



LA CONDUITE DE L'APPLE II

2. LE SYSTÈME GRAPHIQUE ET L'ASSEMBLEUR DE L'APPLE II

Vous désirez tout connaître de votre APPLE II ? Ce second tome de LA CONDUITE DE L'APPLE II vous est destiné :

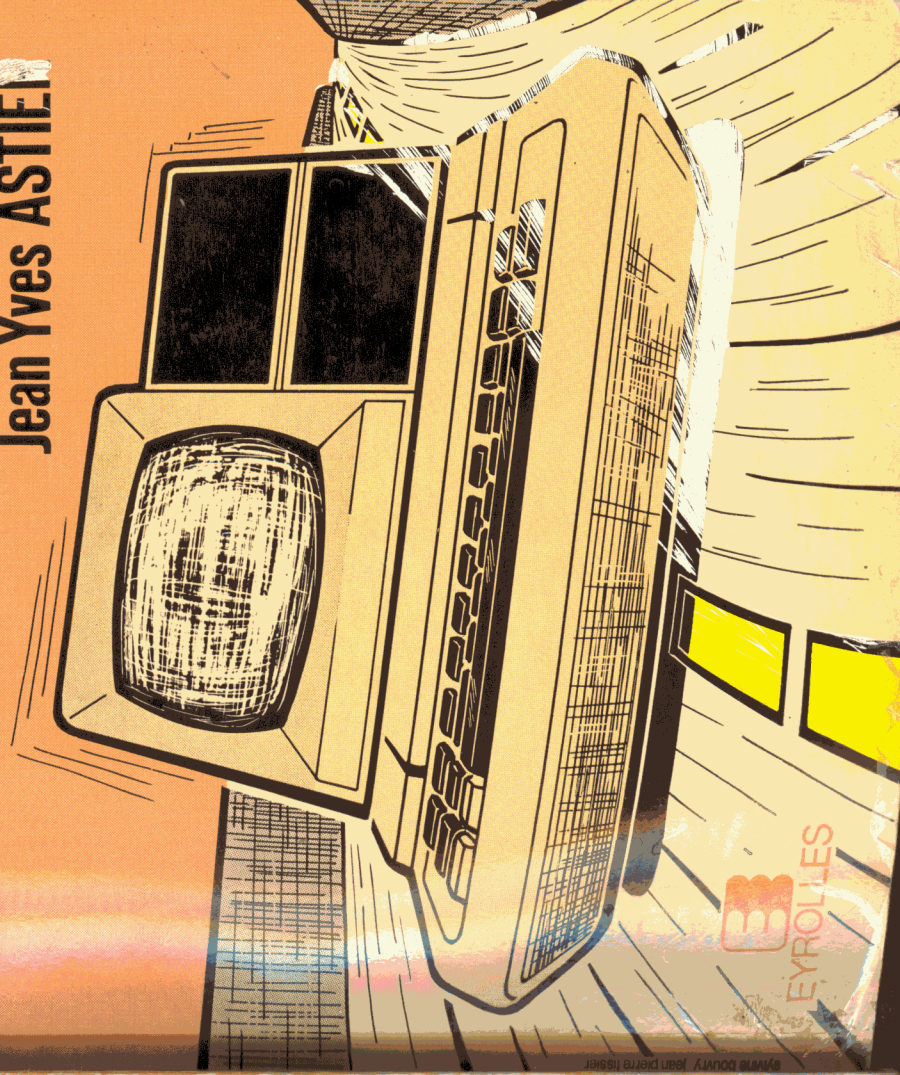
Le fonctionnement du système graphique, qui est l'un des atouts de votre machine, y est exposé en détail. Vous découvrirez, entre autres, qu'il est très facile d'animer un graphique ou un dessin.

Ensuite, la méthode pour programmer en assembleur est décrite. Vous saurez comment faire coexister et communiquer des programmes Basic et Assembleur.

Et afin de rendre vos programmes Assembleur vraiment efficaces, les principaux sous-programmes du moniteur et de l'Applesoft sont expliqués, ainsi que leur utilisation.

Ainsi, les seules limites de votre APPLE II seront-elles celles de votre imagination !

Jean Yves ASTIER



LA CONDUITE DE L'APPLE II

2. Le système graphique et l'assembleur de l'Apple II

DANS LA MÊME COLLECTION

SCHOMBERG Le Basic Universel.
SCHOMBERG Micro-ordinateurs : Comment ça marche ?
HERNANDEZ Pascal par l'exemple.
ASTIER La conduite de l'APPLE II.
NOLLET La conduite du ZX 81.
PELLIER La conduite du TRS 80.
KOSTAKIS & Tout sur les disques du TRS 80 (à paraître).
BAÏKOUR
LADEVIE Votre gestion avec BASIC sur micro-ordinateur.
QUEINNEC Langage d'un autre type : LISP.
LEPAPE L'assembleur facile du Z 80
MONTEIL L'assembleur facile du 6502
PELLIER Programmez vos jeux d'action rapide sur TRS 80.
OROS et ZX81 à la conquête des jeux.
PERBOST CP/M et sa famille.
DAX

LA CONDUITE DE L'APPLE II

2. Le système graphique et l'assembleur de l'Apple II

par

Jean-Yves ASTIER

Collection animée
par Richard SCHOMBERG

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1^{er} de l'article 40) ».

« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal ».

© Éditions EYROLLES, 1982



EYROLLES

61 boulevard Saint-Germain — 75005 Paris

1982

Table des matières

1. Le système graphique de l'Apple II	1
1.1. Vu de l'extérieur : les commandes basic graphiques	1
1.2. Vu de l'intérieur : comment ça marche	2
1.3. Le mode texte	7
1.4. Le mode graphique basse résolution	13
1.5. Le mode graphique haute résolution	16
1.6. Les tables de formes haute résolution	21

L'organisation des quatre pages graphiques, leurs utilisations sont décrites. Puis sont exposés le codage des images et des textes, les commandes d'affichages des pages. Vous saurez, entre autres, comment animer des graphiques, recopier des images, etc...

2. Programmation en assembleur sur Apple II	29
2.1. Intérêt de la programmation en assembleur	29
2.2. Le moniteur de l'Apple II et ses commandes	30
2.3. Créer des programmes en langage machine	39
2.4. Un premier exemple : test des fonctions de l'Applesoft	43
2.5. Un second exemple : l'Apple en musique	48
2.6. La carte mémoire du système Apple II	51
2.7. Des alliés puissants : les sous-programmes du moniteur et de l'Applesoft	61
2.8. Basic-Assembleur : Jonction réussie	77
2.9. Graphique et assembleur	80

Toute personne ayant, une fois dans sa vie, écrit un programme pour calculatrice de poche ou microprocesseur, sait programmer en assembleur. Aussi, pourquoi ne pas utiliser les 56 instructions du 6502 qui est le centre de votre Apple II ?

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

Les divers moyens de faire coopérer programmes basic et assembleur sont expliqués, exemples à l'appui. Vous saurez enfin comment utiliser les sous-programmes en langage machine du moniteur et de l'Applesoft, calculs en nombres réels, tracés en modes graphiques basse et haute résolution, etc...

Annexes

A. Les instructions du 6502	97
B. Les adresses de branchement du système	107
C. Le clavier	109

1

Le système graphique de l'Apple II

1.1. VU DE L'EXTÉRIEUR : LES COMMANDES BASIC GRAPHIQUES

L'Apple peut utiliser son écran de trois façons différentes, que l'on appelle le mode texte, le mode graphique basse résolution et le mode graphique haute résolution.

Le mode texte, comme son nom l'indique, permet d'afficher du texte et de dialoguer avec l'Apple grâce aux commandes INPUT et PRINT. L'écran est constitué de 24 lignes de 40 caractères chacune. Lorsque vous mettez en marche l'Apple, il est en mode texte.

Lorsque l'Apple n'est plus dans ce mode, on y revient par la commande TEXT. Depuis un programme, les commandes couramment utilisées sont PRINT, TAB, INPUT, VTAB, HTAB, POS, SPC, HOME, FLASH, INVERSE et NORMAL.

Le mode graphique basse résolution permet des constructions graphiques simples. On dispose de 16 couleurs différentes, et de quatre lignes de texte au bas de l'écran. Idéal pour représenter par exemple des pyramides d'âges, ou bien un jeu de mur de briques. On place l'Apple dans ce mode par la commande GR. On peut alors utiliser les commandes citées au paragraphe précédent pour

les quatre lignes de texte. Pour le reste de l'écran, on utilise COLOR, PLOT, HLIN, VLIN et SCRN.

Le mode graphique haute résolution offre une meilleure définition, mais on ne dispose plus que de 8 couleurs. On place l'Apple dans ce mode par les commandes HGR et HGR2. On peut alors utiliser les commandes HCOLOR, HPLOT, DRAW et XDRAW.

1.2. VU DE L'INTÉRIEUR : COMMENT ÇA MARCHÉ

a) Les secrets de l'image projetée en mémoire

Le système graphique de l'Apple fonctionne sur le principe d'image projetée en mémoire : Chaque point en mode graphique, ou chaque caractère en mode texte correspond à un emplacement déterminé en mémoire de l'Apple. En permanence, l'écran de télévision explore, lit ces emplacements mémoires et les nombres lus représentent soit la couleur des points en mode graphique, soit le caractère en mode texte. Pour faire apparaître un caractère ou une tache colorée en un point, il suffit donc de placer le bon code dans l'emplacement mémoire associé au point.

Par exemple, en mode texte, l'écran de télévision place dans le coin supérieur gauche de l'écran le caractère correspondant au nombre qu'il a lu dans la case mémoire de numéro 1024 (\$400 en hexadécimal).

En mode immédiat, tapez HOME puis

POKE 1024,193

Cette commande place dans l'octet 1024 la valeur 193. 193 est le code du caractère "A", il apparaît donc un "A" dans le coin supérieur gauche.

Tapez maintenant

POKE 1152,194:POKE 1153,67

1152 et 1153 sont les adresses mémoires des octets (cases) correspondant aux deux premiers caractères de la seconde ligne. 194 est le code du caractère "B". 67 est le code du caractère "C clignotant".

Et maintenant, encore plus fort. Toujours en mode immédiat, tapez

VTAB 20:GR

GR établit le mode graphique basse résolution. Tapez comme précédemment

POKE 1024,193

Là où il était apparu un "A", c'est-à-dire dans le coin supérieur gauche, sont apparus deux petits rectangles superposés, le premier est rouge, et celui placé en dessous est vert clair. Bien entendu, la commande POKE 1024,193 a toujours pour effet de placer la valeur 193 dans l'octet d'adresse 1024. La commande GR a signalé à l'écran que la valeur qu'il lit en 1024 et dans les autres cases représentait chacune la couleur de deux rectangles et non plus le code d'un caractère, comme en mode texte. Tapez maintenant

TEXT

L'écran est rempli de "@" noirs sur fond blanc, à l'exception du coin supérieur gauche où il y a un "A". En effet, la commande GR place des zéros dans toutes les cases correspondant à un rectangle élémentaire. Comme en mode graphique un zéro correspond à la couleur noire, l'écran apparaît donc noir juste après la commande GR. Puis, avec POKE 1024,193 vous avez remplacé la valeur 193 qui avait été effacée en 1024, et les deux rectangles rouge et vert sont apparus. Puis la commande TEXT a signalé à l'écran que les numéros devaient de nouveau être considérés comme des codes de caractères. TEXT, à l'inverse de GR ne modifie pas le contenu de la mémoire. La valeur 193 que vous avez placé en 1024 y est toujours, et les zéros que GR avait placés sont inchangés. Comme, en mode texte, 193 est le code du "A" et 0 le code du "@" noir sur fond blanc, on observe un "A" dans le coin supérieur gauche et des "@" noirs sur fond blanc ailleurs.

b) Entrons dans les détails : les quatre pages graphiques de l'Apple.

En modes TEXT et GR, la zone mémoire lue par l'écran de télévision débute en 1024 (\$400) et se termine en 2047 (\$7FF). Cette zone est appelée page 1 basse résolution. L'écran de l'Apple est capable de lire quatre zones mémoires distinctes.

- * de 1024 à 2047 (\$400 à \$7FF) la page 1 basse résolution. C'est elle qui est lue et utilisée si l'on exécute TEXT ou GR.
- * de 2048 à 3071 (\$800 à \$BFF) la page 2 basse résolution. N'est pas utilisable de façon simple en basic.
- * de 8192 à 16383 (\$2000 à \$3FFF) la page 1 haute résolution. C'est elle qui est lue et utilisée si l'on exécute HGR.
- * de 16384 à 24575 (\$4000 à \$5FFF) la page 2 haute résolution. Elle est lue est utilisée si l'on exécute HGR2.

c) Une question pertinente

Mais, me direz vous, si l'écran de l'Apple a le choix entre quatre zones distinctes, comment lui dire laquelle afficher ? Et si on affiche la page 1 basse résolution, comment lui faire comprendre que les octets doivent être traduits sur l'écran par des caractères en mode texte, ou bien par des rectangles en mode graphique basse résolution ? Et, si l'on a réussi à lui indiquer qu'il doit afficher la page 1 basse résolution en mode graphique, il faut encore lui indiquer si l'on doit ou ne doit pas garder quatre lignes de texte au bas de l'écran.

- Exprimé de façon plus ordonnée, il faut indiquer si l'on désire
- Etre en mode texte ou en mode graphique
 - avoir quatre lignes de texte ou non (mode graphique)
 - afficher une page 1 ou une page 2
 - être en mode basse ou haute résolution (mode graphique).

A chacune de ces quatre questions, une paire d'adresses est associée. Il suffit de référencer l'une ou l'autre adresse d'une paire pour déterminer l'une ou l'autre possibilité de chaque question. Concrètement, "référencer une adresse signifie" :

- exécuter une commande PEEK ou POKE à cette adresse, en basic. Dans le cas de PEEK, la valeur lue n'a aucun intérêt, aucune signification, c'est le seul fait de lire qui provoque l'action. Dans le cas de POKE, la valeur écrite peut être quelconque, c'est le seul fait d'écrire qui provoque l'action,

- en langage machine, exécuter une instruction LDA, STA ou BIT à cette adresse. Là aussi, la valeur lue ou écrite n'a aucune importance.

