT"II":R
ETURN
63530 :
63531 REM VAR. ENTIERES
63532 :
63533 ZK=-((PEEK(ZC+2)AND128)=128)
63534 IFZKTHENZJ=PEEK(ZC+2)*256+PEEK(ZC+3)-65536:GOTO63536
63535 ZJ=PEEK(ZC+2)*256+PEEK(ZC+3)
63536 IFZS>0THENRETURN
63537 GOSUB63507:PRINT"%="ZJ:ZJ=0:PRINT"II":RETURN


READY.
```

Supertableaux.

```
63538 :
63539 REM SUPERTABLEAUX
63540 :
63541 REM CREATION DES VARIABLES
63542 :
63543 ZD$="":ZM=0:ZN=0:ZL=1:ZD=0:ZQ=0:ZS=0:DIMZP(4),ZQ(4)
63544 ZB=PEEK(48)*256+PEEK(47)
63545 ZM=PEEK(56)*256+PEEK(49):ZN=ZB
63546 IFZN=ZMTHENEND
63547 ZQ=PEEK(ZN+4)
63548 FORZF=1TOZQ
63549 ZP(ZF)=PEEK(ZN+4+2*ZF)+PEEK(ZN+5+2*ZF)*256-1:ZQ(ZF)=ZP(ZF):NEXT
63550 ZD=-((PEEK(ZN)AND128)=128)*2-((PEEK(ZN+1)AND128)=128)
63551 ZB$=CHR$(PEEK(ZN)AND127)+CHR$(PEEK(ZN+1)AND127)
63552 IFZD=1THENZD$=ZB$+"$"
63553 IFZD=3THENZB$=ZB$+"%"
63554 PRINT" DIM ";:GOSUB63575:PRINT" ":GOTO63555
63555 FORZF=1TOZQ:ZL=ZL*(ZP(ZF)+1):NEXT
63556 FORZF=1TOZQ:ZP(ZF)=0:NEXT
63557 ZC=ZN+(2*ZQ)+3
63558 FORZS=1TOZL
63559 GOSUB63576
63560 IFZD=0THENGOSUB63511:ZC=ZC+5
63561 IFZD=1THENGOSUB63520:GOSUB63572:ZC=ZC+3
63562 IFZD=3THENGOSUB63533:ZC=ZC+2
63563 IFZD<>1THENPRINT"=";ZJ
63564 FORZF=ZQTO1STEP-1
63565 ZP(ZF)=ZP(ZF)+1
63566 IFZP(ZF)>>ZQ(ZF)THENZP(ZF)=0:GOTO63568
63567 ZF=1
63568 NEXTZF
63569 GETZA$:IFZA$=""THEN63569
63570 NEXTZS
63571 ZN=ZC+2:ZL=1:GOTO63546
63572 PRINT":AD="ZE;"L="ZG
63573 GOSUB63576:PRINT"=";
63574 FORZF=ZETOZE+ZG-1:PRINTCHR$(PEEK(ZF));:NEXT:PRINT" ":RETURN
63575 ZB$=ZB$+"("
63576 PRINTZB$;
63577 FORZF=ZQTO1STEP-1:PRINTZP(ZF)"■,";:NEXT:PRINT"■)":RETURN

READY.
```

Moniteur 6502

```
MONITEUR 6502       PAGE A0  CHECKSUM= $7BAE ( 31662)

  $A000   09 A0 C7 FE 41 30 C3 C2 CD 20 8D FD 20 52 FD 20
  $A010   18 E5 20 19 A3 A9 DF A2 A5 8D 16 03 8E 17 03 AD
  $A020   14 03 AE 15 03 C9 91 D0 04 E0 A3 F0 09 8D 60 03
  $A030   8E 61 03 20 94 A8 A9 75 85 B2 A9 80 8D 8A 02 85
  $A040   9D A2 D6 20 5D AE 8E 48 03 8E 64 03 58 00 CE 3D
  $A050   03 D0 03 CE 3C 03 20 AE A5 A2 42 A9 2A 4C 3D A9
  $A060   A9 3F 20 D2 FF A9 00 2C A9 0F 8D 0E 90 20 AE A5
  $A070   A9 2E 20 D2 FF A9 00 8D 4E 03 8D 56 03 8D 64 03
  $A080   A2 7F 9A 20 8C A8 C9 2E F0 F9 C9 20 F0 F5 A2 24
  $A090   DD 90 AF D0 13 8D 49 03 8A 0A AA BD B5 AF 85 FB
  $A0A0   BD B6 AF 85 FC 6C FB 00 CA 10 E5 4C 60 A0 A2 02
  $A0B0   D0 02 A2 00 B4 FB D0 09 B4 FC D0 03 EE 56 03 D6
  $A0C0   FC D6 FB 60 A9 00 8D 4E 03 20 13 A2 A2 09 20 38
  $A0D0   A9 CA D0 FA 60 A2 02 B5 FA 48 BD 53 03 95 FA 68
  $A0E0   9D 53 03 CA D0 F1 60 AD 54 03 AC 55 03 4C F4 A0
  $A0F0   A5 FD A4 FE 38 E5 FB 8D 53 03 98 E5 FC A8 0D 53

MONITEUR 6502       PAGE A1  CHECKSUM= $7DF7 ( 32247)

  $A100   03 60 A9 00 F0 02 A9 01 8D 57 03 20 CB A7 20 AE
  $A110   A5 20 F0 A0 20 21 A8 90 18 20 E7 A0 90 7F 20 59
  $A120   A1 E6 FD D0 02 E6 FE 20 1F A9 AC 56 03 D0 6E F0
  $A130   E8 20 E7 A0 18 AD 53 03 65 FD 85 FD 98 65 FE 85
  $A140   FE 20 D5 A0 20 59 A1 20 E7 A0 B0 51 20 AE A0 20
  $A150   B2 A0 AC 56 03 D0 46 F0 EB A2 00 A1 FB AC 57 03
  $A160   F0 02 81 FD C1 FD F0 0B 20 F8 A7 20 38 A9 20 E1
  $A170   FF F0 2A 60 20 E6 A7 20 A1 A9 F0 1E AE 56 03 D0
  $A180   1C 20 F0 A0 90 17 60 20 54 A8 8D 4B 03 20 7C A1
  $A190   AD 4B 03 81 FB 20 1F A9 D0 F3 4C 60 A0 4C 68 A0
  $A1A0   20 74 A1 20 8C A8 C9 27 D0 12 20 8C A8 9D 65 03
  $A1B0   E8 20 A4 A9 F0 20 E0 20 D0 F3 F0 1A 8E 59 03 20
  $A1C0   5F A8 90 D6 9D 65 03 E8 20 A4 A9 F0 09 20 57 A8
  $A1D0   90 C8 E0 20 D0 EE 8E 4A 03 20 AE A5 A2 00 A0 00
  $A1E0   B1 FB DD 63 03 D0 0A C8 E8 EC 4A 03 D0 F2 20 68
  $A1F0   A1 20 1F A9 20 7C A1 B0 E3 20 2B A4 20 F0 A0 90

MONITEUR 6502       PAGE A2  CHECKSUM= $77B0 ( 30640)

  $A200   0D A0 2C 20 C4 A0 20 6F A2 20 E1 FF D0 EE 20 B6
  $A210   A5 D0 8A 20 2D A9 20 F8 A7 20 38 A9 20 C9 AD 48
  $A220   20 CF A2 68 20 E6 A2 A2 06 E0 03 D0 14 AC 4D 03
  $A230   F0 0F AD 58 03 C9 E8 B1 FB B0 1D 20 65 A2 88 D0
  $A240   F1 0E 58 03 90 0E BD E9 AE 20 99 A5 BD EF AE F0
  $A250   03 20 99 A5 CA D0 D2 60 20 7B A2 AA E8 D0 01 C8
  $A260   98 20 65 A2 8A 8E 4A 03 20 FF A7 AE 4A 03 60 AD
  $A270   4D 03 20 7A A2 85 FB 84 FC 60 38 A4 FC AA 10 01
  $A280   88 65 FB 90 01 C8 60 A8 4A 90 0B 4A B0 17 C9 22
```

```
$A290   F0 13 29 07 09 80 4A AA BD 98 AE B0 04 4A 4A 4A
$A2A0   4A 29 0F D0 04 A0 80 A9 00 AA BD DC AE 8D 58 03
$A2B0   29 03 8D 4D 03 98 29 8F AA 98 A0 03 E0 8A F0 0B
$A2C0   4A 90 08 4A 4A 09 20 88 D0 FA C8 88 D0 F2 60 B1
$A2D0   FB 20 65 A2 A2 01 20 CE A0 CC 4D 03 C8 90 F0 A2
$A2E0   03 C0 03 90 F1 60 A8 B9 F6 AE 8D 54 03 B9 36 AF
$A2F0   8D 55 03 A9 00 A0 05 0E 55 03 2E 54 03 2A 88 D0

MONITEUR 6502       PAGE A3  CHECKSUM= $737A ( 29562)

$A300   F6 69 3F 20 D2 FF CA D0 EA 4C 38 A9 20 E6 A7 A9
$A310   03 20 9E A3 A0 2C 4C 3C A5 20 F9 FD A9 80 8D 13
$A320   91 20 3A A3 A9 FF 8D 12 91 A5 FB A0 18 20 34 A3
$A330   A5 FF A0 14 8D 10 91 8C 11 91 A0 1C 8C 11 91 60
$A340   20 54 A8 85 FF 20 AE A5 20 6E A1 20 7C A1 20 1C
$A350   A3 AD 49 03 0A 08 90 17 A1 FB 8D 10 91 78 A9 C4
$A360   8D 19 91 A9 3C 8D 11 91 A9 20 2C 1D 91 F0 FB 20
$A370   3A A3 58 8E 12 91 A9 0C 8D 11 91 AD 10 91 28 B0
$A380   04 10 02 81 FB C1 FB F0 03 20 68 A1 20 1F A9 D0
$A390   B7 A9 AC 48 A9 77 48 08 48 48 48 6C 60 03 8D 4B
$A3A0   03 48 20 8C A8 20 00 A9 D0 F8 68 49 FF 4C 72 A2
$A3B0   20 2B A4 AE 56 03 D0 0D 20 F0 A0 90 08 20 C8 A3
$A3C0   20 E1 FF D0 EE 4C 0E A2 20 AE A5 A2 2E A9 3A 20
$A3D0   0E A8 20 38 A9 20 F8 A7 A9 08 20 EA A8 A9 08 20
$A3E0   AB A3 20 38 A9 20 38 A9 A9 12 20 D2 FF A0 08 A2
$A3F0   00 A1 FB 29 7F C9 20 B0 02 A9 2E 20 D2 FF C9 22

MONITEUR 6502       PAGE A4  CHECKSUM= $7C53 ( 31827)