Voici ces quatre paires associées aux quatre questions précédentes :

- * 49232 (\$C050)	Etre en mode graphique
- * 49233 (\$C051)	Etre en mode texte
- * 49234 (\$C052)	En mode graphique, supprimer les 4 lignes de texte
- * 49235 (\$C053)	En mode graphique, garder 4 lignes de texte au bas de l'écran
- * 49236 (\$C054)	Afficher une page 1
- * 49237 (\$C055)	Afficher une page 2
- * 49238 (\$C056)	En mode graphique, choisir la basse résolution
- * 49239 (\$C057)	En mode graphique, choisir la haute résolution

Exemples : Si vous voulez utiliser la page 2 haute résolution avec les quatre lignes de texte en bas de l'écran, vous devez

- être en mode graphique
- garder quatre lignes de texte au bas de l'écran
- afficher une page 2
- choisir la haute résolution

Ce qui, en basic, se fait grâce à

POKE 49232,0:POKE 49235,0
POKE 49237,0:POKE 49239,0

Puis, si maintenant vous désirez afficher la page 2 basse résolution en mode texte, vous devez

- Etre en mode texte
- Afficher une page 2

Comme, dans cet exemple, une page 2 est déjà affichée, il suffit de passer en mode texte par
POKE 49235,0

Alors que les commandes basic usuelles n'autorisent que 4 modes d'utilisation de l'écran (TEXT, GR, HGR et HGR2), ces adresses permettent 10 modes d'utilisation qui sont :

* Afficher la page 1 basse résolution en mode texte (Comme TEXT)
Pour cela,

POKE 49236,0:POKE 49233,0

* Afficher la page 2 basse résolution en mode texte
Pour cela,

POKE 49237,0:POKE 49233,0

* Afficher la page 1 basse résolution en mode graphique avec 4 lignes de texte (GR)
Pour cela,

POKE 49236,0:POKE 49232,0
POKE 49238,0:POKE 49235,0

* Afficher la page 2 basse résolution en mode graphique avec 4 lignes de texte
Pour cela,

POKE 49237,0:POKE 49235,0
POKE 49238,0:POKE 49232,0

* Afficher la page 1 basse résolution en mode graphique sans texte
Pour cela,

POKE 49236,0:POKE 49234,0
POKE 49238,0:POKE 49232,0

* Afficher la page 2 basse résolution en mode graphique sans texte
Pour cela,

POKE 49237,0:POKE 49234,0
POKE 49238,0:POKE 49232,0

* Afficher la page 1 haute résolution avec 4 lignes de texte (comme HGR)

Pour cela,

POKE 49236,0:POKE 49235,0
POKE 49239,0:POKE 49232,0

* Afficher la page 2 haute résolution avec 4 lignes de texte
Pour cela,

POKE 49237,0:POKE 49235,0
POKE 49239,0:POKE 49232,0

* Afficher la page 1 haute résolution sans texte
Pour cela,

POKE 49236,0:POKE 49234,0
POKE 49239,0:POKE 49232,0

* Afficher la page 2 haute résolution sans texte (comme HGR2)
Pour cela,

POKE 49237,0:POKE 49234,0
POKE 49239,0:POKE 49232,0

1.3. LE MODE TEXTE

a) On a perdu des cases mémoires

On peut afficher en mode texte la page 1 basse résolution (de 1024 à 2047) ou la page 2 basse résolution (de 2048 à 3071). Chacune de ces pages comporte 1024 octets. Or en mode texte, l'écran comporte 24 lignes de 40 caractères, soit $24 \times 40 = 960$ caractères en tout. Comme à chaque caractère est associé un octet en mémoire, 960 octets sont utilisés. Il y a donc dans chacune des deux pages basse résolution $1024 - 960 = 64$ octets qui ne sont pas utilisés par le système graphique. Rassurez-vous ces octets ne sont pas perdus pour tout le monde. Ce sont les cartes d'interfaces de périphériques qui utilisent ces octets.

Etant donné un caractère en position (X,Y) sur l'écran, avec $0 \leq X \leq 39$ et $0 \leq Y \leq 23$, les lignes suivantes calculent l'adresse mémoire de l'octet associé à ce caractère dans la page 1 basse résolution.

```
AD=128*Y+X+1024
IF 7<Y THEN AD=AD-984
IF 15<Y THEN AD=AD-984
```

L'octet de la page 2 basse résolution associé au caractère en position (X,Y) a pour adresse la valeur AD calculée par :

```
AD=128*Y+X+2048
IF 7<Y THEN AD=AD-984
IF 15<Y THEN AD=AD-984
```

Attention : N'employer ces formules que pour $0 \leq X \leq 39$ et $0 \leq Y \leq 23$. Sinon, le résultat obtenu peut ne pas être celui d'un octet utilisé par le système graphique. Et si vous exécutez une instruction POKE à l'adresse ainsi calculée, vous risquez de détruire un octet de votre programme, ou bien un octet utilisé par le système Apple.

b) Les codes écrans des caractères

Vous savez déjà que 193 et 194 sont les codes des lettres "A" et "B", que 67 est le code du "C clignotant". A chaque caractère correspond un code, dont voici la liste.

Tout d'abord signalons que

- les codes de 0 à 63 (\$00 à \$3F) sont ceux des caractères noirs sur fond blanc
- les codes de 64 à 127 (\$40 à \$7F) sont ceux des caractères clignotants
- les codes de 160 à 223 (\$A0 à \$DF) sont ceux des caractères normaux
- les codes de 128 à 159 (\$80 à \$9F) et de 224 à 225 (\$E0 à \$FF) sont inutilisés.

\$00	00	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
\$10	16	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
\$20	32	!	"	#	\$	%	&	'	()	*	+	,	.	/		
\$30	48	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	

Table des codes des caractères noirs sur fond blanc

Note : Le code 32 (\$20) correspond à un carré blanc.

\$40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
\$50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
\$60	96	!	"	#	\$	%	&	'	()	*	+	,	.	/		
\$70	112	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	

Table des codes des caractères clignotants

Note : le code 96 (\$60) est le code du curseur clignotant.

\$A0	160	!	"	#	\$	%	&	'	()	*	+	,	.	/		
\$B0	176	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
\$C0	192	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
\$D0	208	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-

c) Rétrécir ou agrandir la fenêtre

Lorsque vous mettez en marche l'Apple, ou lorsque vous exécutez TEXT, la totalité de la surface de l'écran est utilisée. Il est possible de réduire cette surface, que l'on appelle la fenêtre. Cela sert à protéger tout ce qui est à l'extérieur de la fenêtre.

Exemple : Tapez et exécutez le programme suivant :

```
10 TEXT : HOME : VTAB 2: HTAB 6
20 FOR I = 1 TO 30: PRINT "$";: NEXT : PRINT
30 FOR I = 1 TO 17
40 PRINT TAB( 6); "$"; TAB( 35); "$": NEXT
50 HTAB 6: FOR I = 1 TO 30: PRINT "$";: NEXT : PRINT
60 PRINT : PRINT TAB( 15); "ZONE PROTÉGEE"
70 POKE 32,6: POKE 33,28: POKE 34,2
80 POKE 35,16: HOME : LIST : END
```

N'oubliez pas les ; des lignes 20 et 50 !

Le programme efface l'écran, dessine un cadre (ligne 20 à 60) puis définit une fenêtre qui correspond à l'intérieur du cadre tracé. Pour rétablir la fenêtre à toute la surface de l'écran, utilisez la commande TEXT.

Voici les commandes qui permettent de définir la fenêtre

```
POKE 32,MG
POKE 33,L
POKE 34,H
POKE 35,B
```

MG est compris entre 0 et 39 et définit la largeur de la marge, à gauche de l'écran. TEXT fixe une largeur nulle (pas de marge).

L est compris entre 1 et 40 et définit la longueur d'une ligne. Respecter les deux règles suivantes :

- ne jamais prendre L nul. L'Apple ne peut plus rien afficher et il se met à boucler
- on doit avoir $MG+L \leq 40$. Sinon l'Apple peut écrire dans des zones où il ne devrait pas.

H est compris entre 0 et 23. C'est le numéro de la 1^{re} ligne de la fenêtre.

B est compris entre 0 et 23. C'est le numéro de la dernière ligne de la fenêtre.

Il est conseillé d'exécuter une commande HOME après avoir défini une fenêtre. En effet, l'Applesoft garde en mémoire (en 37) le numéro de la ligne où il doit imprimer le prochain caractère. Ce numéro n'est pas modifié par les quatre commandes POKE données ci-dessus, si il se trouve être inférieur à la valeur H ou supérieur à la valeur B, l'Applesoft pourra écrire à l'extérieur de la fenêtre. La commande HOME fixe ce numéro à la valeur H.

Attention : Les commandes HTAB et VTAB ignorent la fenêtre, et peuvent placer le curseur à l'extérieur de cette fenêtre. De façon plus générale, on peut dire que lorsque le curseur est placé à l'intérieur de la fenêtre, il ne peut en sortir et tout ce qu'imprime l'Applesoft reste à l'intérieur de la fenêtre. Si par contre vous placez le curseur hors de la fenêtre (par HTAB, VTAB ou POKE) l'Applesoft peut écrire à l'extérieur de la fenêtre.

Cette notion de fenêtre n'est valable que pour l'Applesoft, qui à chaque fois qu'il doit afficher un caractère, tient compte de la fenêtre définie, et vérifie qu'il en respecte bien les limites. Mais une commande POKE 1024,193 vous permet toujours d'écrire un "A" dans le coin supérieur gauche, même s'il se trouve à l'extérieur de la fenêtre.

La commande basic TEXT rétablit la fenêtre à toute la surface de l'écran. Si donc, ayant défini une fenêtre puis commuté le mode HGR ou HGR2, vous désirez revenir en mode texte sans modifier votre fenêtre, utilisez POKE 49236,0;POKE 49233,0 au lieu de TEXT.

d) Les curseurs de l'Applesoft

Profitons de ce paragraphe pour expliquer ce qu'est un curseur : Il s'agit tout simplement d'un groupe de cases mémoires dont les

contenus indiquent la position sur l'écran de la prochaine entité à afficher. L'Applesoft en possède deux.

Le premier curseur est constitué des cases mémoires 36 et 37 et indiquent la position du prochain caractère à afficher en mode texte. Plus précisément, il y a en 36 un nombre compris entre 0 et 39 qui indique la position horizontale, par rapport à la marge, du prochain caractère à afficher. En 37, il y a le numéro de la ligne où sera affiché le prochain caractère. Ce numéro est compris entre 0 et 23.

Voici les commandes relatives à ce curseur :

```
CH=PEEK(36)
CV=PEEK(37)
POKE 36,CH ou bien HTAB CH
POKE 37,CV ou bien VTAB CV
```

Les deux premières permettent de lire la position du curseur, les autres permettent de positionner ce curseur.

Le second curseur est utilisé pour les modes graphiques haute résolution, voir le chapitre correspondant.

e) Et la page 2, alors ?

Comme vous le savez la commande TEXT permet d'afficher la page 1 basse résolution en mode texte. Vous pouvez faire afficher la page 2 basse résolution en mode texte grâce à POKE 49237,0:POKE 49233,0. Mais il se produit alors un phénomène fort désagréable : lorsque vous frappez sur une touche, le caractère correspondant ne s'affiche plus sur l'écran. En effet, l'Applesoft n'a pas été prévu pour être utilisé avec la page 2 basse résolution. Les codes écrans de tous les caractères sont toujours placés dans la page 1 basse résolution, quel que soit le mode graphique.

La seule commande basic permettant de placer quelque chose dans la page 2 basse résolution est POKE. Ce qui signifie que cette

page est en pratique inutilisable. D'ailleurs l'Applesoft range normalement les programmes basic à partir de l'adresse mémoire 2049(\$801), c'est-à-dire exactement dans cette page.

Néanmoins, elle peut être utilisée dans le cas suivant : lorsque l'on désire garder une page de texte en mémoire. Pour cela il faut :

Avant de charger en mémoire le programme, exécuter

```
JCALL--151"return"
*67:011 0C 03 03 0C 03 0C 03 0C 03 0C"return"
*AF:033 0C ctrl-C"return"
JCALL:98"return"
```

Ceci obligera l'Applesoft à placer ce programme à la fin de la page 2 basse résolution. Charger le programme. Celui-ci devra d'abord composer à l'aide de PRINT, TAB, HTAB, VTAB, SPC, la page de texte que l'on désire conserver. Puis exécuter (ceci est assez long)

```
FOR X=0 TO 39:FOR Y=0 TO 23
AD=128*Y+X+1024:IF 7<Y THEN AD=AD-984
IF 15<<Y THEN AD=AD-984
POKE AD+1024, PEEK(AD):NEXT:NEXT
```

Ce qui recopie la page 1 dans la page 2. Vous pouvez exécuter de nouvelles commandes PRINT,HOME, etc. qui modifieront la page 1, mais pas la page 2. Chaque fois que vous désirez faire apparaître la page 2, utilisez

POKE 49237,0:POKE 49233,0. Pour revenir à la page 1, utilisez TEXT.

1.4. LE MODE GRAPHIQUE BASSE RÉOLUTION

a) L'écran est divisé en 40 x 40 blocs de couleur plus quatre lignes de texte ou bien 40 x 48 blocs de couleur. Comme pour le mode texte, on peut en mode graphique basse résolution afficher la page 1 basse résolution (de 1024 à 2047) ou la page 2 basse résolution

(de 2048 à 3071). Seulement 960 des 1024 octets que comporte chaque page sont utilisés par le mode graphique basse résolution, les 64 octets restant sont utilisés par les cartes d'interfaces.