$A400   F0 04 C9 62 D0 06 20 E2 AA 20 E5 AA 20 1F A9 88
$A410   D0 DF 4C DF AA 20 E6 A7 A9 08 20 9E A3 20 B6 A5
$A420   20 C8 A3 A9 3A 8D 77 02 4C 48 A5 20 E6 A7 85 FD
$A430   86 FE 20 A4 A9 F0 03 20 EB A7 4C AE A5 20 31 A8
$A440   85 FD 86 FE A2 00 8E 66 03 20 8C A8 C9 20 F0 F4
$A450   9D 4F 03 E8 E0 03 D0 F1 CA 30 14 BD 4F 03 38 E9
$A460   3F A0 05 4A 6E 66 03 6E 65 03 88 D0 F6 F0 E9 A2
$A470   02 20 A4 A9 F0 22 C9 3A F0 1E C9 20 F0 F3 20 90
$A480   A5 B0 0F 20 6C A8 A4 FB 84 FC 85 FB A9 30 9D 65
$A490   03 E8 9D 65 03 E8 D0 D9 8E 54 03 A2 00 8E 56 03
$A4A0   A2 00 8E 4B 03 AD 56 03 20 87 A2 AE 58 03 8E 55
$A4B0   03 AA BD 36 AF 20 70 A5 BD F6 AE 20 70 A5 A2 06
$A4C0   E0 03 D0 14 AC 4D 03 F0 0F AD 58 03 C9 E8 A9 30
$A4D0   B0 1E 20 6D A5 88 D0 F1 0E 58 03 90 0E BD E9 AE
$A4E0   20 70 A5 BD EF AE F0 03 20 70 A5 CA D0 D2 F0 06
$A4F0   20 6D A5 20 6D A5 AD 54 03 CD 4B 03 D0 7F 20 21
```

```
MONITEUR 6502       PAGE A5  CHECKSUM= $6C1C ( 27676)

  $A500   A8 AC 4D 03 F0 2F AD 55 03 C9 9D D0 20 20 F0 A0
  $A510   90 01 88 C8 D0 6F 98 2A AE 53 03 E0 82 A8 D0 03
  $A520   B0 03 38 B0 60 CA CA 8A AC 4D 03 D0 03 B9 FC 00
  $A530   91 FB 88 D0 F8 AD 56 03 91 FB A0 41 8C 77 02 20
  $A540   B6 A5 20 C4 A0 20 6F A2 A9 20 8D 78 02 8D 7D 02
  $A550   A5 FC 20 9F A5 8E 79 02 8D 7A 02 A5 FB 20 9F A5
  $A560   8E 7B 02 8D 7C 02 A9 07 85 C6 4C 68 A0 20 70 A5
  $A570   8E 4A 03 AE 4B 03 DD 65 03 F0 0D 68 68 EE 56 03
  $A580   F0 03 4C A0 A4 4C 60 A0 E8 8E 4B 03 AE 4A 03 60
  $A590   C9 30 90 03 C9 47 60 38 60 CD 4E 03 D0 1A 60 48
  $A5A0   4A 4A 4A 4A 20 17 A8 AA 68 29 0F 4C 17 A8 A9 0D
  $A5B0   20 D2 FF A9 0A 2C A9 91 4C D2 FF 8D 3F 03 08 68
  $A5C0   29 EF 8D 3E 03 8E 40 03 8C 41 03 68 18 69 01 8D
  $A5D0   3D 03 68 69 00 8D 3C 03 A9 80 8D 48 03 D0 26 A9
  $A5E0   C0 8D 2E 91 A9 3F 8D 2E 91 20 94 A8 D8 68 8D 41
  $A5F0   03 68 8D 40 03 68 8D 3F 03 68 8D 3E 03 68 8D 3D

MONITEUR 6502       PAGE A6  CHECKSUM= $6974 ( 26996)

  $A600   03 68 8D 3C 03 AD 14 03 8D 44 03 AD 15 03 8D 43
  $A610   03 BA 8E 42 03 58 AD 3E 03 29 10 F0 03 4C 4E A0
  $A620   2C 48 03 50 1F AD 3C 03 CD 5B 03 D0 6B AD 3D 03
  $A630   CD 5A 03 D0 63 AD 5E 03 D0 5B AD 5F 03 D0 53 A9
  $A640   80 8D 48 03 30 12 4E 48 03 90 D2 AE 42 03 9A A9
  $A650   A5 48 A9 BA 48 4C 06 A7 20 AE A5 20 14 A9 8D 4B
  $A660   03 A0 00 20 F2 A8 AD 3D 03 AE 3C 03 85 FB 86 FC
  $A670   20 38 A9 A9 24 8D 4E 03 20 16 A2 20 E4 FF F0 FB
  $A680   C9 03 D0 03 4C 68 A0 C9 4A D0 4E A9 01 8D 48 03
  $A690   D0 47 CE 5F 03 CE 5E 03 AD 21 91 C9 FE D0 3A A2
  $A6A0   53 4C 5B A0 A9 00 F0 12 AD 5C 03 AE 5D 03 8D 5E
  $A6B0   03 8E 5F 03 A9 40 D0 02 A9 80 8D 48 03 20 A4 A9
  $A6C0   F0 0F C9 20 D0 56 20 45 A8 20 E3 A8 20 A4 A9 D0
  $A6D0   4B 20 AE A5 AD 48 03 F0 1F 78 A9 A0 8D 2E 91 A9
  $A6E0   5F 8D 2E 91 A9 DF A2 A5 8D 44 03 8E 43 03 A9 49
  $A6F0   A2 00 8D 28 91 8E 29 91 AE 42 03 9A 78 AD 44 03

MONITEUR 6502       PAGE A7  CHECKSUM= $708F ( 28815)

  $A700   AE 43 03 20 98 A8 AD 3C 03 48 AD 3D 03 48 AD 3E
  $A710   03 48 AD 3F 03 AE 40 03 AC 41 03 40 4C 60 A0 20
  $A720   31 A8 8D 5A 03 8E 5B 03 A9 00 8D 5C 03 8D 5D 03
  $A730   20 42 A8 8D 5C 03 8E 5D 03 4C 68 A0 20 CB A7 8D
  $A740   62 03 8E 63 03 20 42 A8 8D 4F 03 8E 50 03 20 42
  $A750   A8 8D 51 03 8E 52 03 20 A4 A9 F0 0A 20 CF FF C9
  $A760   57 D0 03 EE 4E 03 20 21 A8 AE 56 03 D0 18 20 E7
  $A770   A0 90 13 AC 4E 03 D0 1A B1 FB 20 87 A2 AA BD F6
  $A780   AE D0 06 20 C4 A0 4C 68 A0 AC 4D 03 C0 02 D0 33
  $A790   F0 03 8C 4D 03 88 38 B1 FB AA ED 4F 03 C8 B1 FB
  $A7A0   ED 50 03 90 1E 88 AD 51 03 F1 FB C8 AD 52 03 F1
  $A7B0   FB 90 10 88 18 8A 6D 62 03 91 FB C8 B1 FB 6D 63
```

```
   $A7C0   03 91 FB 20 1F A9 88 10 FA 30 9E 20 31 A8 85 FD
   $A7D0   86 FE 20 42 A8 8D 54 03 8E 55 03 20 8C A8 20 45
   $A7E0   A8 85 FB 86 FC 60 20 31 A8 B0 F6 20 45 A8 B0 03
   $A7F0   20 42 A8 85 FD 86 FE 60 A5 FC 20 FF A7 A5 FB 48

MONITEUR 6502       PAGE A8  CHECKSUM= $7335 ( 29493)

   $A800   4A 4A 4A 4A 20 17 A8 AA 68 29 0F 20 17 A8 48 8A
   $A810   20 D2 FF 68 4C D2 FF 18 69 F6 90 02 69 06 69 3A
   $A820   60 A2 02 B5 FA 48 B5 FC 95 FA 68 95 FC CA D0 F3
   $A830   60 A9 00 8D 59 03 20 8C A8 C9 20 F0 F9 20 6C A8
   $A840   B0 08 20 8C A8 20 57 A8 90 07 AA 20 57 A8 90 01
   $A850   60 4C 60 A0 20 74 A1 A9 00 8D 59 03 20 8C A8 C9
   $A860   20 D0 09 20 8C A8 C9 20 D0 0F 18 60 20 81 A8 0A
   $A870   0A 0A 0A 8D 59 03 20 8C A8 20 81 A8 0D 59 03 38
   $A880   60 C9 3A 08 29 0F 28 90 02 69 08 60 20 A4 A9 D0
   $A890   FA 4C 65 A0 A9 91 A2 A3 8D 14 03 8E 15 03 60 20
   $A8A0   A4 A9 F0 37 20 E6 A7 A5 FB 05 FC F0 22 A5 9A C9
   $A8B0   03 D0 9E A5 FB 8D 93 02 A5 FC 8D 94 02 A9 02 AA
   $A8C0   A8 20 BA FF 20 C0 FF A2 02 20 C9 FF 4C 75 A0 A9
   $A8D0   02 20 C3 FF A9 03 85 9A 4C 68 A0 A5 9A C9 03 F0
   $A8E0   DC D0 F1 8D 3D 03 8E 3C 03 60 8D 4B 03 A0 00 20
   $A8F0   38 A9 B1 FB 20 FF A7 20 1F A9 CE 4B 03 D0 F0 60

MONITEUR 6502       PAGE A9  CHECKSUM= $7F8E ( 32654)

   $A900   20 57 A8 90 08 A2 00 81 FB C1 FB D0 69 20 1F A9
   $A910   CE 4B 03 60 A9 3E 85 FB A9 03 85 FC A9 05 60 E6
   $A920   FB D0 09 E6 FF E6 FC D0 03 EE 56 03 60 98 48 20
   $A930   AE A5 68 A2 2E 20 0E A8 A9 20 4C D2 FF 20 0E A8
   $A940   A2 00 BD 76 AF 20 D2 FF E8 E0 1C D0 F5 A0 3B 20
   $A950   2D A9 AD 3C 03 20 FF A7 AD 3D 03 20 FF A7 20 38
   $A960   A9 AD 43 03 20 FF A7 AD 44 03 20 FF A7 20 14 A9
   $A970   20 EA A8 4C 68 A0 4C 60 A0 20 31 A8 20 E3 A8 20
   $A980   42 A8 8D 44 03 8E 43 03 20 14 A9 8D 4B 03 20 8C
   $A990   A8 20 00 A9 D0 F8 F0 DB 20 CF FF C9 20 F0 F9 D0
   $A9A0   06 20 F0 A7 20 CF FF C9 0D 60 A0 01 84 BA A9 00
   $A9B0   A2 65 A0 03 20 BD FF A8 20 E6 A7 AD 49 03 C9 53
   $A9C0   D0 08 20 A4 A9 F0 AF 20 EB A7 20 98 A9 F0 2D C9
   $A9D0   22 D0 A3 20 CF FF C9 22 F0 0B 91 BB E6 B7 C8 C0
   $A9E0   11 90 F0 B0 91 20 A4 A9 F0 12 20 57 A8 29 0F F0
   $A9F0   85 C9 03 F0 81 85 BA 20 98 A9 D0 D5 AD 49 03 C9

MONITEUR 6502       PAGE AA  CHECKSUM= $7881 ( 30849)

   $AA00   53 D0 0C A9 FB A6 FD A4 FE 20 D8 FF 4C 68 A0 49
   $AA10   4C F0 02 A9 01 A6 FB A4 FC 20 D5 FF A5 90 29 10
   $AA20   F0 EA A9 69 A0 C3 20 1E CB 4C 60 A0 20 E6 A7 20
   $AA30   A5 A0 4C 68 A0 20 E6 A7 20 1F A9 20 1F A9 20 F0
   $AA40   A7 20 38 A9 20 F0 A0 90 0A 98 D0 15 AD 53 03 30
   $AA50   10 10 08 C8 D0 0B AD 53 03 10 06 20 FF A7 4C 68
```

```
$AA60   A0 4C 60 A0 20 E6 A7 20 7A AA 4C 68 A0 20 AE A5
$AA70   A2 2E A9 24 20 0E A8 20 F8 A7 20 EA AA 20 A0 AA
$AA80   20 38 A9 20 86 AA 20 89 AA 20 38 A9 A2 04 A9 30
$AA90   18 0E 54 03 2E 55 03 69 00 20 D2 FF CA D0 EF 60
$AAA0   A5 FC A6 FB 8D 55 03 8E 54 03 20 38 A9 A5 FC 20
$AAB0   B4 AA A5 FB AA 20 38 A9 8A 29 7F C9 20 08 B0 0A
$AAC0   A9 12 20 D2 FF 8A 18 69 40 AA 8A 20 D2 FF C9 22
$AAD0   F0 04 C9 62 D0 06 20 E2 AA 20 E5 AA 28 B0 C0 A9
$AAE0   92 2C A9 14 2C A9 22 4C D2 FF 20 38 A9 A6 FB A5
$AAF0   FC 4C CD DD 20 05 AB B0 41 20 38 A9 20 F8 A7 20

MONITEUR 6502       PAGE AB  CHECKSUM= $824D ( 33357)

$AB00   7D AA 4C 68 A0 A2 04 A9 00 85 FC 20 C2 AB 20 2B
$AB10   AB 85 FB 20 22 AB 20 3D AB CA D0 F7 08 20 38 A9
$AB20   28 60 20 A4 A9 F0 0F C9 20 F0 0B C9 30 90 0B C9
$AB30   3A B0 07 29 0F 60 68 68 18 60 4C 60 A0 85 FE A5
$AB40   FC 48 A5 FB 48 06 FB 26 FC 06 FB 26 FC 68 65 FB
$AB50   85 FB 68 65 FC 85 FC 06 FB 26 FC A5 FE 65 FB 85
$AB60   FB A9 00 65 FC 85 FC 60 20 C2 AB 8D 55 03 48 48
$AB70   20 38 A9 20 38 A9 68 20 FF A7 20 38 A9 68 AA A9
$AB80   00 20 F1 AA 20 38 A9 20 86 AA 4C 68 A0 20 9F AB
$AB90   20 38 A9 20 F8 A7 20 EA AA 20 A0 AA 4C 68 A0 A2
$ABA0   0F A9 00 85 FB 85 FC 20 C2 AB 20 2B AB 20 BC AB
$ABB0   20 22 AB 20 BC AB CA D0 F7 4C 38 A9 4A 26 FB 26
$ABC0   FC 60 20 8C A8 C9 20 F0 F9 60 20 54 A8 8D 88 02
$ABD0   A6 FB A4 FC 20 8A FE A6 FD A4 FE 20 7B FE 20 18
$ABE0   E5 20 A4 E3 4C 68 A0 20 F0 A7 4C DB A7 20 E7 AB
$ABF0   18 A5 FB 65 FD 85 FB A5 FC 65 FE 85 FC 4C 0D AC

MONITEUR 6502       PAGE AC  CHECKSUM= $79D6 ( 31190)

$AC00   20 E7 AB 20 F0 A0 84 FC AD 53 03 85 FB 20 38 A9
$AC10   20 F8 A7 4C 68 A0 A9 F0 2C A9 00 8D 0B 90 4C 65
$AC20   A0 78 20 52 FD 58 A9 3C 85 B2 AE 42 03 9A A5 73
$AC30   C9 E6 F0 95 6C 00 C0 20 E7 AB 20 21 A8 20 38 A9
$AC40   A0 00 8C 54 03 8C 55 03 20 F0 A0 90 1B AC 56 03
$AC50   D0 16 18 B1 FB 6D 54 03 8D 54 03 98 6D 55 03 8D
$AC60   55 03 20 1F A9 4C 48 AC AD 55 03 20 FF A7 AD 54
$AC70   03 20 FF A7 4C 68 A0 AD 64 03 D0 04 A5 C6 D0 03
$AC80   4C 56 FF AD 77 02 C9 11 D0 7D A5 D6 C9 16 D0 F0
$AC90   A5 D1 85 FD A5 D2 85 FE A9 17 8D 5E 03 A0 01 20
$ACA0   51 AE C9 3A F0 1A C9 2C F0 16 C9 24 F0 12 CE 5E
$ACB0   03 F0 CD 38 A5 FD E9 16 85 FD B0 E1 C6 FE D0 DD
$ACC0   8D 49 03 20 0A AE B0 B8 AD 49 03 C9 3A D0 11 18
$ACD0   A5 FB 69 08 85 FB 90 02 E6 FC 20 C8 A3 4C F4 AC
$ACE0   C9 24 F0 1A 20 C9 AD 20 6F A2 A9 00 8D 4E 03 A0
$ACF0   2C 20 13 A2 A9 00 85 C6 4C 0E A2 4C 56 FF 20 1F
```

```
MONITEUR 6502       PAGE AD  CHECKSUM= $8C52 ( 35922)

  $AD00   A9 20 6D AA 4C F4 AC C9 91 D0 F0 A5 D6 D0 EC A5
  $AD10   D1 85 FD A5 D2 85 FE A9 17 8D 5E 03 A0 01 20 51
  $AD20   AE C9 3A F0 1A C9 2C F0 16 C9 24 F0 12 CE 5E 03
  $AD30   F0 15 18 A5 FD 69 16 85 FD 90 E1 E6 FE D0 DD 8D
  $AD40   49 03 20 0A AE 90 03 4C 56 FF AD 49 03 C9 3A F0
  $AD50   06 C9 24 F0 1D D0 27 20 D0 AD 38 A5 FB E9 08 85
  $AD60   FB B0 02 C6 FC 20 CB A3 A9 00 85 C6 20 05 AE 4C
  $AD70   70 A0 20 D0 AD 20 B2 A0 20 70 AA 4C 68 AD 20 D0
  $AD80   AD A5 FB A6 FC 85 FD 86 FE A9 10 8D 5E 03 38 A5
  $AD90   FD ED 5E 03 85 FB A5 FE E9 00 85 FC 20 C9 AD 20
  $ADA0   6F A2 20 F0 A0 F0 07 B0 F3 CE 5E 03 D0 E0 EE 4D
  $ADB0   03 AD 4D 03 20 AB A3 A2 00 A1 FB 8E 4E 03 A9 2C
  $ADC0   20 33 A9 20 16 A2 4C 68 AD A2 00 A1 FB 4C 87 A2
  $ADD0   A6 D2 20 D7 AD A6 F4 E8 86 AD 86 FE A2 00 86 AC
  $ADE0   A9 2C 85 FD A0 CE E8 88 B1 AC 91 FD 98 D0 F8 C6
  $ADF0   AD C6 FE CA 10 F1 A9 20 A6 D2 86 FE 84 FD A0 2B

MONITEUR 6502       PAGE AE  CHECKSUM= $5858 ( 22616)

  $AE00   91 FD 88 10 FB A9 13 4C D2 FF C0 16 D0 02 38 60
  $AE10   20 51 AE C9 20 F0 F3 88 20 3A AE AA 20 3A AE 85
  $AE20   FB 86 FC A9 FF 8D 64 03 85 CC A5 CF F0 0A A5 CE
  $AE30   A4 D3 91 D1 A9 00 85 CF 18 60 20 51 AE 20 81 A8
  $AE40   0A 0A 0A 0A 8D 59 03 20 51 AE 20 81 A8 0D 59 03
  $AE50   60 B1 FD C8 29 7F C9 20 B0 02 09 40 60 BD 98 AD
  $AE60   20 D2 FF E8 D0 F7 60 00 00 00 00 00 00 00 0D 4D
  $AE70   4F 4E 49 54 45 55 52 20 36 35 30 32 20 50 4F 55
  $AE80   52 20 20 20 20 20 56 49 43 20 32 30 20 20 52 4F
  $AE90   4D 20 24 41 30 30 30 20 40 02 45 03 D0 08 40 09
  $AEA0   30 22 45 33 D0 08 40 09 40 02 45 33 D0 08 40 09
  $AEB0   40 02 45 B3 D0 08 40 09 00 22 44 33 D0 8C 44 00
  $AEC0   11 22 44 33 D0 8C 44 9A 10 22 44 33 D0 08 40 09
  $AED0   10 22 44 33 D0 08 40 09 62 13 78 A9 00 21 81 82
  $AEE0   00 00 59 4D 91 92 86 4A 85 9D 2C 29 2C 23 28 24
  $AEF0   59 00 58 24 24 00 1C 8A 1C 23 5D 8B 1B A1 9D 8A

MONITEUR 6502       PAGE AF  CHECKSUM= $647C ( 25724)

  $AF00   1D 23 9D 8B 1D A1 00 29 19 AE 69 A8 19 23 24 53
  $AF10   1B 23 24 53 19 A1 00 1A 5B 5B A5 69 24 24 AE AE
  $AF20   A8 AD 29 00 7C 00 15 9C 6D 9C A5 69 29 53 84 13
  $AF30   34 11 A5 69 23 A0 D8 62 5A 48 26 62 94 88 54 44
  $AF40   C8 54 68 44 E8 94 00 B4 08 84 74 B4 28 6E 74 F4
  $AF50   CC 4A 72 F2 A4 8A 00 AA A2 A2 74 74 74 72 44 68
  $AF60   B2 32 B2 00 22 00 1A 1A 26 26 72 72 88 C8 C4 CA
  $AF70   26 48 44 44 A2 C8 0D 20 20 20 20 50 43 20 20 49
  $AF80   52 51 20 20 53 52 20 41 43 20 58 52 20 59 52 20
  $AF90   53 50 41 42 43 44 46 47 48 4C 4D 4E 51 52 28 54
  $AFA0   57 58 2C 3A 3B 24 23 22 2B 2D 4F 49 4A 25 26 45
  $AFB0   56 29 3D 5C FF AA A9 9F A8 3D A4 1F A7 02 A1 F9
```

```
$AFC0   A1 87 A1 A4 A6 A0 A1 AA A9 B0 A3 3C A7 A8 A6 40
$AFD0   A9 16 AC 06 A1 B8 A6 2A AC 0C A3 15 A4 79 A9 64
$AFE0   AA F4 AA 68 AB ED AB 00 AC 35 AA CA AB 2C AA 8D
$AFF0   AB 37 AC 21 AC AA A9 19 AC 40 A3 40 A3 40 A3 00
```

Chargeur hexa.

```
10 REM ------------------------------------CHARGEUR HEXA   B.C.M.
20 PRINT"■POUR ARRETER UNE      FONCTION ,ENFONCER F1 "
30 INPUT"ADRESSE ";AD$:GOSUB3000
90 GOSUB1000:PRINTAD$" ";
100 OC=PEEK(AD):GOSUB2000:PRINTOC$"-";
110 GETA$:IFA$=""GOTO110
120 IFA$=" "THENPRINTOC$:GOTO220
130 IFA$=CHR$(133)THENRUN
140 IF(A$<"0")OR(A$>"F")GOTO110
150 IF(A$<"A")AND(A$>"9")GOTO110
160 PRINTA$;:OC$=A$
170 GETA$:IFA$=""GOTO170
180 IFA$=CHR$(133)THENRUN
190 IF(A$<"0")OR(A$>"F")GOTO170
200 IF(A$<"A")AND(A$>"9")GOTO170
210 PRINTA$:OC$=OC$+A$:GOSUB4000:POKEAD,OC
220 AD=AD+1:IFAD/8=INT(AD/8)THENPRINT
230 GOTO90
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

READY.
```

Verimon.