Chacun de ces 960 octet correspond à deux rectangles élémentaires de couleur placés l'un au-dessus de l'autre. De manière plus précise, les quatre bits de poids fort d'un octet déterminent la couleur du rectangle inférieur, les quatre bits de poids faible déterminent la couleur du rectangle supérieur.

Quatre bits peuvent être représentés par un nombre décimal compris entre 0 et 15, ou par un nombre hexadécimal compris entre \$0 et à \$F. Voici les couleurs correspondant à ces valeurs.

binaire	décimal	Hexa.	Couleur	binaire	décimal	Hexa.	Couleur
0000	0	\$0	noir	1000	8	\$8	vert foncé
0001	1	\$1	violet très foncé	1001	9	\$9	vert clair
0010	2	\$2	bleu	1010	10	\$A	gris
0011	3	\$3	bleu ciel	1011	11	\$B	gris bleu
0100	4	\$4	rouge	1100	12	\$C	ocre
0101	5	\$5	orange	1101	13	\$D	jaune
0110	6	\$6	rose tyrien	1110	14	\$E	beige
0111	7	\$7	rose	1111	15	\$F	blanc

Les correspondances entre couleur et numéro sont les mêmes que pour la commande basic COLOR=

b) Encore la page 2 I

De même qu'en mode texte, la page 2 basse résolution n'est pas utilisable en pratique. Si grâce aux commandes POKE données en 1.2.c vous faites afficher la page 2 basse résolution, vous ne verrez rien puisque les commandes PLOT, HLIN et VLIN n'affectent que la page 1 basse résolution exclusivement.

Néanmoins, le sous-programme donné en 1.3.e qui recopie la page 1 dans la page 2 vous permet de stocker une image dans la

page 2, après l'avoir composée dans la page 1. Puis POKE 49237,0 POKE 49238,0: POKE 49232,0 font apparaître cette image.

c) Les quatre lignes de texte

La commande GR affiche la page 1 basse résolution en mode texte en laissant subsister 4 lignes de texte en bas de l'écran. Pour supprimer ces lignes, et utiliser la totalité de l'écran en mode graphique, exécutez POKE 49234,0 après la commande GR. Pour retrouver les quatre lignes de texte, exécutez GR, ce qui efface la partie graphique de l'écran, ou bien POKE 49235,0, ce qui n'efface pas cette partie graphique.

d) Les bonnes adresses

* La commande COLOR = arg, avec "arg" compris entre 0 et 15 est équivalente à la commande.

POKE 48, arg + 16* arg

C'est-à-dire que arg peut être représenté par un nombre de 4 bits, et que ces 4 bits sont placés dans les 4 bits de poids faible et dans les 4 bits de poids fort de l'octet 48.

* Le programme suivant permet de calculer l'adresse mémoire dans la page 1 basse résolution de l'octet contenant les 4 bits correspondant au point de coordonnées (X, Y).

```
AD=128*(Y/2)+X+1024
IF 15<=Y THEN AD=AD-984
IF 31<=Y THEN AD=AD-984
```

Si Y est pair, les 4 bits correspondant au point sont les 4 bits de poids faible de l'octet d'adresse AD.

Si Y est impair, les 4 bits correspondant au point sont les 4 bits de poids fort de l'octet d'adresse AD.

N'utilisez ce petit programme que pour $0 \leq X \leq 39$ et $0 \leq Y \leq 47$. Sinon le résultat AD peut être l'adresse d'un octet et n'appartenant pas à la page graphique.

1.5. LE MODE GRAPHIQUE HAUTE RÉOLUTION

a) L'écran est constitué de 280×160 points de couleur plus 4 lignes de texte ou de 280×192 points. On peut afficher la page 1 haute résolution (de 8192 à 16383) ou la page 2 haute résolution (de 16384 à 24575).

A chaque point de l'écran correspond un bit en mémoire. Lorsque ce bit vaut 0, le point est éteint, lorsqu'il vaut 1 le point est allumé. Dans chaque octet de la page haute résolution, les sept bits de poids faible, correspondent chacun à un point de l'écran, le huitième bit est utilisé pour déterminer la couleur des sept autres. A chaque ligne de l'écran correspond 40 octets d'adresses consécutives. Comme dans chaque octet, sept bits sont affichés, on a donc $40 \times 7 = 280$ points par ligne. Le bit de poids faible du 1^{er} octet correspond au point de colonne 0, le septième bit du 1^{er} octet correspond au point en colonne 6, le bit de poids faible du second octet correspond au point en colonne 7, etc...

Lorsqu'un bit est à 0, le point correspondant est éteint, noir. Cela est simple. Par contre, lorsqu'un bit vaut 1, le point correspondant est allumé et il est moins simple de savoir quelle est sa couleur. Il faut savoir que :

- un point allumé situé dans une colonne de numéro pair ne peut être que violet ou bleu
- un point allumé situé dans une colonne impaire ne peut être que vert ou rouge
- deux points consécutifs allumés d'une même ligne apparaissent toujours tous les deux blancs.

Ainsi, lorsque deux bits correspondant à deux points consécutifs d'une même ligne sont à « 1 », les deux points correspondant sont blancs. Lorsque l'un de ces bits est à « 1 » et l'autre à 0, le point correspondant au bit 1 peut être soit violet soit bleu s'il est dans une colonne paire, soit vert soit rouge s'il est dans une colonne impaire. Comment choisir entre ces deux couleurs possibles ? Cela se fait grâce au huitième bit de chaque octet.

Si le huitième bit est nul, les bits correspondant à une colonne paire donnent des points bleus alors que les bits correspondant à une colonne impaire donnent des points rouges.

Si le huitième bit vaut 1, les bits correspondant à une colonne paire donnent des points violets alors que les bits correspondant à une colonne impaire donnent des points verts.

Ainsi, si vous voulez un écran uniformément vert, il faut que dans chaque octet de la page graphique haute résolution :

- le huitième bit vale 1
- les bits correspondant à des colonnes paires valent 0
- les bits correspondant à des colonnes impaires valent 1.

Pour un écran uniformément blanc, il faut que les sept bits de poids faibles de tous les octets valent 1. Dans ce cas la valeur du huitième bit n'a pas d'importance. Les octets doivent donc être égaux à \$FF ou \$7F.

b) La page 2, enfin !

Alors que pour les modes texte et basse résolution, la page 2 basse résolution est pratiquement inutilisable, la page 2 haute résolution est elle parfaitement utilisable. Alors que PLOT, VLIN et HLIN, en mode basse résolution, ne peuvent tracer qu'à l'intérieur de la page 1 basse résolution, HPLOT, DRAW et XDRAW peuvent tracer dans la page 1 ou dans la page 2 haute résolution.

Après une commande HGR, on affiche la page 1 haute résolution et HPLOT, DRAW et XDRAW tracent dans cette page. Après une commande HGR2, on affiche la page 2 haute résolution et l'on y trace.

c) Ça bouge !

Pour donner une impression de mouvement, il faut faire apparaître une page, mettons la page 1, tandis que cette page 1 est affichée exécuter un tracé dans la page 2, puis faire apparaître la page 2,

tandis que la page 2 est affichée exécuter un tracé légèrement différent dans la page 1, faire apparaître la page 1, etc...

Les seules commandes HGR et HGR2 ne permettent pas ceci car elles provoquent un tracé dans la page affichée, alors qu'il faudrait provoquer un tracé dans la page qui n'est pas affichée. On utilise les cinq commandes suivantes

```
POKE 49236,0 ou POKE-16300,0 afficher la page 1
POKE 49237,0 ou POKE-12299,0 afficher la page 2
POKE 230,32 ou POKE-65306,32 tracer dans la page 1
POKE 230,64 ou POKE-65306,64 tracer dans la page 2
CALL 62450,46 ou CALL-3086 effacer la page où l'on trace
```

Elles s'utilisent comme suit :

- au début du programme exécuter une fois HGR:HGR2. Ceci efface les deux pages et initialise le mode graphique haute résolution.
- POKE 230,32:POKE-16299,0 permet d'afficher la page 2 tandis que l'on trace dans la page 1
- POKE 230,64:POKE-16300,0 permet d'afficher la page 1 tandis que l'on trace dans la page 2

Voici un exemple de courbe se déformant (figure de lissage 1-2)

\$LIST

```
5 REM COURBE DE LISSAJOU ANIMEE
10 HGR : HGR2 : HCOLOR= 3:PI = 6.283185
20 FOR I = 0 TO PI STEP .1
30 POKE 230,64: POKE - 16300,0: GOSUB 60
40 POKE 230,32: POKE - 16299,0: GOSUB 60
50 NEXT I: END
60 CALL 62450,62446
65 HPLOT 278,95 * (1 + SIN (I))
70 FOR J = 0 TO PI STEP .5
80 HPLOT TO 139 * (1 + COS (J)),95 * (1 + SIN (2 * J + I))
90 NEXT J
95 HPLOT TO 278,95 * (1 + SIN (I)): RETURN
```

d) Une astuce pour colorier l'écran

HCOLOR=V:HPILOT 0,0:CALL-3082 (62454)

Ceci efface l'écran haute résolution en lui donnant la couleur V

e) Les bonnes adresses

* \$E4 contient un masque correspondant à la valeur définie par la dernière commande HCOLOR (\$E4 vaut 228).

Valeur de HCOLOR	0	1	2	3	4	5	6	7
	noir	ocre	bleu	blanc	noir	vert	rouge	blanc
Valeur rangée en \$E4	\$00 0	\$2A 42	\$55 85	\$7F 127	\$80 128	\$AA 170	\$D5 213	\$FF 255

POKE 228,213 est exactement équivalent à HCOLOR=6

* \$E6 (230 en décimal) contient \$20 (32) si l'on trace dans la page 1, et \$40 (64) si l'on trace dans la page 2. Ces valeurs sont les octets de poids fort des adresses de début de page, rappelons que la page 1 débute en \$2000 et la page 2 en \$4000.

* \$26, \$27, \$E5, \$30 et \$1C constituent le curseur graphique haute résolution. Dire que le curseur est positionné sur un point M(x,y) de l'écran signifie que

- A chaque ligne de l'écran sont associés 40 octets d'adresses consécutives. \$26 et \$27 contiennent respectivement les parties basses et hautes de l'adresse mémoire du premier octet de la ligne y.

- \$E5 contient le numéro d'octet dans une ligne correspondant à l'abscisse x. Comme chaque octet correspond à 7 points, \$E5 contient donc la partie entière de x/7. Etant donné un point M(x,y), l'adresse mémoire de l'octet contenant le bit associé à ce point est égale à la somme des nombres contenus en (\$26-\$27) et \$E5. Pratiquement, pour accéder à cet octet, on charge dans le registre Y la valeur contenue en \$E5, l'adresse de l'octet est alors (\$26).Y

Exemple : LDY \$E5

LDA (\$26).Y charge cet octet dans l'accumulateur.

- \$30 contient un masque qui permet d'isoler le bit correspondant au point M(x,y) dans l'octet dont l'adresse est définie par \$26, \$27 et \$E5.

numéro du bit (0 = poids faible)	0	1	2	3	4	5	6
valeur du masque en \$30	\$81	\$82	\$84	\$88	\$90	\$A0	\$C0

- \$1C. Si x est pair, \$1C contient le "et" logique des valeurs contenues en \$30 et \$E4. Si x est impair, \$1C contient le "et" logique de la valeur contenue en \$30 et de celle obtenue en décalant d'un bit les bits 0 à 6, de la valeur contenue en \$E4. Ce masque sert à positionner le bit correspondant au point M(x,y), en tenant compte de la couleur définie par la valeur en \$E4.

* (\$E0-\$E1) et \$E2 contiennent les coordonnées de l'extrémité d'un segment tracé par HPLLOT.

Par exemple si vous exécutez la commande HPLLOT TO X1, Y1

- les parties basses et hautes de X1 sont placées en \$E0 et \$E1 respectivement alors que Y1 est placé en \$E2.

- Puis le curseur est déplacé point par point depuis l'endroit où il se trouvait jusqu'au point (X1,Y1) dont les coordonnées sont en \$E0, \$E1, \$E2. A chaque déplacement, un point est tracé.

Si vous exécutez la commande HPLLOT X2, Y2 TO X3, Y3

- le curseur est placé en (X2,Y2)

- X3 est placé en (\$E0-\$E1) tandis que Y3 est mis en \$E2

- Puis le curseur est déplacé comme expliqué précédemment.

1.6. LES TABLES DE FORMES HAUTE RÉOLUTION

a) Ce que c'est :

Les instructions DRAW et XDRAW permettent de tracer et d'effacer une forme en une seule commande basic. Cette forme peut être placée n'importe où sur l'écran, on peut la grossir (commande SCALE =), la faire tourner (commande ROT =). Vous disposez donc d'un motif, que vous pouvez très simplement répéter autant de fois que nécessaire.

Exemples :

- Si vous désirez disposer d'autres jeux de caractères alphanumériques, vous définirez et utiliserez une forme (shape) pour chaque caractère.

- Dans le cas des jeux, l'inévitable petit vaisseau spatial ainsi que les nombreux projectiles qu'il émet seront des formes.

Une table de formes est un groupe constitué d'une ou plusieurs formes (255 au maximum), précédées de valeurs qui permettent à l'AppleSoft de déterminer le début de chaque forme.

b) Dessiner une forme

Une forme est uniquement composée de points de l'écran haute résolution qui en comporte 280 x 192. Dessiner une forme signifie donc placer des points dans une grille. Munissez vous donc d'un bord de papier quadrillé. Placez les points de la forme à raison d'un point par carreau. Puis, en suivant les lignes du quadrillage, tracer un chemin formé de petits vecteurs horizontaux décrivant la figure.

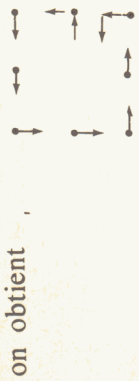
Exemple : pour tracer

. . .