```
5000 REM ----------------VERIMON  :  VERIFICATION DU MONITEUR 6502
5001 :
5002 REM     ACCES PAR  :  RUN 5000
5003 :
5010 FORB=40960TO44800STEP256
5020 CS=0:FORJ=0TO255:CS=CS+PEEK(B+J):NEXTJ
5030 AD=B:GOSUB1000:PRINT"BLOC "AD$" = "CS
5040 NEXTB:END
```

Memory save.

```
6000 REM ----------------------SAUVE LE MONITEUR 6502 SUR CASSETTE
6001 :
6002 REM     ACCES PAR  :  RUN 6000
6003 :
6005 PRINT"BASCULER LA MEMOIRE   $A000 EN $2000        ET POUSSER RETURN"
6006 GETA$:IFA$=""GOTO6006
6010 S1=8192:S2=S1+4096:REM AD. DE DEBUT ET FIN BLOC
6020 A1=INT(S1/256):A2=S1-256*A1
6030 B1=INT(S2/256):B2=S2-256*B1
6040 POKE251,A2:POKE252,A1
6050 FORI=0TO2:POKEI,ASC(MID$("MON",I+1,1)):NEXTI
6060 POKE780,1:POKE781,8:POKE782,1:    SYS65466:REM SETLFS
6070 POKE780,3:POKE781,0:POKE782,0:    SYS65469:REM SETNAM
6080 POKE780,251:POKE781,B2:POKE782,B1:SYS65496:REM SAVE
6090 PRINT"BASCULER LA MEMOIRE   $2000 EN $A000        ET POUSSER RETURN"
6100 GETA$:IFA$=""GOTO6100
6110 END
6120 :
6130 MON SE RECHARGE AVEC LA MEMOIRE EN $2000
6140 PAR : LOAD"MON",8,1  POUR DISQUE.
6150 POUR CASSETTE, METTRE EN 6060 POKE781,1 AU LIEU DE 8
6160 SE RECHARGE DE LA CASSETTE PAR  :  LOAD"MON"
```

ROM autobasic.

```
0000                        .SYM            ;        ROM AUTOBASIC
0000                    ;
0000                    PRTMSG = $CB1E      ;IMPRESSION MESSAGE
0000                    SWITCH = $9111      ;INTERRUPTEUR CASSETTE
0000                    PROVIS = $63        ;POINTEUR PROVISOIRE ACCU FLP 1
0000                    KBDMAX = $0289      ;TAILLE MAXI TAMPON CLAVIER
0000                    KBDPTR = $C6        ;INDEX DANS TAMPON CLAVIER
0000                    KBDBUF = $0277      ;TAMPON CLAVIER
0000                    READY  = $C474      ;INTERPRETEUR BASIC
0000                    WARM   = $FEC7      ;REDEMARRAGE A CHAUD BASIC
0000                    CR     = $0D        ;CARACTERE A LA LIGNE
0000                           * = $A000    ;*****************************
A000  09 A0                    .WORD START  ;ADRESSE DE DEMARRAGE A FROID
A002  C7 FE                    .WORD WARM
A004  41 30                    .WORD $3041  ;'A0'
A006  C3 C2                    .WORD $C2C3  ;'CB'
A008  CD                       .BYTE $CD    ;'M'
A009  20 8D FD        START    JSR $FD8D     ;INITIALISE PAGE 0
```

```
A00C  20 52 FD          JSR $FD52       ;INITIALISE PAGE 3
A00F  20 F9 FD          JSR $FDF9       ;INITIALISE LES V.I.A.
A012  20 18 E5          JSR $E518       ;INITIALISE ECRAN
A015  58                CLI
A016  20 5B E4          JSR $E45B       :INITIALISE PAGE 3
A019  20 A4 E3          JSR $E3A4       ;INITIALISE CHARGOT
A01C  20 04 E4          JSR $E404       ; TESTE MEMOIRE
A01F  A2 FB             LDX £$FB
A021  9A                TXS             ;INITIALISE LA PILE
A022  A9 62             LDA £<SIGON     ;MESSAGE INDIQUANT LA PRESENCE
A024  A0 A0             LDY £>SIGON     ;DE LA ROM AUTOBASIC
A026  20 1E CB          JSR PRTMSG      :AFFICHE LE MESSAGE
A029  AD 11 91          LDA SWITCH      ;INTERRUPTEUR CASSETTE
A02C  29 40             AND £$40        ; ENFONCE  ?
A02E  F0 07             BEQ CASS
A030  A0 72             LDY £<MES0      ;SI NON, METTRE
A032  A2 A0             LDX £>MES0
A034  4C 3B A0          JMP INKBD       ;MESDSK DANS TAMPON CLAVIER
A037  A0 81       CASS  LDY £<MES1       ;SI OUI, METTRE
A039  A2 A0             LDX £>MES1      ;MESCAS DANS TAMPON CLAVIER
A03B  84 63       INKBD STY PROVIS       ;POINTEUR VERS MESSAGE
A03D  86 64             STX PROVIS+1    ;A METTRE DANS TAMPON
A03F  A2 00             LDX £$00        ;BOUCLE DE DELAI
A041  A0 00       DELAY1 LDY £$00        ;POUR INITIALISATION DES
A043  A9 02       DELAY2 LDA £$02        ;6522 ET 6551
A045  38          DELAY3 SEC
A046  E9 01             SBC £$01
A048  D0 FB             BNE DELAY3
A04A  88                DEY
A04B  D0 F6             BNE DELAY2
A04D  CA                DEX
A04E  D0 F1             BNE DELAY1
A050  A0 0F             LDY £$0F        ;MESSAGE DE 14 CARACTERES(+1)
A052  8C 89 02          STY KBDMAX      ;LONGUEUR TAMPON CLAVIER
A055  84 C6             STY KBDPTR      ;NOMBRE DE CAR. DANS TAMPON CLAVIER
A057  B1 63       LOOP  LDA (PROVIS),Y   ;PRENDS UN OCTET DU MESSAGE
A059  99 77 02          STA KBDBUF,Y    ;LE MET DANS LE TAMPON CLAVIER
A05C  88                DEY             ;DECOMPTE DES CARACTERES
A05D  D0 F8             BNE LOOP        ;JUSQU'AU DERNIER
A05F  4C 74 C4          JMP READY       ;PASSE A L'INTERPRETEUR BASIC
A062  0D          SIGON .BYTE CR,'B','C','M',' ','A','U','T','O'
A063  42
A064  43
A065  4D
A066  20
A067  41
A068  55
A069  54
A06A  4F
A06B  42                .BYTE 'B','A','S','I','C',CR,CR,$00
A06C  41
A06D  53
A06E  49
A06F  43
A070  0D
```

```
A071  0D
A072  00
A073  4C          MESDSK .BYTE $4C,$CF,'"','A','U','T','O'
A074  CF
A075  22
A076  41
A077  55
A078  54
A079  4F
A07A              MESO   = MESDSK-1
A07A  22                 .BYTE '"',$2C,'8',CR,$52,$D5,CR,$20
A07B  2C
A07C  38
A07D  0D
A07E  52
A07F  D5
A080  0D
A081  20
A082  4C          MESCAS .BYTE $4C,$CF,CR,$52,$D5,CR,$20
A083  CF
A084  0D
A085  52
A086  D5
A087  0D
A088  20
A089              MES1   = MESCAS-1
A089  20 20              .WORD $2020
A08B  20 20              .WORD $2020
A08D  20 20              .WORD $2020
A08F  20 20              .WORD $2020
A091                     .END
```

Quicksort.

```
1 REM  QUICKSORT
2 N=6
3 DIM U(16),V(16),A$(N)
4 FORI=1TO6:READA$(I):NEXT
5 DATA XAVIER,JEROME,ARTHUR,PAUL,EMILIE,TRISTAN
6 GOSUB1000:REM TRI
7 FORI=1TO6:PRINTA$(I):NEXT
8 END
1000 REM TRI CROISSANT
1100 W=1:U(1)=1:V(1)=N
1200 U1=U(W):V1=V(W):W=W-1
1300 X=U1:Y=V1:X$=A$(INT((U1+V1)/2))
1400 IFX$<A$(Y)THENY=Y-1:GOTO1400
1500 IFX$>A$(X)THENX=X+1:GOTO1500
1600 IFX>YGOTO1800
1700 W$=A$(Y):A$(Y)=A$(X):A$(X)=W$:X=X+1:Y=Y-1
1800 IFX<=YGOTO1400
1900 IFX>=V1GOTO2100
2000 W=W+1:U(W)=X:V(W)=V1
2100 V1=Y:IFU1<V1GOTO1300
```

```
2200 IFW<>0GOTO1200
2300 RETURN

READY.
```

DE NEW.

```
DE-NEW : RECUPERE UN PROGRAMME BASIC
         APRES NEW OU RESET (POUSSOIR)
POKE46,29:CLR
FORI=4102 TO 4189:IF PEEK(I-1)<>0 THEN NEXT
J=INT(I/256):POKE 4098,J:POKE 4097,I-J*256
LIST              (MAIS PAS ENCORE RUN !)
FORI=4104 TO 29*256:IF PEEK(I-1)+PEEK(I-2)+PEEK(I-3)<>0 THEN NEXT
J=INT(I/256):POKE 888,J:POKE 887,I-J*256
POKE 46,PEEK(888):POKE 45,PEEK(887)
RUN               (!)


READY.
```

Imprimante parallèle sur port utilisateur.

a.En basic.

```
 1 REM IMPRIMANTE PARALLELE SUR PORT UTILISATEUR
10 GOSUB 1000
11 :
20 INPUTA$:A$=A$+CHR$(13)+CHR$(10)
30 FORI=1 TO LEN(A$):C=ASC(MID$(A$,I,1)):GOSUB2000:NEXT
90 GOTO20
98 :
99 :
1000 REM*************************INIT
1001 V=37136:POKEV+2,255
1002 RETURN
1999 :
2000 REM***********SEND CHAR  CHAR = C
2010 POKE V,C AND 127
2020 POKE V+12,PEEK(V+12)OR 224
2030 POKE V+12,PEEK(V+12)AND 31 OR 192
2040 AC=PEEK(V+13)AND 16:IF AC<>16 GOTO 2040
2999 RETURN

READY.
```

b.En langage machine.

```
IMPRIMANTE PARALLELE SUR PORT UTILISATEUR

INITIALISATION PAR OPEN 2,2:SYS900
ACCES PAR PRINT£2,....
   OU PAR CMD2:LIST
ATTENTION : SYS900 EST APRES OPEN2,2 CAR OPEN MODIFIE LA VALEUR DDRB EN $9112
```

```
0011  0000                    LF      = $0A           ; LINE FEED  (AVANCE PAPIER)
0012  0000                    CR      = $0D           ; CARRIAGE RETURN (RETOUR CHARIOT
)
0013  0000                    STATUS  = $90           ; OCTET ST-STATUS
0014  0000                    PERI    = $9A           ; NUMERO DU PERIPHERIQUE DE SORTI
E
0015  0000                    VECWRT  = $0326         ; VECTEUR DE SORTIE D'UN CARACTER
E
0016  0000                    PORTB   = $9110         ; PORT UTILISATEUR
0017  0000                    DDRB    = $9112         ; DIRECTION DU PORT
0018  0000                    PCR     = $911C         ; CONTROLE DE CB2
0019  0000                    IFR     = $911D         ; LECTURE DE CB1
0020  0000                    WRT     = $F27A         ; ROUTINE NORMALE DE SORTIE D'UN
CARACTERE
0021  0000                    ;
0022  0000                    ;*****************************************************
0023  0000                    ;
0024  0000                                            ;  MODIFIABLES A VOLONTE
0025  0000                    ;
0026  0000                    DEBUT   = $0384         ; (SYS 900)
0027  0000                    CANAL   = 02            ; NUMERO DU CANAL UTILISE : REMPL
ACE L' RS232
0028  0000                    ;
0029  0000                    ;*****************************************************
0030  0000                    ;
0031  0000                            * = DEBUT
0032  0384                    ;
0033  0384  A9 9F             INIT    LDA #<PRINT
0034  0386  8D 26 03                  STA VECWRT
0035  0389  A9 03                     LDA #>PRINT
0036  038B  8D 27 03                  STA VECWRT+1    ;  MODIFIE LE VECTEUR DE SORTIE
0037  038E  A9 FF                     LDA #$FF
0038  0390  8D 12 91                  STA DDRB        ;  PORT UTILISATEUR EN SORTIE
0039  0393  A9 00                     LDA #$00
0040  0395  8D 10 91                  STA PORTB       ;  REMET LES SORTIES A ZERO
0041  0398  20 D2 03                  JSR CB2         ;  IMPULSION STROBE
0042  039B  AD 1D 91                  LDA IFR         ;  EFFACE LE DRAPEAU IFR-4
0043  039E  60                       RTS
0044  039F                    ;
0045  039F                    ;*****************************************************
0046  039F                    ;
0047  039F  48                PRINT   PHA             ;  SAUVE LE CARACTERE
0048  03A0  A5 9A                     LDA PERI        ;  EST-CE
0049  03A2  C9 02                     CMP #CANAL      ;  LE BON CANAL
0050  03A4  D0 17                     BNE FIN         ;  SI NON, FINI
0051  03A6  68                       PLA             ;  RECUPERE LE CARACTERE
0052  03A7  48                       PHA
0053  03A8  C9 0D                     CMP #CR         ;  EST-CE RETURN
0054  03AA  D0 08                     BNE SUITE       ;  SI NON, L'ENVOYER
0055  03AC  8D 10 91                  STA PORTB       ;  SI OUI, L'ENVOYER AUSSI




0056  03AF  20 C1 03                  JSR DIALOG      ;  ATTEND L'IMPRIMANTE
0057  03B2  A9 0A                     LDA #LF         ;  PUIS ENVOIE AUSSI UN LINE FEEI
0058  03B4  8D 10 91           SUITE   STA PORTB       ;  ENVOIE LE CARACTERE
0059  03B7  20 C1 03                  JSR DIALOG      ;  ATTEND L'IMPRIMANTE
0060  03BA  68                       PLA             ;RECUPERE LE CARACTERE
0061  03BB  18                       CLC             ;SI O.K.
0062  03BC  60                       RTS             ;  ET RETOUR.....
0063  03BD  68                 FIN   PLA             ;  SI PAS LE BON CANAL, RETOUR
0064  03BE  4C 7A F2                  JMP WRT         ;  A LA ROUTINE DE SORTIE NORMALE
0065  03C1                    ;
0066  03C1  20 D2 03           DIALOG  JSR CB2         ;  IMPULSION STROBE SUR CB2
0067  03C4  AD 1D 91           LOOP    LDA IFR         ;  ATTEND REPONSE
0068  03C7  29 10                     AND #$10        ;  SUR CB1
0069  03C9  C9 10                     CMP #$10        ;  IFR BIT4
0070  03CB  D0 F7                     BNE LOOP
```

```
0071  03CD  A9 00           LDA #$00
0072  03CF  85 90           STA STATUS      ; OCTET ST=STATUS A ZERO
0073  03D1  60              RTS
0074  03D2                 ;
0075  03D2  AD 1C 91  CB2   LDA PCR:LIT CB2
0076  03D5  09 E0           ORA #$E0
0077  03D7  8D 1C 91        STA PCR         ; MET CB2 A ZERO
0078  03DA  29 1F           AND #$1F
0079  03DC  09 C0           ORA #$C0
0080  03DE  8D 1C 91        STA PCR         ; REMET CB2 A UN
0081  03E1  60              RTS
0082  03E2                 ;***************************************************
0083  03E2                 .END
```

ERRORS = 0000

SYMBOL TABLE

SYMBOL	VALUE						
CANAL	0002	CB2	03D2	CR	000D	DDRB	9112
DEBUT	0384	DIALOG	03C1	FIN	03BD	IFR	911D
INIT	0384	LF	000A	LOOP	03C4	PCR	911C
PERI	009A	PORTB	9110	PRINT	039F	STATUS	0090
SUITE	03B4	VECWRT	0326	WRT	F27A		

c.Chargeur Basic pour le programme ci-dessus.

```
5 REM CENTRONICS       COPYRIGHT BCM
10 :
20 :
30 REM INITIALISER PAR SYS 900
40 REM ACCES PAR : OPEN 6,6:PRINT£6,"...
50 :
60 FORI=900TO993:READA:POKEI,A:NEXT
70 OPEN2,2:SYS900
80 PRINT£2," PROGRAMME DE GESTION IMPRIMANTE PARALLELE":PRINT£2
90 CMD2:LIST
99 END
900 DATA169,159,141,38,3,169,3,141
908 DATA39,3,169,255,141,18,145,169
916 DATA0,141,16,145,32,210,3,173
924 DATA29,145,96,72,165,154,201,2
932 DATA208,23,104,72,201,13,208,8
940 DATA141,16,145,32,193,3,169,10
```

```
948 DATA141,16,145,32,193,3,104,24
956 DATA96,104,76,122,242,32,210,3
964 DATA173,29,145,41,16,201,16,208
972 DATA247,169,0,133,144,96,173,28
980 DATA145,9,224,141,28,145,41,31
988 DATA9,192,141,28,145,96

READY.
```

Barrettes verticales.

```
20 FORI=0TO7:READA:POKE7680+I*22,A:POKE38400+I*22,0 :   NEXTI
30 DATA101,84,71,66,93,72,89,103:REM BARRETTES VERTICALES
40 GOTO40

READY.
```

Barrettes horizontales.

```
20 FORI=0TO7:READA:POKE7680+I,A:POKE38400+I,0:NEXTI
30 DATA99,69,68,67,64,70,82,100:REM BARRETTES HORIZONTALES
40 GOTO40

READY.
```

Noircis à partir du bas et du haut.

```
20 FORI=0TO15:READA:POKE7680+I,A:POKE38400+I,0:NEXTI
30 DATA100,111,121,98,248,247,227,160:REM NOIRCIS A PARTIR DU BAS
31 DATA99,119,120,226,249,239,228,160:REM NOIRCIS A PARTIR DU HAUT
40 GOTO40

READY.
```

Noircis à partir de gauche et de droite.

```
20 FORI=0TO15:READA:POKE7680+I*22,A:POKE38400+I*22,0:NEXTI
30 DATA101,116,117,97,246,234,231,160:REM NOIRCIS A PARTIR DE GAUCHE
31 DATA103,106,118,225,245,244,229,160:REM NOIRCIS A PARTIR DE DROITE
40 GOTO40