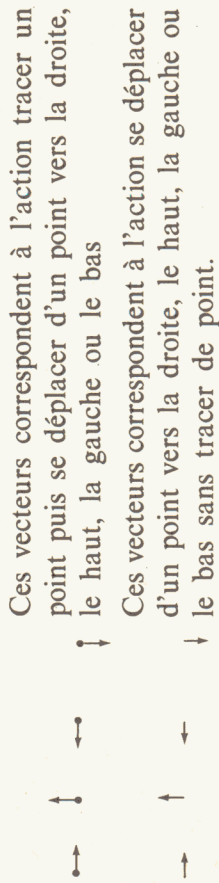
. . .

. . .

en partant du centre puis en y revenant



Vous pouvez utiliser huit types de vecteurs



Puis il faut coder cette suite de vecteurs, en suivant le sens du parcours.

Dans l'exemple précédent, il faut coder la suite



c) Coder la forme

Cette suite de vecteurs va être codée par une suite d'octets d'adresses mémoires consécutives. Chaque octet est divisé en trois parties



Chacune de ces trois parties code un vecteur, dans l'ordre A, B, C.

Pour les sections A et B, constituées de 3 bits, on a correspondance suivante

le vecteur	est codé par les 3 bits
↑	000
→	001
↓	010
←	011
↗	100
↘	101
↖	110
↙	111

Pour la section C, continuée de 2 bits, on a la correspondance

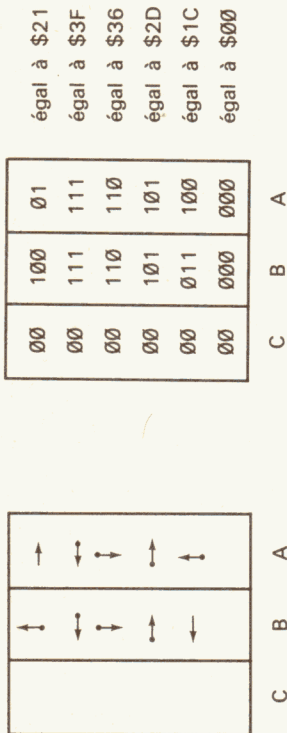
le vecteur	est codé par les 2 bits
→	01
↓	10
←	11

Cette section ne peut contenir que l'un des trois vecteurs →, ↓, ou ←.

Puis l'on code la suite de vecteurs, en se souvenant que le premier est codé par la section A, le second par la section B, etc...

Il faut savoir que lorsque la section C est nulle, elle est simplement ignorée pour DRAW et XDRAW. Lorsque les sections B et C sont toutes deux nulles, elles sont ignorées. Et lorsque les trois sections sont nulles, cela indique la fin de la définition de la forme. Une définition de forme doit toujours se terminer par un octet nul.

En reprenant l'exemple précédent



Le premier vecteur, →, est placé dans la section A du premier octet. Le second, ↑, va dans la section B de l'octet. Le troisième vecteur, ↔ ne peut être mis dans la section C du premier octet (voir table), on place donc deux zéros dans la section C qui sera donc ignorée par DRAW et XDRAW, et l'on met le vecteur dans la section A du second octet. Le vecteur suivant, ↔ est placé dans la section B du second octet. De même, le cinquième vecteur, ↗ ne peut être mis dans la section C. On met deux zéros dans la section C et l'on place le vecteur dans la section A de l'octet suivant. Et ainsi de suite. Puis l'on ajoute à la fin, un octet nul, qui indique à DRAW et XDRAW la fin de la forme. ↑

REMARQUE. Lorsque les sections B et C sont nulles elles sont ignorées. Lorsque les sections A,B,C sont nulles, le tracé s'arrête. Il n'est donc pas possible de coder plus de deux vecteurs ↑ consécutifs.

Il est donc impossible de coder ↑↑↑
Aussi, il faut éviter les suites de déplacements vers le haut lorsqu'on dessine la figure. Ou bien, on peut remplacer la suite ↑↑↑ ... par ↑ → ↑ ← ... qui produit le même effet et est codable.

Puis l'on traduit ces nombres binaires en nombres hexadécimaux. Les quatre bits de gauche correspondent au chiffre hexadécimal de gauche, les quatre bits de droite correspondent au chiffre hexadécimal de droite.

Les 4 bits	correspondent au chiffre hexadécimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

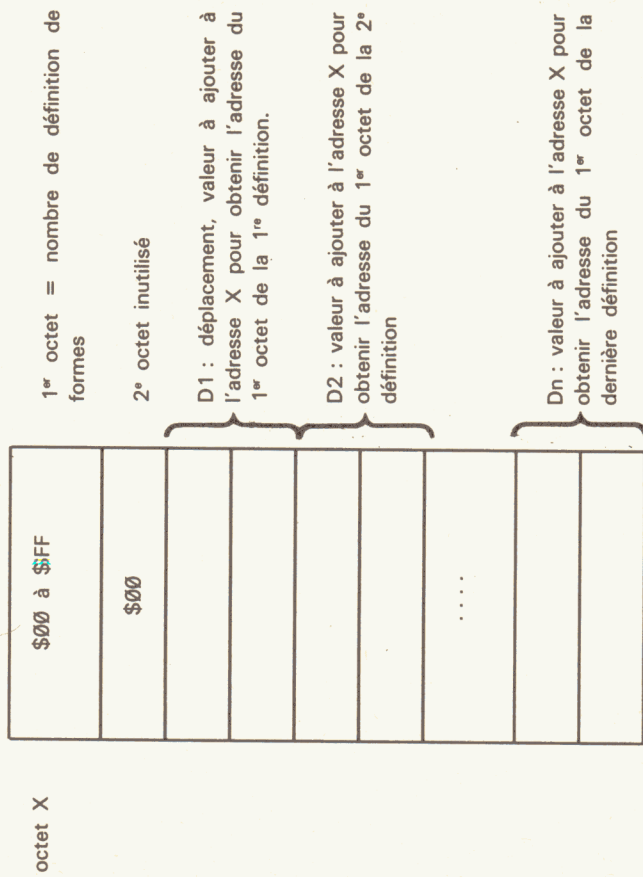
On obtient donc finalement pour notre exemple \$21, \$3F, \$36, \$2D, \$1C, \$00.

d) **Encore un effort, la table est presque terminée**

Une table peut être constituée de 1 à 255 définitions de formes, telles l'exemple précédent. Les groupes d'octets constituant les diverses définitions doivent être placés les uns à la suite des autres. Le premier bloc pourra être tracé par la commande DRAW 1, le second bloc par la commande DRAW 2, etc...

Ces définitions doivent être précédées d'un en-tête, c'est-à-dire d'un certain nombre d'octets indiquant le nombre total de définitions de formes, et les valeurs à ajouter à l'adresse de début de l'en-tête pour obtenir les adresses de début des définitions de formes.

Puis, il faut placer tout ceci en mémoire, soit en le tapant à l'aide du moniteur, soit par des commandes POKE. Pour parachever le tout, l'adresse du premier octet de l'index doit être placée en \$E8-\$E9. Ceci pour indiquer à l'AppleSoft où il peut trouver cette table.



Exemple : Nous allons créer une table qui ne contient qu'une définition de forme, celle des exemples précédents.

\$01
\$00
\$04
\$00
\$21
\$3F
\$36
\$2D
\$1C
\$00

La table est :

Nombre de définition, ici 1.

Inutilisé

il faut ajouter \$0004 à l'adresse du premier octet de la table pour obtenir celle du premier octet de la définition.

définition de la forme

dernier octet nul

Note, ici la définition est placée immédiatement après l'index. Cela n'est pas nécessaire, bien que courant.

Pour placer la table en mémoire ainsi que son adresse de début, ici \$300 :

```
]CALL-151 "return"  
*300:01 00 04 00 21 3F 36 2D 1C 00 "return"  
*E8:00 03 "return"  
*"ctr-C" "return"  
HGR:HCOLOR=3:ROT=0:SCALE=6:DRAW 1 AT 9 , 9
```

Pour sauvegarder la table sur cassette il faut connaître :

- l'adresse du début, ici \$0300 et l'adresse du dernier octet, ici \$0309
 - la différence entre les deux adresses précédentes, ici \$0009
- Il faut placer en \$00-\$01 ce dernier nombre, puis utiliser la commande W du moniteur pour recopier le nombre et la table.

```
]CALL-151  
*0:09 00 "return" (On place $0009 en $00-$01)  
*0.1W 300.309W Stop! ne frappez pas "return" !
```

Mettez d'abord le magnétophone en marche, en enregistrement. Puis frappez "return". A la fin de l'enregistrement, l'Apple émet un bip ! Stoppez le magnétophone puis frappez "ctrl-C" "return"

Pour recharger la table, utilisez SHLOAD (Voir § 2.2. Tome 1).

2

Programmation en assembleur sur Apple II

2.1. INTÉRÊT DE LA PROGRAMMATION EN ASSEMBLEUR

Les langages évolués, comme le basic, simplifient considérablement la tâche du programmeur. Mais cela au prix d'une perte de temps importante à l'exécution du programme. Pour chaque ligne de programme basic, il faut rechercher la ligne en mémoire, vérifier que sa syntaxe est correcte, rechercher en mémoire les variables qu'elle utilise, effectuer des conversions puis enfin l'exécuter. C'est-à-dire que le microprocesseur n'effectue des calculs, des traitements "intéressants" pour le programmeur que pendant une fraction de son temps. Aussi, lorsque l'on a à effectuer un traitement simple mais nécessitant un grand nombre d'itérations, la programmation en assembleur est la meilleure solution.

Le calcul matriciel, l'animation de graphiques en sont deux exemples.

Le microprocesseur utilisé par l'Apple s'appelle "6502". Le but de cet ouvrage n'est pas d'exposer le fonctionnement détaillé du 6502, il existe des volumes entiers consacrés à ce sujet. Précisons que dans la pratique, les expressions "programmation en assembleur" et "programmation en langage machine" ont la même signification.

Une liste des instructions du 6502 est donnée en annexe.

2.2 LE MONITEUR DE L'APPLE II ET SES COMMANDES

a) **Le moniteur est un programme implanté en ROM**, c'est-à-dire qu'il est en permanence dans l'Apple, et il s'étend de l'adresse \$F800 jusqu'à \$FFFF.

Il vous permet de lire le contenu de n'importe quelle case mémoire, et de changer le contenu de n'importe quelle case mémoire en RAM.

Contrairement au basic qui utilise la numération décimale, le moniteur n'utilise que la numération hexadécimale, qui est ici beaucoup plus adaptée.

En notation hexadécimale, une adresse mémoire est un nombre compris entre \$0 et \$FFFF (0 et 65535 en décimal) tandis que la valeur d'une case mémoire est un nombre compris entre \$00 et \$FF (0 et 255 en décimal).

De même que l'Applesoft vous signale qu'il attend une commande en affichant un caractère J (ou > pour l'Integer Basic), Le moniteur vous signale qu'il est à votre disposition en affichant un caractère *.

Pour passer de l'Applesoft (caractère J) au moniteur (caractère *), utilisez la commande CALL-151. Pour revenir du moniteur à l'Applesoft, frappez "ctrl-C" puis "return". Ces deux procédures sont valables pour l'Integer Basic.

Toute ligne de commande doit être terminée par la frappe de la touche "return", comme en basic.

b) **Afficher le contenu de cases mémoire**

— Lorsque le caractère * est affiché, vous pouvez frapper une adresse (hexadécimale) suivie de "return". Ceci affiche le contenu de la case mémoire situé à cette adresse.

Exemple : *F800 "return"
F800- 4A

— Pour faire afficher le contenu de plusieurs adresses consécutives, tapez deux adresses séparées par un point. L'Apple affiche le contenu des cases mémoires situées de la première adresse à la seconde.

Exemple : *F800•F818 "return"
F800- 4A 08 20 47 F8 28 A9 0F
F808- 90 02 69 E0 85 2E B1 26
F810- 45 30 25 2E 51 26 91 26
F818- 60

(Ceci est le sous programme PLOT)

— Pour faire afficher le contenu des adresses suivantes, tapez un point suivi d'une adresse. L'Apple affiche le contenu des cases mémoires situées de la dernière adresse affichée jusqu'à celle tapée ici.

Exemple : Si ayant exécuté l'exemple précédent, vous désirez connaître le contenu des adresses \$F819 à \$F825, tapez •F825 puis "return"

*•F825 "return"
F819- 20 00 F8 C4 2C B0 11
F820- C8 20 0E F8 90 F6

(Ceci est le sous-programme HLIN)

— Pour faire afficher le contenu des 8 adresses suivantes, tapez simplement "return"

Exemple : Si vous tapez "return" à la suite de l'exemple précédent :

*"return"
F826- 69 01 48 20 00 F8 68 C5

— Mais il faut bien avouer que faire lister un programme sous forme de code hexadécimale n'est pas très "parlant". Il est préférable d'utiliser la commande L, qui affiche les mnémoniques de 20 instructions correspondant aux codes hexadécimaux, à partir de l'adresse qu'on lui fournit.

Exemple :

SCALL-151

*F800L

F800-	4A	LSR	
F801-	08	PHP	
F802-	20 47 FB	JSR	\$F847
F805-	28	PLP	
F806-	A9 0F	LDA	#\$0F
F808-	90 02	BCC	\$F80C
F80A-	69 E0	ADC	#\$E0
F80C-	85 2E	STA	\$2E
F80E-	B1 26	LDA	(\$26),Y
F810-	45 30	EDR	\$30
F812-	25 2E	AND	\$2E
F814-	51 26	EDR	(\$26),Y
F816-	91 26	STA	(\$26),Y
F818-	60	RTS	
F819-	20 00 FB	JSR	\$F800
F81C-	C4 2C	CPY	\$2C
F81E-	B0 11	BCS	\$F831
F820-	C8	INY	
F821-	20 0E FB	JSR	\$F80E
F824-	90 F6	BCC	\$F81C

On a affiché ici les sous-programmes PLOT et HLIN. Pour obtenir les 20 instructions suivantes, tapez simplement L puis return. Si vous désirez obtenir le listing de par exemple 60 instructions utilisez *F800LLL "return".

c) Modifier le contenu de cases mémoires

Alors que vous pouvez afficher le contenu de toutes les cases mémoires, vous ne pouvez heureusement pas modifier le contenu de toutes les cases mémoires. L'Apple comporte deux types de mémoire.