READY.
```

CODES CARACTERES DU VIC-20

POKE	PRINT		POKE	PRINT		POKE	PRINT		POKE	PRINT	
32	32	$20	0	64	$40	96	160	$A0	64	192	$C0
33	33	$21	1	65	$41	97	161	$A1	65	193	$C1
34	34	$22	2	66	$42	98	162	$A2	66	194	$C2
35	35	$23	3	67	$43	99	163	$A3	67	195	$C3
36	36	$24	4	68	$44	100	164	$A4	68	196	$C4
37	37	$25	5	69	$45	101	165	$A5	69	197	$C5
38	38	$26	6	70	$46	102	166	$A6	70	198	$C6
39	39	$27	7	71	$47	103	167	$A7	71	199	$C7
40	40	$28	8	72	$48	104	168	$A8	72	200	$C8
41	41	$29	9	73	$49	105	169	$A9	73	201	$C9
42	42	$2A	10	74	$4A	106	170	$AA	74	202	$CA
43	43	$2B	11	75	$4B	107	171	$AB	75	203	$CB
44	44	$2C	12	76	$4C	108	172	$AC	76	204	$CC
45	45	$2D	13	77	$4D	109	173	$AD	77	205	$CD
46	46	$2E	14	78	$4E	110	174	$AE	78	206	$CE
47	47	$2F	15	79	$4F	111	175	$AF	79	207	$CF
48	48	$30	16	80	$50	112	176	$B0	80	208	$D0
49	49	$31	17	81	$51	113	177	$B1	81	209	$D1
50	50	$32	18	82	$52	114	178	$B2	82	210	$D2
51	51	$33	19	83	$53	115	179	$B3	83	211	$D3
52	52	$34	20	84	$54	116	180	$B4	84	212	$D4
53	53	$35	21	85	$55	117	181	$B5	85	213	$D5
54	54	$36	22	86	$56	118	182	$B6	86	214	$D6
55	55	$37	23	87	$57	119	183	$B7	87	215	$D7
56	56	$38	24	88	$58	120	184	$B8	88	216	$D8
57	57	$39	25	89	$59	121	185	$B9	89	217	$D9
58	58	$3A	26	90	$5A	122	186	$BA	90	218	$DA
59	59	$3B	27	91	$5B	123	187	$BB	91	219	$DB
60	60	$3C	28	92	$5C	124	188	$BC	92	220	$DC
61	61	$3D	29	93	$5D	125	189	$BD	93	221	$DD
62	62	$3E	30	94	$5E	126	190	$BE	94	222	$DE
63	63	$3F	31	95	$5F	127	191	$BF	95	223	$DF

LE LIVRE DU VIC

ANNEXE 2.

MATERIELS ET FOURNISSEURS.

(Liste non limitative, loin de là !)

EXT VIC-BUS 4	Carte-mère pour extensions : 4 cartes possibles. SEDERMI 28, rue VICQ d'AZIR 75010 PARIS FRANCE.
EXP CHASSIS	Alimentation + 7 emplacements d'extensions. ARFON USA, 111, Renadrive, Lafayette, La 70503, USA
EXP BOARD	Carte d'expérimentation pour développer vos propres extensions. VIC réf.£4844 ELCOMP 53 , Redrock lane, Powona, CA 91766, USA.
STOREBOARD	Extension 32K RAM + 4K ROM et connecteur d'extension. STACK computers LTD, 290,Derby road, Bootle, Liverpool zo, U.K.
RAMAX	Extension mémoire 27K + 2 connecteurs d'extensions. Sélection des adresses par interrupteurs. APROPOS TECHNOLOGY, 350 N.Lantana ave.,suite 821, Camarillo, CA 93010 USA.
CARD/?	Interface IEEE série-CENTRONICS parallèle CARDCO-LIGHTNER,812 S.LIGHTNER,WICHITA,KS 67218 USA
INTERPOD	Interface IEEE 488 et RS 232. Se branche sur l'IEEE série. OXFORD COMPUTERS SYSTEMS The Old Signal Box, Hensingtonroad, WOODSTOCK, U.K.
VIC-TO-IEEE	Interface IEEE 488 bon marché pour connecter les périphériques PET. OEM inc,2729 South U.S.£1,Ft Pierce,Florida,33450 USA
RS 232 + JOY	Interface RS 232 + second manche de commande. OEM inc.
VIDEOPAK	Interface vidéo 40/80 colonnes+extensions mémoire : DATA 20 CORPORATION 23011, Moulton Parkway,suite B10, Laguna HILLS, CA92653,USA.
COMBO	Interface video 40/80 colonnes avec ou sans mémoire 16K. QUANTUM DATA suite A,BOX 285, 14252 Culver Drive, IRVINE CA 92714 USA.

INTERFACE VIDEO 40/80 COLONNES	Interface video 40/80 colonnes OXFORD COMPUTERS SYSTEMS, The Old Signal Box, Hensingtonroad, Woodstock U.K.
PROMQUEEN	Programmeur d'EPROM. Simule et programme les mémoires de 4K : 2732A, 2732 et 2716. Avec logiciel en ROM. GLOUCESTER COMPUTER CO, 6,Brooks Road, Gloucester, MA 01930, USA.
TELEX CW	Cartouche de conversion RTTY pour recevoir, afficher et émettre des messages en code morse et Baudot transmis par ondes courtes (entrée écouteur radio) avec logiciel en ROM. COMPUTERWORLD, Hilvertsweg, 99 1214 JB Hilversum, Nederland (PAYS-BAS).
D/A CONVERTER	Cartouche sur port utilisateur 8 entrées analogiques 8 sorties analogiques. MICRO WORLD ELECTRONICS, 6340 W.Mississipi ave.,Lakewood, Colorado 80226, USA.
GAMES MULTIPLEXER	Dédouble le connecteur de jeu par une commande POKE STACK. (voir plus haut).
GAMELOADER	Cartouche de conversion permettant d'utiliser les cartouches ATARI sur le VIC-20. PROTECTO, P.O.BOX 550, BARRINGTON. ILLINOIS 60010 USA.
ESF-20/64	Stringy floppy, microcassette ultra-rapide 7200 bauds/64K. EXATRON, 181, Commercial St, Sunnyvade CA94086, USA.
BREEZE MACHINE	Ventilateur de refroidissement spécial VIC . DIGITAL INTERFACES SYSTEMS, PO BOX 8715, Portland, Oregon 97207 USA.
DUSTCOVER	Housse de protection pour VIC et CBM64 . INTERESTING SOFTWARE, 21101,S.Harvard Blvd,Torrance CA90501 USA.
SPOKESMAN	Synthétiseur vocal pour VIC-20 et COMMODORE 64. MOOSEWAREinc. PO BOX 17868,Irvine, California 92713 USA.
CHATTERBOX	Synthétiseur vocal. CURRAH COMPUTERS, Hartlepool, Cleveland.
MYNAH MODULE	Comme ci-dessus + logiciel sophistiqué en ROM. CURRAH.
TYPE & TALK	Synthétiseur vocal VOTRAX

ANNEXE 3.

LOGICIELS ET FOURNISSEURS.

(liste non limitative, loin de là !)

VICKIT 1 ET 2	Extensions BASIC en ROM de 2 et 4K. STARK COMPUTERS LTD 290, Derby Road, Bootle, Liverpool zo U.K.
ARROW	Cartouche ROM 4K contenant plusieurs nouveaux mots-clé, cassette plus rapide (6x), RENUM, FIND, MOVE, conversions hexa. DATACAP, rue du village 73, B-4545 FENEUR, BELGIQUE.
RABBIT	Cartouche ou cassette 12 commandes supplémentaires avec connecteur extension reproduit à l'arrière : LOAD et SAVE à 1200 bauds (6x plus vite) sur cassette. EASTERN HOUSE, 3239 Linda Drive, Winston Salem, NC 27109, USA.
WORDCRAFT 20	Traitement de texte en ROM avec 8K RAM. AUDIOGENIC, PO BOX 88, Reading, Berks, U.K.
VICWRITER	Traitement de textes, imprime sur VIC 1515 (pour 16K) PROCEP, 19 rue Mathurin Regnier, Paris, France.
QUICK BROWN FOX	Traitement de textes. Q.B.F., 548 Broadway, New York 10012 USA.
RAPIDWRITER +	Traitement de texte avec courrier personnalisé et télécommunications. RAPIDWRITER, 91 Long Hill Rd, Lewerett, MA 01054, USA.
VICFORTH	Cartouche de langage FORTH, contient 3K RAM. HUMAN ENGINEERING SOFTWARE, 71, Park Lane, Brisbane, CA94005 USA. (distribué en FRANCE par PROCEP.)
VIGIL	Interpréteur de langage VIGIL (spécial graphiques animés). ABACUS SOFTWARE, PO BOX 7211 Grand Rapids Michigan 49510, USA.
PIPER	Interpréteur de langage PIPER (musical). ABACUS.
TINY COMPILER	Compilateur BASIC (ne compile pas tout). ABACUS (voir VIGIL).

TINY PILOT	Interpréteur de langage PILOT. ABACUS (voir VIGIL).
TURTLE GRAPHICS	Graphiques basés sur le principe **LOGO** H.E.S. (voir VICFORT).
HESBAL	Assembleur 2 passes pour VIC 8K cassette ou disque. H.E.S. (voir VICFORTH).
EZASM	Cartouche ROM 4K : assembleur symbolique en 2 passes avec référence croisée DATACAP (voir ARROW).
HESCOUNT	Optimiseur de programmes BASIC. H.E.S. (voir VICFORTH).
HESPLOT	Sous-routines ultra-rapides de tracés graphiques . H.E.S. (voir VICFORTH).
VICSTAT	Cartouche : 15 commandes BASIC supplémentaires : diagrammes en barre, moyenne, écart-type, régression linéaire. PROCEP, 19 rue Mathurin Régnier Paris, France.
VICGRAF	Affichage graphique de fonctions, calcul d'intégrales définies. PROCEP.
VICFILE	Gestion de fichiers : adresses, inventaires, etc : 1000 fiches maximum par disque, impression sur imprimante VIC 1515 (pour 16K et disque). PROCEP.
SUP-EX DUMP	Recopie d'écran graphique SUPEREXPANDER sur imprimante. ABACUS , (voir VIGIL).
SIMPLICALC	Tableau de calcul électronique de 19*3 à 100*100 cases (cassette ou disque). PROCEP, (voir VICSTAT).
COPYCALC	3 programmes : comptabilité/tableau de calcul/écriture du courrier personnalisé. COMPUTER SERVICE CENTER, 1115, Third st. SAN RAFAEL,CA 94901, USA.
PRACTICALC	Tableau de calcul avec tris, 20 fonctions mathématiques. COMPUTER SOFTWARE ASSOCIATES, 50, Teed Drive, RANDOLPH,MA 02368, USA.
TERMINAL-40	Programme de terminal pour le VIC. Affiche 40 caractères par ligne en majuscule (matrice 3*6 points); nécessite 300 bauds RS 232 et extension 8K. MIDWEST MICRO ASSOCIATES, PO BOX 6148, K.C. MA64110 USA.

3000 PROGRAMMES	Pour PET et VIC sont disponibles dans la bibliothèque du club : TORONTO PET USERS GROUP (TPUG), 381 Lawrence Avenue West, TORONTO, ONTARIO, CANADA M5M 1B9. ou NATIONAL VIC-20 USER'S GROUP, PO BOX 34575, OMAHA, NE 68134, USA.
FOX 20	Revue pour le VIC publiée sur cassettes (1 par mois) avec au minimum 5 programmes par numéros (jeux, éducation, utilitaires) VIXEN, PO BOX 507, Deer Park, Texas 77536, USA.
VIC-PICS	19 images graphiques en mode haute résolution dynamique créées par caméra vidéo sur APPLE II et transférées sur VIC. Très beau graphiques ! MIDWEST MICRO ASSOCIATES (voir TERMINAL-40).
THE CUBE	Résolution et affichage du RUBIK'S CUBE. QUMAX, GRW laboratoris, PO BOX 17010, Rochester, NY 14617, USA.
AMOK	Graphiques, manche de commande. UNITED MICROWARE INDUSTRIES 3431 H, POMONA Blvd, POMONA, CA 91768, USA.
SNAKMAN	Graphiques, manche de commande. MICRODIGITAL, 752 John Glenn Blvd, Webster, NY 14580, USA.
KILLER	Jeu graphique chenilles contre araignées. WUNDERWARE, PO BOX 1287, JACKSONVILLE, OR 97530, USA.
MADPAINTER	Jeu graphique : peintres de route contre piétons. WUNDERWARE .
SNAKE	Jeu graphique : serpent contre souris, lapins etc. WUNDERWARE.
BOSS	Echecs (BAT SARGON II). D.E.S. 8315, Firestone Blvd, Downey, CA 90241,USA.
BONZO	Jeu vidéo classique (pour 8K) : échelles et tonneaux. DES.
HOPPER	Jeu vidéo classique : autoroute et grenouilles. DES.
VIKING	Jeu de stratégie (1 à 4 joueurs) exige 16K. PRIKLY-PEAR 9822,E.STELLA Road, Tucson, Arizona 85730 USA.
DR FLOYD	Dialogue psychoanalytique, intelligence artificielle (pour VIC 16 K). APROPOS TECHNOLOGY (voir RAMAX annexe 2).
CRUSH	Labyrinthe spatial, 60 chambres avec monstres, robots, etc

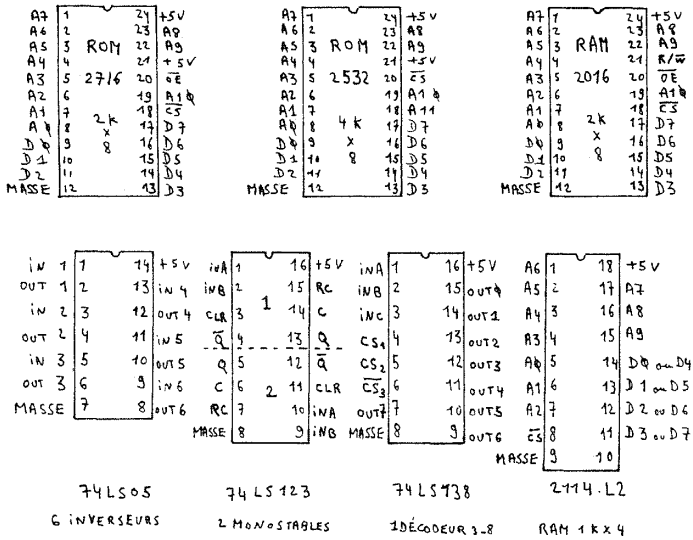
	(pour VIC 16K). EPYX A.S.I. 1043 Kiel Court, Sunnyvale, CA 94086, USA.
PINBALL	Jeu de flipper. Bons graphiques et sons. HES (voir VICFORTH).
BUG OFF	Graphiques, haute vitesse et résolution : détruisez les vilains "bugs". WESTERN NEWENGLAND SOFTWARE ASSOC., PO BOX 31, Wilbraham, MA01095 USA.
MAZEMAN	Graphiques rapides (manche de commande). RAR-TECH Box 761, Rochester, MI 48063, USA.
BACKGAMMON	2 versions 5K et 8K. RAR-TECH.
CHIMPCHASE	Chasse au gorille dans un labyrinthe (manche de commande). LITTLE WIZARD, 622, North Broadway, £301, Milwaukee, Wisconsin 53202, USA.
COSMIC CRUSADER	Défendez la base spatiale contre les forces du mal. LITTLE WIZARD.
REBEL DEFENDER	Mode multicolore, 2 joueurs, avec poignées de jeu et extension 8K. LITTLE WIZARD.
GRAVE ROBBERS	Jeu d'aventure graphique : explorez le tombeau perdu et affrontez ses périls. VICTORY SOFTWARE, 2027-A, S.J.RUSSEL CIRCLE, ELKINSPARK, PA 19117, USA.
TREASURE OF THE BATCAVE	Labyrinthe vu en perspective avec trésors, vampires, etc (manche de commande). VICTORY SOFTWARE.
ARCADIA	envahisseurs de l'espace. IMAGINE, Exchange Streets East Liverpool, Mersezside L2 3PN U.K.
WACKYWAITER	serveur de restaurant contre clients fous (haute résolution couleur, son). IMAGINE.
VICPANIC	combat contre dragon. BUG-BYTE SOFTWARE, IDO, ALBANY, Old Hall Street,Liverpool L3 9EP UK.
VIC-20 CHESS	échecs pour VIC 16K. BUG-BYTE
ORB	labyrinthe, monstres, trésors, etc. IMPACT Software, 70 Redford av. Edinburgh EH 13 OBW UK.
VICAID	moniteur LM + AUTO, NUMBER, HELP,etc...(circuit ROM 4K).

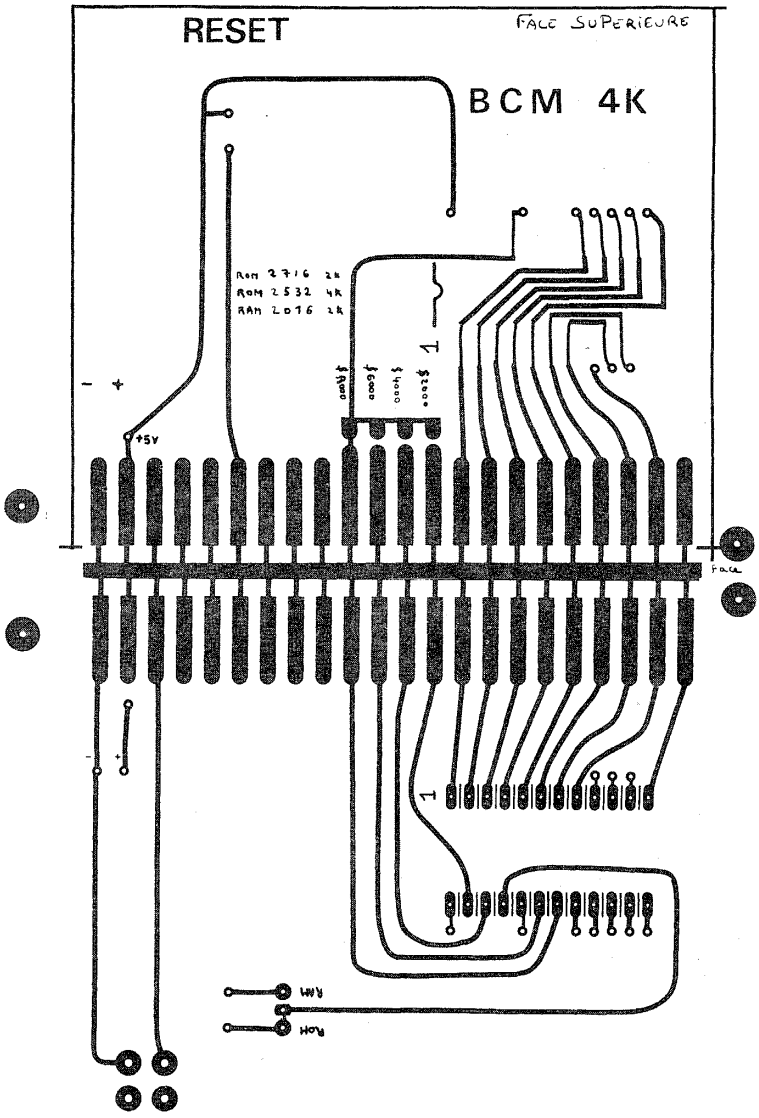
DAMS, GORES Road Kirby, near Liverpool L33 7UA UK.

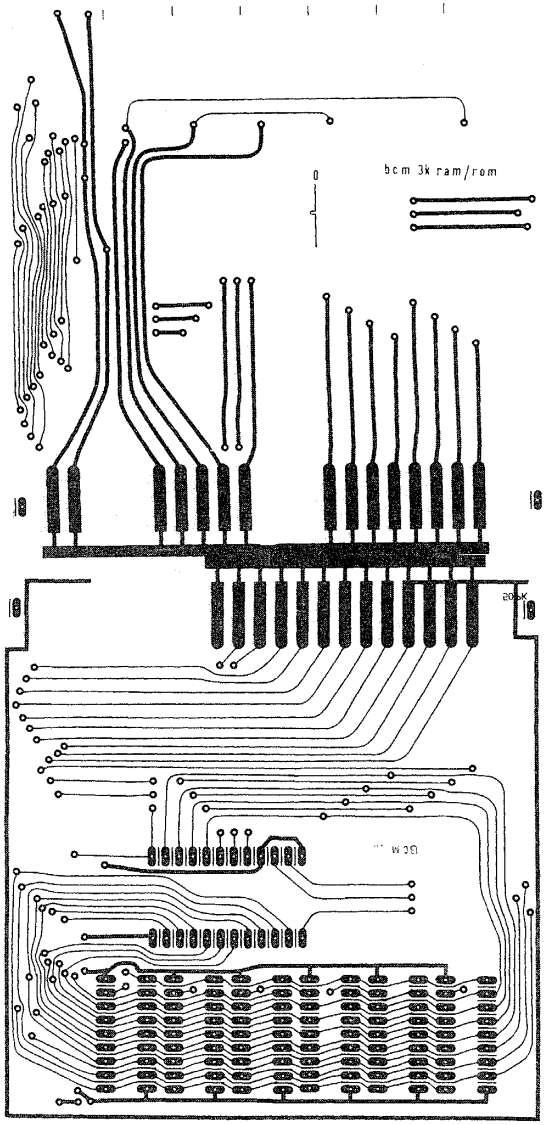
VICMON — moniteur LM en ROM 4K.
DAMS.

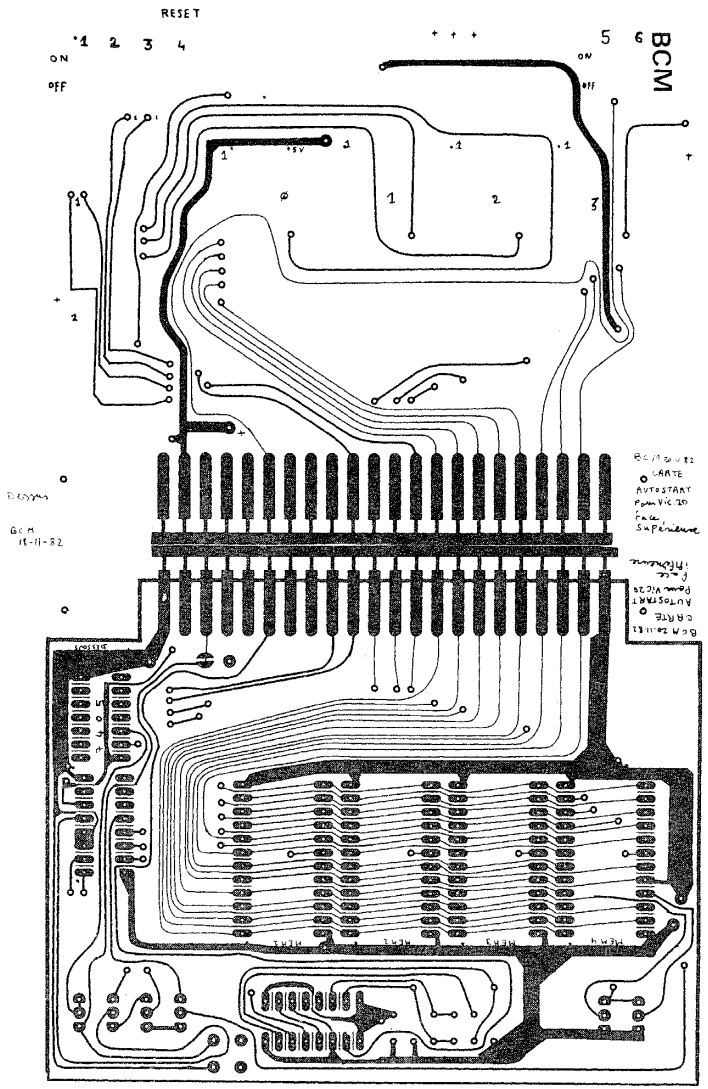
ANNEXE 4

CLICHES DES CARTES.




RESET
FACE SUPERIEURE
BCM 4K
Rom 2716 2K
Rom 2532 4K
RAM 2016 2K
$A000
$8000
$6000
$4000
+5V
FACE
RAM
ROM


bcm 3k ram/rom


RESET
ON
OFF
1 2 3 4
5 6
BCM
CARTE
AUTOSTART
Pour VIC 20
Face Supérieure
BCM
18-11-82

ANNEXE 5

BIBLIOGRAPHIE

Livres

La découverte du vic., (2 tomes), D.J.DAVID (**P.S.I.**, 41, rue Jacquard BP86, 77400 Lagny, France.

VIC revealed., N.HAMPSHIRE (**N.H.Publications**).

PET graphics N.HAMPSHIRE.

Library of PET subroutines N.Hamphire.

Understanding your VIC., (**COMPUTE**, voir ci-dessous).

Compute's firstbook of VIC. (**COMPUTE**).

Starting with BASIC on the VIC-20 (**COMPUTE**).

Input/output programming with your VIC, **ELCOMP** Publishing inc, 53, Redrock Lane, Pomona, CA 91766,USA.

VICstuff : liste de logiciels/matériels pour VIC. **MOOSEWARE** :voir SPOKESMAN annexe 3).

PAL/programmers Aids and Logs. : trucs et ficelles de programmation, feuilles pour préparer les dessins d'écran etc. **PM PRODUCTS**, 4455 Torrence Blvd, £177, Torrance, CA 90503, USA>

VIC 20 interfacing blue book : 20 circuits pour VIC : thermomètre, composeur téléphonique, synthétiseur vocal, etc ... **MICROSIGNAL** p.o. box 22, Millwood, NY 10546, USA.

The Dr Watson Book of Assembly language Programming.. par D.HOLMES, 21 Colin Drive, LONDON NW96ES, U.K.

Pet Machine Language Guide., **ABACUS** (voir annexe 3).

PET and IEEE488, E FISHER, **Osborne-MAC GRAW HILL**.

Mise en oeuvre du bus IEEE488, Bastide et Vellas, **EDITESTS**

Pet graphics ,**Total information Software**, P.O. box 921, Los Alamos, NM 87544, U.S.A

Revues..

La commode, 28 rue Vicq d'Azir 75010 PARIS.

Compute S.S.S PO BOX 11747, Philadelphia, PA 19101,USA.

Microcomputer printout, Benn Brothers, 25 New Street Square, London EC4A 3JA, U.K.

Your computer, Quadrant House, The Quadrant, Sutton, Surrey SM2 SAS,U.K.

VIC computing, radox, 39 North Road, London N7.9DP,U.K.

Micro 6502 : MICROINK, Chelinsford, MA 01824, USA.

FOX 20 : revue sur cassette (voir annexe 3) concerne la vidéo en général et le VIC en particulier (pour VIC de base).

C.P.U.C.N.., 360, Euston Road, London, NW1 3BL U.K

INDEX.

Accumulateur FLP 107, 168
Action 80
Adresse 5
Alarme 24
Aléatoire 109
Alimentation 150
AND 32
Attente 79
Autobasic 166, 216
Autoprint 204
Autostart 13
Baud 172
Blockdata 198
Branchements conditionnels 96
Branchements inconditionnels 95
Break flag 12
BRK 170
Bus 5
Calcul des pointeurs 34
Canal d'entrée 101, 130
Canal de sortie 101, 130
CARA 35
Caractères à barrettes 28, 222
Caractères blocs 4*4 points 29
Caractères en 8*n points 28, 222
Caractères graphiques 26
Caractères programmables 18, 33, 200
Carry flag 12
Carte mémoire 3, 7, 75, 167
Cassette 61
CA 1 54
CA 2 54
CB 1 55
CB 2 56
Centronics 148
Cercle 189
Chaînes de caractères 88
Chargeur hexa 160, 187
Chargot 82, 92
CHRIN 66, 130
CHROUT 66, 130