* Les ROM (Read Only Memory) sont des mémoires dont le contenu ne peut être modifié par aucun moyen, pas même une coupure de tension. Sur l'Apple les emplacements mémoires allant de \$C800 à \$FFFF sont en ROM et contiennent l'Applesoft ou

l'Integer Basic, le moniteur, et des sous-programmes destinés aux périphériques. Ce qui signifie concrètement que volontairement ou non, vous ne pouvez jamais détériorer ces programmes.

* Les RAM (Random Access Memory) sont des mémoires dont le contenu peut être modifié par les commandes du moniteur, ou la commande POKE du basic. Une coupure de tension les efface. C'est là que l'on range les programmes basic ou assembleur.

- Lorsque le caractère * est affiché, vous pouvez frapper une adresse (hexadécimale) suivie de deux points suivie d'une valeur hexadécimale (2 chiffres). Lorsque vous frappez "return", la valeur est placée à l'adresse spécifiée.

Exemple : *300:11"return"
 *300"return"
 300-11
 *

On a placé en \$300 la valeur \$11, puis on a fait afficher le contenu de \$300 pour vérifier la valeur que l'on y a mise.

- Pour modifier le contenu de plusieurs adresses consécutives, il suffit de taper la première adresse à modifier, deux points, puis la suite de valeurs. Si cette suite ne tient pas sur une ligne, frapper return, puis deux points et le reste de la suite de valeurs.

Exemple : *300:00 01 06 AA 04 FF 87 07 09 "return"
 *:0A 0B 8F E3 44 0F 10 11 34 F9 14 "return"
 *:15 C0 D4 D4 19 1A 3F 00 00 1E 1F "return"

Ceci modifie les contenus des adresses \$300 à \$31F, en y plaçant les valeurs tapées.

d) Déplacer et comparer des zones de mémoire

- Pour déplacer, plus exactement pour recopier une zone dans une autre, il faut préciser l'adresse du début de la zone destination, suivie du symbole <, puis l'adresse du début de la zone à recopier,

suivie d'un point, puis l'adresse de la fin de la zone transférée, suivie de la lettre M (M comme Move).

Exemple : *0<300*310M "return"

recopie la zone \$300 à \$310 en \$0-\$10

*1024<330*33AM "return"

recopie la zone \$330-\$33A en \$1024-\$102E

Particularité : Lorsque l'adresse du début de la zone destination est comprise entre les deux autres adresses, cela ne marche pas. En effet, l'algorithme de transfert est le suivant : on prend le premier élément de la zone source, on le place dans le premier élément de la zone destination, puis l'on prend le second élément de la zone source, on le place dans le second élément de la zone destination, etc.

Supposons que vous exécutiez la commande 301<300*304M. On prend l'élément en \$300 et on le place en \$301. Puis on prend l'élément situé en \$301 et on le place en \$302. Comme on vient de placer en \$302 ce qu'il y a en \$300, on retrouve en \$302 ce qu'il y a en \$300. Puis on prend l'élément en \$302 et on le place en \$303. On en vient de placer en \$302 le contenu de \$300... Vous avez compris, on retrouve finalement dans la zone \$300-\$305 l'élément qui se trouvait au départ en \$300 répété 6 fois en tout, alors que le contenu des éléments \$301 à \$306 a été perdu. Cela peut être utilisé pour répéter une valeur ou un groupe de valeurs.

Exemple : *300:00 FF "return"

*302<300*30FM "return"

*300*30F "return"

300 - 00 FF 00 FF 00 FF 00 FF

308 - 00 FF 00 FF 00 FF 00 FF

— Pour vérifier que deux zones sont identiques, on utilise une commande analogue.

Exemples : *0<300*305V"return"

Vérifie que le contenu de \$0 est le même que celui de \$300, puis que celui de \$1 est le même que celui de \$301 etc... Si il y a égalité, un nouveau caractère * est affiché. Sinon, on s'arrête à la première différence trouvée. On affiche alors l'adresse de la zone source où a été trouvée la différence, puis la valeur située à cette adresse, puis entre parenthèses la valeur correspondante dans la zone destination.

Exemple : *300:F0 F1 F2 F3 F4 F5 "return"

*310<300*305M "return"

*314:F6 "rreturn"

*310<300*303V

*310<300*305V

304-F4 (F6)

*

e) Sauvegarde et rechargement d'une zone mémoire sur magnétophone

— Pour sauvegarder une zone mémoire sur minicassette : Frappez l'adresse de début de la zone, puis un point, puis l'adresse de fin de zone suivie de W/. Stop. Ne pas frapper "return". Mettre le magnétophone en marche, en position enregistrement. Frapper alors return. Lorsque l'enregistrement est terminé, l'Apple émet un signal sonore (bip !) et affiche un nouveau caractère *. Vous pouvez arrêter le magnétophone.

Exemple : *300*340WV "return"

Recopie sur bande le contenu des adresses \$300 à \$340. W signifie Write (écrire)

— Pour recharger une zone mémoire : frapper l'adresse de début de zone, puis un point, puis l'adresse de fin de zone suivie de R. Mettre le magnétophone en marche, en position lecture, puis frapper la touche "return". La fin de lecture est signalée par l'apparition d'un nouveau caractère *.

Exemple : *300*340R "return"

Recharge en \$300-\$340 la zone sauvegardée sur bande.

Particularités : * Lors de l'enregistrement, le moniteur effectue la somme de tous les octets qu'il sauvegarde. Puis la somme partielle (un octet, appelé checksum) est enregistré à la suite des octets sauvegardés. Lors de la relecture, le moniteur effectue de même la somme de tous les octets qu'il lit. Puis, en fin de lecture, compare cette somme à la valeur checksum qu'il lit. Si elles sont différentes, les valeurs lues ne sont pas celles qui ont été enregistrées. Le message ERR est affiché, vous pouvez recommencer l'opération.

* Il est parfaitement possible de recharger dans une zone différente de celle où l'on avait effectué la lecture. Mais attention :

- Ces deux zones doivent avoir la même taille,
- Si ce que vous rechargez est un programme, il est très probable qu'il ne fonctionnera plus, par exemple les instructions JMP effectueront des branchements toujours à la même adresse, quel que soit l'emplacement où le programme est rechargé, ce qui est gênant dans la plupart des cas.

f) Exécuter un programme en langage machine

Il suffit de frapper l'adresse de départ du programme, suivie de G, puis de "return".

Exemple : *300G "return"

exécute un programme en langage machine situé en \$300. Le programme doit se terminer par une instruction RTS, qui rend le contrôle au moniteur, ce dernier affiche alors un caractère *.

Particularité : Un même programme peut être utilisé indifféremment depuis le moniteur ou le basic. Prenons le cas d'un programme en langage machine situé en \$300 et terminé par une instruction RTS.

(Note : \$300 vaut 768 en décimal)

	L'exécution du programme est provoquée par	l'instruction RTS provoque
depuis le moniteur	*300G"return"	un retour au moniteur qui affiche un nouveau "*"
depuis le basic en mode immédiat (Applesoft ou Integer Basic)]CALL 768 "return" >CALL 768 "return"	un retour au basic qui affiche un nouveau] ou >
depuis un programme basic: (Applesoft ou Integer Basic)	CALL 768	un retour au programme basic qui exécute la commande suivante

Il est donc tout à fait possible, et recommandé, de créer un programme machine grâce au moniteur puis de l'utiliser depuis un programme basic.

g) Les autres commandes du moniteur

*"ctrl-B" "return"

Ceci vous permet de retourner au basic. N'est jamais utilisée en pratique, car elle détruit le programme basic qui se trouvait en mémoire.

*"ctrl-C" "return"

C'est la commande que l'on utilise pour retourner au basic, elle ne détruit aucun programme, n'efface pas les valeurs des variables basic en mémoire.

*I "return"

Est équivalent à la commande INVERSE du basic

*N "return"

Est équivalent à la commande NORMAL du basic

*arg "ctrl-P" "return"

"arg" est un nombre compris entre 0 et 7. Est équivalent à la commande PR# arg du basic

* "ctrl-K" "return"

"arg" est un nombre compris entre 0 et 7. Est équivalent à la commande IN# arg du basic

Il est possible de créer sa propre commande. Chaque fois que le moniteur rencontre le caractère "ctrl-Y" dans une ligne de commande, il exécute un branchement en \$3F8. Normalement, l'utilisateur a placé en \$3F8 une instruction JMP vers un programme qui réalise sa commande. Ce programme se termine par une instruction JMP \$FF69 qui provoque un retour au moniteur.

Exemple : Si l'on veut que le caractère "ctrl-Y" provoque l'effacement de l'écran (HOME)

*300:20 58 FC 4C 69 FF "return"

*3F8:4C 00 03 "return"

On a placé les codes des instructions JSR \$FC58
JMP \$FF69 en \$300.

Le sous programme en \$FC58 efface l'écran (HOME). Puis on a placé en \$3F8 les codes d'une instruction JMP \$0300.

Si vous tapez

*"ctrl-Y" 400*410 "return"

l'écran est effacé puis le contenu des emplacements \$400 à \$410 est affiché.

Si vous tapez

*"ctrl-Y" "return"

l'écran est effacé.

Le moniteur est capable d'exécuter une addition ou une soustraction en hexadécimal.

Exemples : *1+2 "return"

=03

*FF+1 "return"

=00

*0A-5 "return"

=05

*1-2 "return"

=FF

*

Enfin, bien que cela ne soit pas d'un grand intérêt pratique, précisons qu'une ligne de commande du moniteur peut comporter plusieurs commandes séparées par des blancs

Exemple : *300 303*305 3F8:4C 00 03 "return"

300- 20

303- 4C 69 FF

*

Cette ligne de commande provoque l'affichage du contenu de \$300, puis l'affichage de la zone \$303 à \$305, puis place à partir de \$3F8 les valeurs \$4C, \$00, \$03 en mémoire.

2.3. CRÉER DES PROGRAMMES EN LANGAGE MACHINE

a) Mais où le mettre celui-là ?

Les programmes en langage machine ne fonctionnent généralement qu'en un emplacement déterminé de la mémoire. La première chose à faire est donc décider où ce programme devra être implanté.

- Si vous désirez écrire un programme uniquement en programme machine, vous pouvez l'implanter n'importe où en RAM. Toutefois, ne pas l'implanter dans les pages graphiques que le programme utilisera. On rappelle que les zones \$400-\$7FF, \$800-\$BFF, \$2000-\$3FFF et \$4000-\$5FFF correspondent à ces pages graphiques.

- Si vous désirez avoir simultanément en mémoire un programme basic et un à plusieurs programmes ou sous-programmes en langage machine, il faut s'assurer que ces divers programmes ne vont pas se recouvrir ou se détruire les uns les autres.

* Une première méthode consiste à placer le ou les programmes machines entre \$300 et \$3CF (en décimal, 768 et 975, soit 208 octets). Ces emplacements ne sont utilisés ni par le moniteur, ni par le basic, ni le DOS. Vous pouvez donc les utiliser en toute sécurité, sans risquer de détruire des données utiles au système Apple, ou de voir votre programme détruit par le basic ou le moniteur. Mais attention : veillez à ne pas dépasser ces limites, les contenus des adresses \$2FF et \$3D0 sont utiles au système.

* On peut aussi placer le ou les programmes en haut de la mémoire, c'est-à-dire les cases mémoires RAM d'adresses les plus élevées. Puis, abaisser la valeur de HIMEM afin que ces programmes en langage machine débutent à une adresse inférieure à HIMEM, pour que les chaînes créées pour le basic ne détruisent pas les programmes.

* Lorsqu'un programme est au point, on place les programmes machines juste après le programme basic : Une simple commande SAVE permet alors de sauvegarder ensemble sur cassette ou disquette le programme basic et les programmes machine. Réciproquement, une simple commande LOAD recharge tous ces programmes. Cette méthode est un peu plus compliquée, mais le résultat en vaut la peine !

Tapez le programme basic puis exécutez CALL-151 en mode immédiat pour appeler le moniteur.

Exécutez alors

```
*AF•B0 "return"
```

Le moniteur affiche alors le contenu des adresses \$AF et \$B0. \$AF et \$B0 contiennent respectivement les octets de poids faible et de poids fort de l'adresse de fin du programme basic.

Exemple : *AF•B0 "return"
00AF-82 0A

Ce qui signifie que le programme basic se termine en \$0A82.

Puis, toujours sous le moniteur, placer en mémoire les codes hexadécimaux correspondant aux programmes machine. Il est conseillé de laisser un peu de place libre entre la fin de programme basic et le début des programmes machine, il se peut en effet que des corrections sur le programme basic allongent quelque peu ce dernier par la suite. Reprenant l'exemple précédent, implanter par exemple vos programmes machine à partir de l'adresse \$B82, ce qui laisse \$100 (256 en décimal) octets libres.

Déterminez l'adresse de fin des programmes machine, ajoutez-y quelques unités (2 au moins) et placez l'adresse obtenue en \$AF-\$B0, \$69-\$6A, \$6B-\$6C et \$6D-\$6E. Voilà, vous n'avez plus qu'à frapper "ctrl-C" "return" pour revenir au basic. Le programme basic et les sous-programmes machine ne forment plus qu'un seul programme. (Voir un exemple en 2.4.a.)

b) La corvée : le codage et l'entrée du programme

Ayant écrit un programme machine, il faut le traduire en codes hexadécimaux. Utilisez pour cela la table donnée en annexe A.

Puis il faut entrer le programme sous forme de codes hexadécimaux, à l'aide du moniteur.

Exemple : Le programme suivant actionne le haut parleur de l'Apple

```
LDX  #$FF
LDY  #$0F
LDA  $C030
STX  $06
LDX  #$A0
DEX
BOUCL
BNE  BOUCL
LDX  $06
DEX
BNE  BIP
DEY
BNE  BIP
RTS
```

Puis on le code

```

300 A2 FF #FFF
302 A0 0F #0F
304 AD 30 C0 LDA $C030
307 86 06 STX $06
309 A2 A0 LDX #SA0
30B CA DEX
30C D0 FD BNE $30B
30E A6 06 LDX $06
310 CA DEX
311 D0 F1 BNE $304
313 88 DEY
314 D0 EE BNE $304
316 60 RTS

```

Puis l'on place le programme en mémoire

```

JCALL-151"return"
*300:A2 FF A0 0F AD 30 C0 86 06 A2 A0"return"
*:CA D0 FD A6 06 CA D0 F1 88 D0 EE 60"return"
*300L"return"

```

La dernière commande vous permet de vérifier que le programme a été codé correctement.

```

$CALL-151
*300L
0300- A2 FF LDX #FFF
0302- A0 0F LDY #0F
0304- AD 30 C0 LDA $C030
0307- 86 06 STX $06
0309- A2 A0 LDX #A0
030B- CA DEX
030C- D0 FD BNE $030B
030E- A6 06 LDX $06
0310- CA DEX
0311- D0 F1 BNE $0304

```

```

0313- BB DEY
0314- D0 EE BNE
0316- 60 RTS
0317- 00 BRK
0318- 00 BRK
0319- 00 BRK
031A- 00 BRK
031B- 00 BRK
031C- 00 BRK
031D- 00 BRK
*
```

Puis pour exécuter le programme

```
*300G "return"
```

2.4. UN PREMIER EXEMPLE : TEST DES FONCTIONS DE L'APPLESOFT

a) Ce programme permet le test des sous-programmes de calcul des fonctions mathématiques de l'Applesoft (sinus, cosinus, racine carrée, etc.). Le programme basic utilise un sous-programme machine très court (4 octets), placé à sa suite. Le sous-programme est appelé par la commande USR de la ligne 130.

SLIST

```
1 REM *****  
2 REM * TEST FONCTIONS APPLESOFT *  
3 REM * AUTEURS: -J.Y. ASTIER *  
4 REM * -O. KAUF *  
5 REM * COPYRIGHT (C) 1982 *  
6 REM * BY MA POMME ET MOI-MEME *  
7 REM *****  
10 POKE 10,76: POKE 11,128: POKE 12,10  
20 INPUT "ADRESSE HEXA : ";AD$  
30 IF AD$ = "" THEN END  
40 REM CONVERSION AD$=>AD  
50 AD = 0: FOR I = 1 TO LEN (AD$):C$ = MID$ (AD$,I,1)  
60 C = ASC (C$) - 48: IF C > 9 THEN C = C - 7  
70 IF C < 0 OR C > 15 THEN 90  
80 AD = 16 * AD + C: NEXT I: GOTO 100  
90 PRINT "ADRESSE INCORRECTE": GOTO 20  
100 OH = INT (AD / 256):OL = AD - 256 * OH  
110 POKE 2689,OL: POKE 2690,OH  
120 INPUT "VALEUR : ";V$: IF V$ = "" THEN 20  
130 PRINT "RESULTAT = ";USR ( VAL (V$)): GOTO 120  
$CALL-151
```

*AB0.AB3

0AB0- 20 8D EE 60

Premièrement, tapez le programme basic puis exécutez CALL-151 en mode immédiat pour appeler le moniteur.

Exécutez alors

*AF•B0"return"

Le moniteur affiche alors le contenu des octets d'adresse \$AF et \$B0, c'est-à-dire respectivement les octets de poids faible et fort de l'adresse de fin du programme basic que vous venez de taper.

]CALL-151 "return"

*AF•B0 "return"

*00AF- 43 0A

*

Ce qui signifie que le programme basic se termine en \$0A43. Il est tout à fait possible que vous n'obteniez pas exactement cette valeur, si vous n'avez pas tapé les REM vous obtiendrez un programme plus court et une adresse de fin plus petite, ou plus longue si vous avez tapé des REM ou des caractères en sus. Quoi qu'il en soit, vous devez normalement trouver un valeur inférieure à \$0A80. Nous allons donc placer en \$0A80 le sous-programme machine (4 octets) utilisé par USR.

```
]CALL-151 "return"  
*AF•B0 "return"  
00AF- 43 0A  
*A80:20 00 00 60 "return"  
*
```

Ce programme machine se termine en \$A83. Nous allons donc placer cette adresse plus 2, soit \$A85 en \$AF-\$B0, \$69-\$6A, \$6B-\$6C et \$6D-\$6E.

```
]CALL-151 "return"  
*AF•B0 "return"  
00AF- 43 0A  
*A80:20 00 00 60 "return"  
*AF:85 0A "return"  
*69:85 0A "return"  
*6B:85 0A "return"  
*6D:85 0A "return"  
* "ctrl-C" "return"  
]
```

Puis "ctrl-C" "return" vous permet de quitter le moniteur. Vous pouvez maintenant faire exécuter le programme, pour tester par exemple les fonctions sinus, située en \$EFF1 et racine carrée, située en \$EE8D.

* "ctrl-C" "return"

JRUN

ADRESSE HEXA : EFF1 "return"

VALEUR : 1.570796327 "return"

RESULTAT = 1

VALEUR : "return"

ADRESSE HEXA : EE8D "return"

VALEUR : 25 "return"

RESULTAT = 5

VALEUR : 16 "return"

RESULTAT = 4

VALEUR : "return"

ADRESSE HEXA : "return"

JCALL-151 "return"

*A80.A83 "return"

0A80-20 8D EE 60

*

On a d'abord testé $\sin(\pi/2)$, ce qui donne comme résultat 1, puis on a calculé les racines carrées de 25 et 16.

Si vous exécutez SAVE, le sous-programme machine sera sauvegardé avec le programme basic, ce qui vous évite d'avoir à le retaper à chaque utilisation.

b) Explication du fonctionnement du programme.

A la ligne 20, une chaîne de caractères représentant une valeur hexadécimale est lue, par exemple EFF1. Les lignes 40 à 100 convertissent cette chaîne de quatre caractères en deux valeurs numériques, correspondant aux octets de poids faible et fort de la valeur hexadécimale représentée par la chaîne EFF1. C'est-à-dire qu'à la ligne 100, on affecte la valeur \$EF (239 en décimal) à la variable OH tandis que la valeur \$F1 (225 en décimal) est affectée à OL.

Puis, attention, ceci est une astuce assez fréquemment utilisée, à la ligne 110 on modifie le sous-programme machine. 2689 et 2690 valent \$A81 et \$A82 en hexadécimal. A cette ligne 110, on place la valeur \$F1 en \$A81 et la valeur \$EF en \$A82.

On a donc de \$A80 à \$A83 les valeurs \$20, \$F1, \$EF et \$60, qui sont les codes du sous-programme machine suivant :

0A80	20	F1	EF	JSR	\$EFF1
0A83	60			RTS	

A la ligne 130, la commande USR place la valeur de son argument dans l'accumulateur flottant (adresses \$9D à \$A2) puis effectue un branchement à l'emplacement \$0A. On a la ligne 10, on a placé en \$0A, \$0B et \$0C (10, 11 et 12 en décimal) les codes d'une instruction JMP \$0A80. On effectue donc un nouveau branchement en \$0A80, c'est-à-dire au début de votre sous-programme machine.

Ce dernier appelle le sous-programme Applesoft situé en \$EFF1, qui calcule le sinus de la valeur contenue dans l'accumulateur flottant et place le résultat dans ce même accumulateur. Puis l'instruction RTS provoque un retour au programme basic, qui affiche alors la valeur contenue dans cet accumulateur.

Exécutons une nouvelle fois ce programme. A la ligne 20, une chaîne de caractères est lue, par exemple EE8D. A la ligne 100, OH contient la valeur \$EE (238 en décimal) tandis que OL contient \$8D (141 en décimal). A la ligne 110, on modifie à nouveau le sous-programme qui devient donc

0A80	20	8D	EE	JSR	\$EE8D
0A83	60			RTS	

A la ligne 130, la commande USR place la valeur de son argument dans l'accumulateur flottant puis se branche en \$0A, puis en \$0A80. On appelle le sous-programme situé en \$EED8, qui calcule la racine carrée de la valeur contenue dans l'accumulateur. RTS provoque un retour au programme basic qui affiche le résultat.

Si maintenant vous cessez l'exécution, appelez le moniteur et examinez le contenu des adresses \$A80 à \$A83.

JCALL-151 "return"

*A80.A83 "return"

0A80-20 8D EE 60

*

Voilà ce qu'est devenu le sous-programme, on retrouve bien en \$A81 et \$A82 les valeurs qui y ont été placées lors de la dernière exécution.

2.5.. UN SECOND EXEMPLE : L'APPLE EN MUSIQUE

a) Voici un second exemple de programme basic utilisant un sous-programme machine. Dans l'exemple précédent, le sous-programme était placé à la suite du programme basic, et était sauvegardé avec lui par une commande SAVE. Ici, le sous-programme sera placé dans la zone inutilisée \$300-\$3CF. Une commande SAVE ne permet donc pas de sauvegarder le sous-programme. On utilise donc une seconde astuce : Plutôt que de retaper les codes correspondant au sous-programme à chaque utilisation, c'est le programme basic qui placera lui-même ces codes en mémoire grâce à des commandes POKE.

SLIST

```

10 REM  IMPLANT SOUS PROGRAMME
20 DATA 184,160,0,173,48,192,166,26,202
30 DATA 234,208,252,198,24,208,243,196
40 DATA 25,240,4,198,25,80,235,96
50 FOR I = 816 TO 840: READ J: POKE I,J: NEXT
60 LG2 := - LOG (2):K = 2000:L = 120
90 RESTORE : FOR I = 1 TO 25: READ V: NEXT
140 REM  INTERPRETATION DE L'AIR
150 READ DUREE: IF DUREE = 0 THEN END
160 READ NUTE$:OCTAVE = VAL ( RIGHT$ (NUTE$,1))
170 NUTE$ = LEFT$ (NUTE$, LEN (NUTE$) - 1)
180 IF NUTE$ = "LA" THEN N = 1
190 IF NUTE$ = "LA#" THEN N = 2
200 IF NUTE$ = "SI" THEN N = 3
210 IF NUTE$ = "DO" THEN N = 4
220 IF NUTE$ = "DO#" THEN N = 5
230 IF NUTE$ = "RE" THEN N = 6
240 IF NUTE$ = "RE#" THEN N = 7
250 IF NUTE$ = "MI" THEN N = 8
260 IF NUTE$ = "FA" THEN N = 9
270 IF NUTE$ = "FA#" THEN N = 10
280 IF NUTE$ = "SOL" THEN N = 11
290 IF NUTE$ = "SOL#" THEN N = 12

```

```

300 T = LG2 * (OCTAVE + (N / 12))
305 T = INT (K * EXP (T)): POKE 26,T
310 DUREE = INT ((L * DUREE) / T):T = INT (DUREE / 256)
320 POKE 25,T:T = DUREE - 256 * T: POKE 24,T: CALL 816
330 GOTO 150
510 DATA 60,"SOL3",60,"SOL3",60,"SOL3"
520 DATA 60,"LA4",120,"SI4",120,"SI4"
530 DATA 60,"SOL3",60,"SI4",60,"LA4"
540 DATA 60,"LA4",180,"SOL3",60,"SOL3"
550 DATA 60,"SOL3",60,"SOL3",60,"LA4"
560 DATA 120,"SI4",120,"LA4",60,"SOL3"
570 DATA 60,"SI4",60,"LA4",60,"LA4"
580 DATA 180,"SOL3",60,"LA4",60,"LA4"
590 DATA 60,"LA4",60,"LA4",120,"MI3"
600 DATA 120,"MI3",60,"LA4",60,"SOL3"
610 DATA 60,"FA#3",60,"MI3",180,"RE3"
620 DATA 60,"SOL3",60,"SOL3",60,"SOL3"
630 DATA 60,"LA4",120,"SI4",120,"LA4"
640 DATA 60,"SOL3",60,"SI4",60,"LA4"
650 DATA 60,"LA4",400,"SOL3",0

```

Pour utiliser ce programme, le taper, puis lancer l'exécution par RUN.

Pour avoir un autre air, changer les lignes 510 à 650. Chaque note est constituée d'un nombre qui représente sa durée (60, 120, 180 ou 400 dans cet exemple) et d'une chaîne qui représente sa hauteur.

L'ordre des notes est

LA2, LA#2, SI2, DO2, DO#2, RE2, RE#2, MI2, FA2, FA#2, SOL2, SOL#2, LA3, LA#3, ... etc.

b) Explication du fonctionnement

Chaque note est produite par le sous-programme suivant

2.6. LA CARTE MEMOIRE DU SYSTEME APPLE II

a) Organisation générale

Adresses		Utilisation
hexa	deci	
\$0000	0	Mémoire RAM Pages graphiques programmes basic programme assembleur DOS
\$BFFF	49151	
\$C000	49152	Adresses réservées aux E/S Il n'y a pas de mémoire ici. C'est la référence à l'adresse qui produit une action. Exemple : haut parleur, com- muter pages graphiques, etc.
\$C7FF	51239	
\$C800	51240	Mémoire ROM pour E/S Contient des programmes utiles aux périphériques.
\$CFFF	53247	
\$D000	53248	Mémoire ROM Contient l'interpréteur basic et le moniteur (\$F800-\$FFFF).
\$FFFF	65535	

1K = 1024

330	B8	CLV		
331	A0	LDY	#00	
333	AD	LDA	C0	\$C030
336	A6	LDX	1A	\$1A
338	CA	DEX		
339	EA	NOP		
33A	D0	BNE	FC	\$0338
33C	C6	DEC	18	\$18
33E	D0	BNE	F3	\$0333
340	C4	CPY	19	\$19
342	F0	BEQ	03	\$0348
344	C6	DEC	19	\$19
346	50	BVC	EB	\$0333
348	60	RTS		

La membrane du haut parleur de l'Apple peut avoir deux positions. A chaque fois que l'on exécute LDA \$C030, la membrane change de position. La valeur chargée dans l'accumulateur n'a aucun intérêt ni aucune signification, c'est le simple fait de lire en \$C030 qui provoque le déplacement.