Clavier 58, 60
CLR 89, 92
CMD 105
Codes caractères 223
Cold start 114
Commandes BASIC 92
Commandes sans argument 92
Commande avec argument 93

Commandes moniteur 161
Connecteurs 49, 50, 61, 67, 136, 149
CONT 92
Couleur auxiliaire 23
Couleur du bord 21
Couleur du fond 21
Curseur 169
DATA 93, 94
Decimal flag 12
DEF 94
Denew 219
DIM 93
Double alpha 17, 195
Drapeau 12, 96
Duplex 72
Dynahires 41, 45, 201
Ecran 16
Ecrans multiples 17, 195
Editeur d'écran 79
Emission de caractères 73
END 92
Entrées analogiques 23, 144
Entrelacement 14
Eprom 160
Erreur RS232 75
Espace 80
Expression conditionnelle 111
Extensions logicielles 13
Fermeture du canal de communication 73
Fichier de données 36
Flag 12
Fonctions 107, 178
Explosion 189
Expressions 104
Fonctions FN 88
FOR-NEXT 89, 96
Garbage collection 108
Générateur de caractères 18, 171
Gestion des erreurs 74
GET IN 58
GOSUB 96
GOTO 95
Guillemets 80
Half-duplex 72
Haute résolution bicolore 37
Hauteur écran 16
Hires 6, 38, 39, 40, 200
Horloge 6
IEEE 64
IF 96
Imprimante parallèle 148, 219
inlistable 191
INPUT 94
Instructions du 6502 11

Interface IEEE série 64
Interpréteur BASIC 79, 82
Interpréteur de commande 79
Interupt flag 12
Interruption 56
Inversion 23
IRQ 47
Jeu d'instructions 11
Kernal 75, 79, 113
Largeur écran 15
LET 94
LIST 94
LOAD 106
LOGO 61
Lores 26, 27, 68, 198
LSB 85
Manche de commandes 67, 68, 204
Marge gauche 15
Marge supérieure 15
Medres 30, 31, 199
Memory save 160, 187, 216
Messages d'erreurs 178
Mini orgue 25, 196
Minuterie-compteur 1 51
Minuterie T 1 51
Minuterie 2 52
Modes d'adressage 9, 10
Modes différents 18
Modes graphiques 26
Modem 72
Moniteur 6502 13, 160, 187, 209
Mots-clé BASIC 80, 177
Multicolore (mode) 6, 23, 43
Multires 45, 202
MSB 85
NEW 92
NEXT 89, 96
NMI 47, 170
Notation flottante 84
Numéro de fichier 101
Numéro de périphérique 101, 129
ON 96
Opcode 9
OPEN 102, 129
OR 32
Overflow flag 13
Paddles 24
Page 0 167
Périphériques 58
PET-VIC 191
PHI 1 6
PHI 2 6
Photostyle 23, 147, 172
Pile 95
Poignées de jeu 77, 196

POKE 94
Polynôme 108
Ports parallèles 48
Port utilisateur 49
Position balayage écran 18
POT X, POT Y 23, 144
Préparation de la mémoire 33
Préparation de l'image 14
Préparation des couleurs 21
Prog DATA 82, 94, 206
Programmation des caractères 35
Programmation des fréquences sonores 25
Programmation du volume généré 25
Quicksort 192, 218
Ramassage d'ordures 108
RAM couleur 7, 17
READ 93
Réception de caractères 73
Registre A 8
Registre à décalage 53
Registre PC 8
Registres de contrôle 54
Registre SP 8, 95, 96
Registres X et Y 8
REM 93
Renum(ber) 191, 205
Répétition automatique 59
RESET 13, 137, 145, 190, 192
Résolution moyenne multi-colore 42
RESTORE 57, 59, 93, 173
RETURN 80, 96
RND 109
RS-232 52, 70, 75, 145, 170
RTS 83, 95
RUN 95
SAVE 106
SCN key 58
Shéma du manche 67
SHIFT-RETURN 80
Sign flag 13
Sommet de mémoire 87, 188
Sortie sonore 24
SRQ 55
STOP 58, 59, 92, 132
ST-STATUS 36, 66, 107, 128, 168
Superexpander 153
Superlist 82, 94, 205
Super tableaux 90, 208
Supervariables 89, 207
Sychro 6
SYS 86, 95
Tableau 89

Taille des caractères 17
Taille mémoire 190
Tampon d'entrée 80
Tampon clavier 80
Terminal 190, 204
Texte et graphiques 47
THEN 96
Thermomètre 196
TOKEN 81, 177
TOP 33, 34
Touche programmable 59
Tri 192, 218
Usage de la mémoire 87
Valeurs 92
Variable chaîne de caractères 88
Variable entière 88
Variable flottante 87
Vecteur 113, 170
Verimon 160
V.I.A. 47
V.I.C. 6, 172
VIC-PET 191
WAIT 96
Warm start 114
Watchdog 141
X-line 74, 75
Zéro flag 12
Zone d'impression 105
Zone texte 81
Zone tableaux 89
Zone variables 87, 89
3-fils 72
6502 A 5, 8
6561 14

TABLE DES MATIERES

INTRODUCTION 3

CHAPITRE 1 : LES ELEMENTS DU SYSTEME. 5

Les deux processeurs. 5
Le 6502 8
Le 6561 14

Les modes graphiques. 26
Les caractères programmables 33
La haute résolution bicolore. 37
La résolution moyenne multicolore. 42

Les deux VIA 47

Les périphériques. 58
Le clavier 58
Le lecteur de cassettes. 61
L'interface IEEE série. 64
Le manche de commande. 67
L'interface série RS-232. 69
Les poignées de jeu. 77

CHAPITRE 2 : LES PROGRAMMES INTERNES DU VIC. 79

L'interpréteur BASIC. 79
Le KERNAL. 113
Les routines du BASIC. 115
Les routines du KERNAL. 120

CHAPITRE 3 : LES EXTENSIONS MATERIELLES. 135

Le connecteur d'extension. 136
La carte ROM. 138
La carte 3K RAM/ROM. 139
La carte 8K RAM/ROM + Watchdog. 140
Les entrées analogiques du VIC. 144
Modification des cartouches existantes. 144
Poussoir de RESET. 145
Interface RS-232 145
Le photostyle. 147
Imprimante parallèle sur port utilisateur. 148
Alimentation ininterruptible. 150

CHAPITRE 4 : LES EXTENSIONS LOGICIELLES. 153

Superexpander. 153
Le moniteur 6502. 160
La ROM autobasic. 166

CHAPITRE 5 : CARTE DES ADRESSES MEMOIRE DU VIC-20. 167

CHAPITRE 6 : DIVERS. 187

Hexadécimal. 187
Pointeurs. 187
Sommet de mémoire. 188
Erreurs de l'interpréteur BASIC. 188
Erreur du lecteur VIC-1540 189
Explosion. 189
Cercles en haute résolution. 189
Si la mémoire est trop petite... 190
Le VIC comme terminal. 190
RESET. 190
Programme inlistable. 191
Renuméroter les lignes BASIC. 191
Changement de diapositives. 191
Tranferts PET-VIC et VIC-PET. 191
Tri. 192
Récupérer un programme après NEW ou RESET. 192

ANNEXE 1 : PROGRAMMES. 195

Ecrans multiples. 195
Double alpha. 195
Mini-orgue. 196
Poignées de jeu. 196
Thermomètre. 196
Lores. 198
Block data. 198
Medres 199
Caractères programmables. 200
Hires. 200
Dynahires. 201
Multires. 202
Touches programmables. 202
Manche de commande. 204
Terminal RS-232. 204
Autoprint. 204
Renumber. 205
Superlist. 205
Progdata. 206
Supervariables. 207
Supertableaux. 208
Moniteur 6502. 209
Chargeur hexa. 215

Verimon, 216
Memory save. 216
ROM autobasic. 216
Quicksort. 218
De new. 219
Imprimante parallèle sur port utilisateur. 219
a. En BASIC. 219
b. En langage machine. 219
c. Chargeur BASIC pour le programme ci-dessus. 221
Barrettes verticales. 222
Barrettes horizontales. 222
Noircis à partir du bas et du haut. 222
Noircis à partir de gauche et de droite. 222
Codes caractères du VIC-20.

ANNEXE 2 : MATERIELS ET FOURNISSEURS. 225

ANNEXE 3 : LOGICIELS ET FOURNISSEURS. 227

ANNEXE 4 : CLICHES DES CARTES. 232

ANNEXE 5 : BIBLIOGRAPHIE. 236

INDEX. 239

TABLE DES MATIERES. 245

B.C.M. soc.coop.
24, route de la Sapinière
B-4960 LOUVEIGNE.
BELGIQUE.

D/1983/3827/1

Comment faire des images graphiques, écrire des programmes rapides et efficaces en basic et langage machine? Comment faire démarrer seul un programme basic en allumant le VIC? Comment utiliser les poignées de jeu, les manches de commande, le photostyle, les connecteurs d'extension? Raccorder une imprimante, dialoguer avec un VIC par téléphone, construire soi-même des extensions mémoires?
Tout cela et bien d'autres choses se trouvent dans:

LE LIVRE DU VIC

Il est, par ses très nombreux exemples pratiques et son index en fin de volume, à la fois guide pour le débutant et ouvrage de référence pour l'amateur averti et le professionnel.

BCM

24, route de la Sapinière 4960 BANNEUX BELGIUM