Pour émettre une note, on place en \$18-\$19 (24 et 25 en décimal) le nombre de déplacements de la membrane à effectuer, ce qui détermine la durée de la note. On place en \$1A (26 en décimal) le temps qu'il faut attendre entre deux déplacements, ceci détermine la fréquence, en d'autres termes la hauteur de la note, puis on effectue la boucle \$338-\$33A afin de perdre un temps entre deux déplacements, puis décrémente d'une unité le nombre de déplacements (\$33C à \$344). Si ce nombre est nul, on retourne au programme basic par RTS, sinon on se branche en \$0333 pour un nouveau déplacement.

Pour émettre une note, le programme basic place donc le délai d'attente entre deux déplacements grâce à la commande POKE 26, T de la ligne 305, place le nombre de déplacements grâce aux commandes POKE 25,T et POKE 24,T de la ligne 320, puis appelle le sous-programme par CALL 816 à la ligne 320. (816 vaut \$330).

b) Organisation et gestion de la mémoire RAM en Applesoft

L'Applesoft, comme l'Integer Basic, est un interpréteur basic. Lorsque vous chargez un programme basic, ou lorsque vous avez

juste fini de le taper, il n'y a en mémoire que le programme. Puis vous faites exécuter le programme. A chaque fois qu'une variable qui n'a jamais été utilisée est rencontrée, l'interpréteur cherche une place libre en mémoire pour ranger la valeur affectée à cette variable (7 octets par variable).

De même, à chaque fois qu'une commande DIM est rencontrée, l'interpréteur cherche de la place en mémoire pour le tableau, à chaque fois qu'une chaîne est affectée à une variable chaîne, de l'espace mémoire est "consommé" pour ranger cette nouvelle chaîne.

L'Applesoft utilise quatre zones mémoire, qui sont toujours rangées dans le même ordre.

- La première zone débute normalement en \$801 et contient le programme basic
- La seconde contient les variables simples et les adresses des chaînes
- Puis vient la zone des tableaux
- Enfin, en haut de la mémoire RAM (adresses hautes) se trouvent les chaînes.

A tout instant, l'Applesoft conserve et met à jour les adresses de début et de fin de ces quatre zones, dans les emplacements mémoire suivants :

Adresses mémoire hexadécimal	Adresses mémoire décimal	Désignation	Contenu
\$67-\$68	103-104	P1	Contient l'adresse de début du programme basic. C'est-à-dire \$801 (2049) si vous ne l'avez pas modifié.
\$AF-\$B0	175-176	P2	Contient l'adresse de fin du programme basic.
\$69-\$6A	105-106	LOMEM	Contient l'adresse de début de la zone réservée aux variables simples et aux adresses de chaînes.
\$6B-\$6C	107-108	T1	Contient l'adresse de début de la zone réservée aux tableaux. Cette adresse correspond aussi à la fin de la zone réservée aux variables simples et adresses de chaînes.
\$6D-\$6E	109-110	T2	Contient l'adresse de fin de zone réservée aux tableaux.
\$6F-\$70	111-112	C2	Contient l'adresse de fin de zone réservée aux chaînes.
\$73-\$74	115-116	HIMEM	Contient l'adresse de début de zone réservée aux chaînes.

On a toujours

$$P1 < P2 \leq LOMEM \leq T1 \leq T2 \leq C2 \leq HINEM$$

Exemple : Voici les valeurs de ces adresses, pour un système équipé de 48K de mémoire RAM, d'un DOS, lorsque l'on charge un programme particulier.

	P1	P2	LOMEM	T1	T2	C2	HIMEM
avec DOS sans programme basic	2049 \$801	2052 \$804	2052 \$804	2052 \$804	2052 \$804	38400 \$9600	38400 \$9600
avec DOS programme basic chargé. Avant exécution	2049 \$801	2455 \$997	2455 \$997	2455 \$997	2455 \$997	38400 \$9600	38400 \$9600
avec DOS programme basic chargé. Après exécution	2049 \$801	2455 \$997	2455 \$997	2511 \$9CF	2511 \$9CF	38394 \$95FA	38400 \$9600

Avant le chargement du programme on constate que $P2 = P1 + 3$, la zone programme est bien vide, $LOMEM = T1$, $T1 = T2$ et $C2 = HIMEM$ ce qui signifie que les trois autres zones sont elles aussi vides.

Après le chargement du programme et avant son exécution • P2 est devenu nettement supérieur à P1, ce qui indique la présence du programme. On a toujours $LOMEM = T1$, $T1 = T2$ et $C2 = HIMEM$, les trois autres zones sont toujours vides. Après exécution du programme, on a $LOMEM < T1$: des variables ont donc été utilisées. $T1 = T2$ prouve qu'aucun tableau n'a été créé. $C2 < HIMEM$ permet d'affirmer que le programme utilise des variables chaînes, des chaînes ayant été créées.

Etat de la mémoire d'un système possédant 48K de RAM avant le chargement d'un programme basic

\$0000	0	Page zéro
\$00FF	255	
\$0100	256	Pile du 6502
\$01FF	511	
\$0200	512	Utilisé par GETLN, sous programme de lecture d'une ligne
\$02FF	767	
\$0300	768	\$300 à \$3CF inutilisés. Le reste =
\$03FF	1023	adresses système
\$0400	1024	Page 1 graphique basse résolution
\$07FF	2047	
\$0800	2048	Page 2 graphique basse résolution
\$0BFF	3071	
\$0C00	3072	Rien
\$1FFF	8191	
\$2000	8192	Page 1 graphique haute résolution
\$3FFF	16383	
\$4000	16384	Page 2 graphique haute résolution
\$5FFF	24575	
\$6000	24576	Rien
\$95FF	38399	
\$9600	338400	D.O.S.
\$BFFF	49151	

← P1
← P2, LOMEM, T1, T2

← C2, HIMEM

Etat de la mémoire d'un système possédant 48K de RAM après le chargement et l'exécution d'un programme basic, dans le cas où l'utilisateur n'a modifié aucune des valeurs P1, P2, LOMEM, T1, T2, C2 et HIMEM.

\$0000	0	Page zéro	
\$00FF	255		
\$0100	256	Pite du 6502	
\$01FF	511		
\$0200	512	Utilisée par GETLN	
\$02FF	767		
\$0300	768		
\$03FF	1023		
\$0400	1024	Page 1 graphique basse résolution	
\$07FF	2047		← P1 (\$801)
\$0800	2048	Programme basic	
Variables simples et adresses des chaînes affectées aux variables chaînes simples			← P2, LOMEM
Tableaux			← T1
Rien			← T2
\$1FFF	8191		
\$2000	8192	Page 1 graphique haute résolution	
\$3FFF	16388		
\$4000	16384	Page 2 graphique haute résolution	
\$5FFF	24575		
\$6000	24576	Rien	
\$95FF	38399	Chaînes	← C2
\$9600	38400		← HIMEM (\$9600)
\$BFFF	49151	D.O.S.	

On remarque que l'Applesoft place le programme basic dans la page 2 basse résolution, ce qui la rend donc inutilisable.

c) Format des variables en mémoire

L'Applesoft ne garde en mémoire que les deux premiers caractères de chaque nom de variable simple ou de tableau. Chacun de ces caractères est représenté par un code Ascii (1 octet). Par exemple pour la variable réelle ABC, on ne conserve en mémoire que les codes Ascii de "A" et de "B", c'est-à-dire \$41 et \$42.

Mais, afin de pouvoir par exemple distinguer la variable réelle AB de la variable entière AB\$, on emploie deux codes Ascii différents.

Voici ces deux codes

\$30	\$31	\$32	\$33	\$34	\$35	\$36	\$37	\$38	\$39	\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47	\$48
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	
\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F	\$50	\$51	\$52	\$53	\$54	\$55	\$56	\$57	\$58	\$59	\$5A	
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	

code positif

\$B0	\$B1	\$B2	\$B3	\$B4	\$B5	\$B6	\$B7	\$B8	\$B9	\$C0	\$C1	\$C2	\$C3	\$C4	\$C5	\$C6	\$C7	\$C8
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	
\$C9	\$CA	\$CB	\$CC	\$CD	\$CE	\$CF	\$D0	\$D1	\$D2	\$D3	\$D4	\$D5	\$D6	\$D7	\$D8	\$D9	\$DA	
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	

code négatif

Le premier code est appelé code positif car tous les octets le composant sont positifs, c'est-à-dire ont le bit 7 nul.

Le second code est lui appelé code négatif car tous les octets le composant sont négatifs, ont le bit 7 à 1. Pour un caractère donné, on passe d'un code à l'octet en changeant l'état du bit 7, c'est-à-dire en ajoutant ou retranchant \$80. "0" à \$30 pour code positif et \$30 + \$80 = \$B0 pour code négatif.

Pour une variable simple ou un tableau réel, les deux premiers caractères du nom sont codés en code positif.

Exemples : ABC est codé par \$41 et \$42
Z1 est codé par \$5A et \$31.

Pour une variable simple ou un tableau entier, les deux premiers caractères du nom sont codés en code négatif

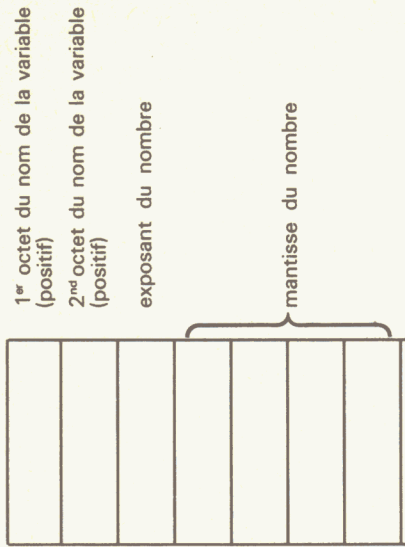
Exemples : VAR% est codé par \$D6 et \$C1
X2% est codé par \$D8 et \$B2

Pour une variable simple ou un tableau chaîne, le premier caractère du nom a un code négatif alors que le second a un code positif

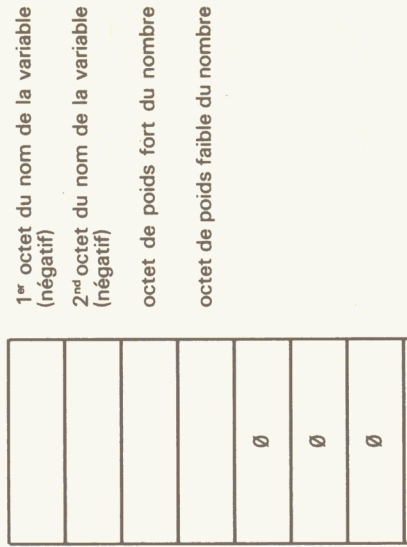
Exemples : MOT\$ est codé par \$CD et \$4F
V1234\$ est codé par \$D6 et \$31

Les variables simples, réelles entières et chaînes, occupent chacune 7 octets de mémoire et sont placées en mémoire à partir de LOMEM, au fur et à mesure qu'on les rencontre lors de l'exécution. T1 est l'adresse de la fin de cette zone.

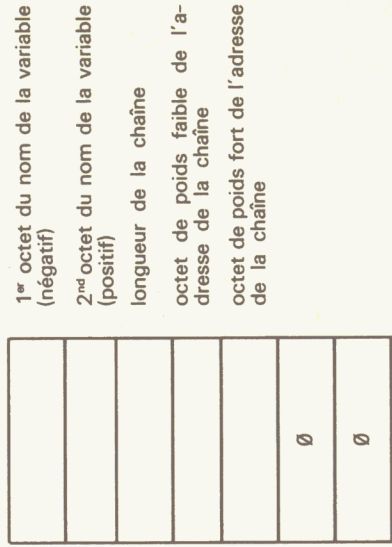
— les variables simples réelles.
Un nombre réel est représenté sur 5 octets



— les variables simples entières.
Un nombre entier est représenté sur deux octets



— les variables simples chaîne
Comme une chaîne est de taille variable, on place ici l'adresse de la chaîne, qui se trouve en haut de la mémoire, entre HIMEM et C2



2.7. DES ALLIÉS PUISSANTS : LES SOUS PROGRAMMES DU MONITEUR ET DE L'APPLESOFT

a) Le moniteur et l'Applesoft représentent 12 K octets de programmes en langage machine. Ces programmes sont pratiquement tous des sous-programmes, ce qui signifie qu'ils sont utilisables par un programme utilisateur. Ils possèdent les deux caractéristiques rares suivantes :

- ils existent
 - et ils fonctionnent parfaitement !
- Il serait donc vraiment dommage de ne pas les utiliser.

D'un point de vue pratique, on peut distinguer trois catégories parmi ces sous-programmes

- Ceux qui ne nécessitent aucun paramètre d'entrée. Une simple commande CALL en basic ou une simple instruction JSR en langage machine suffit pour les utiliser.

Exemple : CALL-936 ou CALL 64600 ou JSR \$FC58 efface l'écran (HOME)

CALL-1184 ou CALL 64352 ou JSR \$FB60 efface l'écran et affiche le titre APPLE

CALL-1059 ou CALL 64477 ou JSR \$FBDD produit un bip !

- Ceux dont les paramètres d'entrée doivent être placés dans des emplacements mémoire. En basic, on place les paramètres en mémoire grâce à des commandes POKE, puis on appelle le sous programme par CALL

Exemple : POKE 230,32:CALL 62450

On place la valeur \$20 et \$E6 puis on appelle HCLR, ce qui efface la page 1 haute résolution.

POKE 230,64:CALL 62450

Les tableaux, réels, entiers ou chaînes, occupent une place mémoire qui dépend de leur nombre de dimensions et de la taille de ces dimensions, c'est-à-dire de leur nombre d'éléments. L'adresse du premier tableau est T1. Pour trouver l'adresse de début du tableau suivant, ajouter la quantité "déplacement" à T1, etc.

Tableau réel	Tableau entier	Tableau chaîne	
positif	positif	positif	1 ^{er} octet du nom
positif	positif	positif	2 nd octet du nom
			octet de poids faible déplacement
			octet de poids fort déplacement
			nombre de dimensions
			poids fort taille première dimension
			poids faible taille première dimension
.....	poids fort taille dernière dimension
			poids faible taille dernière dimension
exposant	octet de poids fort	longueur de la chaîne	élément (0, 0, ..., 0)
mantisse	octet de poids faible	poids faible adresse chaîne	
mantisse		poids fort adresse chaîne	
mantisse	
.....			
exposant	octet de poids fort	longueur de la chaîne	élément (N, N, ..., N) (dernier élément)
mantisse	octet de poids faible	poids faible adresse chaîne	
mantisse		poids fort adresse chaîne	
mantisse			

On place la valeur \$40 en \$E6 puis on appelle HCLR, ce qui efface la page 2 haute résolution.

- Enfin, ceux dont les paramètres d'entrée doivent être placés dans les registres du microprocesseur. En basic, il faut appeler un petit sous programme en langage machine qui charge les registres puis appelle à son tour le sous programme désiré.

Exemple: Si vous désirez utiliser BKGND, pour effacer la page courante haute résolution en lui donnant la couleur rouge, il faut placer le code de la couleur, soit \$AA dans l'accumulateur, puis appeler BKGND. En basic, ceci peut se faire de la façon suivante :

```
10 POKE 768,169:POKE 769,170:POKE 770,32
20 POKE 771,244:POKE 772,243:POKE 773,96
30 CALL 768
```

Les lignes 10 et 20 placent en \$300 le petit sous programme suivant

```
300 A9 AA LDA #AA
302 20 F4 F3 JSR $F3F4
305 60 RTS
```

A la ligne 30, on appelle ce sous programme qui charge le code de la couleur dans l'accumulateur puis appelle à son tour BKGND.

Si maintenant vous désirez réeffacer l'écran en lui donnant la couleur blanche, il suffit d'exécuter la ligne.

```
100 POKE 769,255:CALL 768
```

En effet, la commande POKE modifie le sous programme machine qui devient

```
300 A9 FF LDA #FF
302 20 F4 F3 JSR $F3F4
305 60 RTS
```

Le code \$FF (255 en décimal) est celui du blanc, la commande CALL 768 permet donc d'obtenir une page blanche.

b) Quelques sous programmes du moniteur

Pour utiliser ces sous programmes vous devez initialiser les paramètres d'entrée, puis exécuter une commande CALL en basic ou JSR en langage machine à l'adresse de début du sous programme.

Notations: A désigne le registre accumulateur du microprocesseur, tandis que X et Y désignent respectivement les registres d'index X et Y.

\$F800 PLOT Trace en rectangle élémentaire dans la page
61568 1 basse résolution
-3968 * en entrée - A contient le numéro de ligne (0-47)
- Y contient le numéro de colonne (0-39)
- \$30 contient le code de la couleur, normalement initialisé par la commande COLOR basic, ou par SETCOL.

* en sortie - un rectangle est tracé
- A est modifié
- X, Y et \$30 inchangés

\$F819 HLINE Trace une ligne horizontale dans la page 1
63513 basse résolution

* en entrée - A contient le numéro de ligne (0-47)
- Y contient l'abscisse de début de la ligne (0-39)
- \$2C contient l'abscisse de fin de la ligne (0-39)

Le contenu de Y doit être inférieur à celui de \$2C
- \$30 contient le code de couleur, normalement initialisé par COLOR en basic ou SETCOL

* en sortie - une ligne horizontale est tracée
- A et Y sont modifiés
- X, \$2C et \$30 sont inchangés

\$F828 VLINE Trace une ligne verticale dans la page 1 basse
63528 résolution
-2008 * en entrée - Y contient le numéro de colonne (Ø-39)
- A contient l'ordonnée de début de la ligne (Ø-47)
- \$2D contient l'ordonnée de fin de la ligne (Ø-47) le contenu de A doit être inférieur à celui de \$2D
- \$3Ø contient le code de couleur, normalement initialisé par COLOR ou SET-COL.

* en sortie - une ligne verticale est tracée
- A est modifié
- X, Y, \$2D et \$3Ø sont inchangés.

\$F832 CLRSCR Efface la page 1 basse résolution
63538
-1998 * pas de paramètre d'entrée.
* en sortie, A, X et Y sont modifiés. \$3Ø (COLOR) est nul.

\$F836 CLRTOP Efface les 4Ø premières lignes de la page
63542 1 basse résolution
-1994 * pas de paramètre d'entrée

* en sortie, A, X et Y sont modifiés. \$3Ø (COLOR) est nul.

\$F847 GBASCALC Calcul de l'adresse en page 1 du 1^{er} octet
63559 d'une ligne
-1977 * en entrée A contient un numéro de ligne (Ø-23)

* en sortie - \$26-\$27 contient l'adresse mémoire du 1^{er} octet de cette ligne
- A est modifié
- X et Y inchangés.

\$F85F NEXTCOL Incrémente de 3 la valeur de la couleur
63583 basse résolution
-1953 * pas de paramètre d'entrée

* en sortie - A et \$3Ø contiennent le code de la nouvelle couleur
- X et Y inchangés

\$F864 SETCOL Définit la couleur basse résolution
63588
-1948 * en entrée - A contient le numéro de la couleur désirée

* en sortie - A et \$3Ø contiennent le code de cette couleur
- X et Y inchangés.

\$F871 SCRNDÉTERMINE la couleur d'un rectangle élémentaire
63601 de la page 1 basse résolution
-1935 * en entrée - A contient le numéro de ligne (Ø-47)
- Y contient le numéro de colonne (Ø-39)

* en sortie - A contient le numéro de la couleur de ce point (Ø-15)
- X et Y inchangés

\$F94Ø PRNTYX Affiche sous la forme d'un nombre de 4
63808 chiffres hexadécimaux le contenu des registres Y et X.
-1728 Les deux premiers chiffres représentent la valeur de Y, les deux seconds celle de X.

\$F941 PRNTAX Idem au précédent, en utilisant A à la place
 63809 de Y
 -1727

\$F944 PRNTX Affiche X sous la forme d'un nombre de 2
 63812 chiffres hexadécimaux
 -1724 * Pour ces trois sous programmes X et Y sont inchangés
 en sortie alors que A contient la dernière valeur
 affichée.

\$FAD7 REGDSP Affiche le contenu des registres qui avaient
 64215 été placés de \$45 à \$49 par IOSAVE. En sortie, le
 -1321 contenu de A, X et Y est modifié.

\$FB1E PREAD Lecture de la position d'un potentiomètre de
 64286 jeu.
 -1250 * en entrée, X contient le numéro du potentiomètre
 (0-3)

* en sortie, Y contient un nombre compris entre 0 et
 255 qui indique la position lue. A est modifié, X ne
 l'est pas.

Attention : le contenu de X doit être compris entre 0 et 3. Sinon,
 le moniteur lit une adresse réservée qui n'est pas celle des manettes,
 ce qui peut par exemple commuter un mode graphique, ou provoquer
 d'autres phénomènes inattendus.

\$FB60 APPLE II Efface l'écran et affiche le titre APPLE
 64352 II. Modifie A et Y.
 -1184

\$FBDD BELL1 Produit un bip ! Modifie A et Y. X inchangé
 64577
 -959

\$FCS8 HOME Efface la fenêtre de l'écran
 64600
 -936 * en entrée - \$20, \$21, \$22 et \$23 définissent la fenêtre
 (Voir page 10)
 - \$24 et \$25 contiennent la position du
 prochain caractère.

* en sortie - la fenêtre est effacée.
 - \$20, \$21, \$22 et \$23 ne sont pas modifiés.
 - \$24 est nul, \$25 contient la même valeur
 que \$22
 - les registres sont modifiés

\$FD0C RDKEY Lecture d'un caractère. Place un curseur
 64780 clignotant sur l'écran puis effectue un branchement au
 -756 sous programme de lecture dont l'adresse est contenue
 en \$38 et \$39. Ce sous programme est habituellement
 KEYIN, si l'on n'a pas de DOS.

\$FD1B KEYIN Lecture d'un caractère du clavier.. Attend
 64795 qu'une touche du clavier soit frappée. Dès que cela se
 -741 produit, le curseur clignotant est retiré et l'on retourne
 au programme appelant avec le code du caractère lu
 dans l'accumulateur.

Note : Pendant l'attente, on incrémente le contenu des adresses \$4E-
 \$4F. Ce qui permet de disposer en \$4E-\$4F d'un nombre aléatoire
 sur deux octets.

\$FD35 RDCHAR Appelle RDKEY. Puis, si le code lu est
 64863 celui de la touche "escape", les touches I, J, M et K
 -673 serviront par la suite à déplacer le curseur.

\$FD6A GETLN Lecture d'une ligne
 64874
 -662 * en entrée - \$33 contient un code écran de caractère

* en sortie - \$33 est inchangée

- lorsque l'utilisateur frappe "return", retour au programme appellant. Les codes de la ligne lue se trouvent en mémoire à partir de l'adresse \$200. Le dernier code est celui de la touche "return".

Le sous programme agit de la façon suivante :

- Tout d'abord, il affiche le caractère dont le code est en \$33. Par exemple, une * pour le moniteur, un > pour l'Integer basic, un J pour l'Applesoft ou un ? pour une instruction INPUT. En effet, tous ces programmes utilisent GETLN.

- Chaque fois que vous frappez une touche, le code de cette touche est placé dans le buffer d'entrée (emplacements mémoire \$200 à \$2FF), et le caractère frappé est affiché sur l'écran si possible (par exemple A est affiché, ctrl-A ne l'est pas). Le code du premier caractère frappé est placé en \$200, le second en \$201, etc... Il y a néanmoins quelques exceptions à cette règle :

- Lorsque vous frappez ctrl-X, tout ce que vous avez tapé est annulé. GETLN affiche un backslash (\) à la fin de la ligne, passe à la ligne suivante, et on recommence tout, on affiche le caractère dont le code est en \$33, etc...

- Lorsque vous frappez la touche ←, le curseur est déplacé vers la gauche et le dernier caractère placé dans le buffer d'entrée (\$200-\$2FF) en est éliminé.

- Lorsque vous frappez la touche →, le curseur est déplacé vers la droite et le caractère sur lequel se trouvait le curseur est placé dans le buffer d'entrée.

- Lorsque vous frappez la touche "escape", le curseur n'est pas placé, aucun code n'est ajouté ou retranché au buffer d'entrée. Mais cela modifie le rôle des touches I, J, M et K qui permettent alors de déplacer le curseur sur l'écran sans modifier le contenu du buffer d'entrée.

- Enfin, lorsque vous frappez "return", le code de cette touche

est placé dans le buffer d'entrée, à la suite des caractères qui s'y trouvaient, la fin de la ligne de l'écran est effacée, un passage à la ligne suivante est effectué sur l'écran. On retourne alors au programme appellant, les codes des caractères lus débutent à l'adresse \$200 et sont suivis d'un code "return" (\$8D)

Encore un mot et vous saurez tout sur GETLN : Il ne peut traiter des lignes de plus de 255 caractères. Aussi, dès le 249^e caractère, il émet un bip ! à chaque fois que vous frappez une touche, pour vous avertir qu'il serait temps de frapper "return" !

\$FD67 GETLNZ Provoque un passage à la ligne puis exécute
64871 GETLN
-665

\$FD6F GETLN1 Identique à GETLN, sauf que le caractère
64879 dont le code est en \$33 n'est pas affiché. Néanmoins,
-657 si vous annulez la ligne par "ctrl-X", alors le caractère
dont le code est en \$33 sera affiché au début de la
nouvelle ligne. Que voulez-vous, nul n'est parfait...

\$FDDA PRBYTE Affiche la valeur contenue dans l'accumula-
64986 teur sous forme de deux chiffres hexadécimaux. En
-550 sortie, l'accumulateur est modifié. X et Y ne sont pas
changés

\$FDE3 PRHEX Affiche les quatre bits de poids faible de la
64995 valeur contenue dans l'accumulateur sous forme d'un
-541 chiffre hexadécimal. L'accumulateur est modifié, mais
pas X et Y.

\$FDED COUT Saut au sous programme de sortie de caractère
65005 dont l'adresse est en \$36-\$37. Cette adresse est nor-
-531 malement \$FDF0 (COUT 1), si l'on a pas de D.O.S.

\$FDF0 COUT1 Affiche le caractère dont le code étant dans
65008 A. Ne modifie pas les registres
-528

Note : A chaque fois que le code vaut \$8D ("return"), COUT1 lit le clavier. Si un "ctrl S" a été frappé, COUT1 se bloque tant qu'on ne frappe pas un nouveau caractère. Ceci permet, par exemple d'interrompre un listing. Si vous frappez un caractère quelconque, COUT1 se débloque et provoque le saut à la ligne que le code \$8D provoque. Si vous frappez un "ctrl-C", ce dernier est conservé afin de pouvoir être utilisé par la suite (par exemple pour interrompre un programme (basic) puis COUT1 se débloque et exécute le saut de ligne.

\$FE80 SETINV Etablit le mode vidéo inverse pour COUT1, 65152 c'est-à-dire que les prochains caractères affichés seront -384 noirs sur fond blanc. En sortie, le registre Y et \$32 contiennent la valeur \$3F, A et X ne sont pas affectés.

\$FE84 SETNORM Etablit le mode vidéo normal pour 65156 COUT1, c'est-à-dire que les prochains caractères affichés -380 seront blancs sur fond noir. En sortie, le registre Y et \$32 contiennent la valeur \$FF, A et X ne sont pas affectés.

\$FF4A IOSAVE Sauvegarder les registres.

65354 -182

* Les valeurs de A, X, Y, P et S en entrée du sous programme sont placées respectivement en \$45, \$46, \$47, \$48 et \$49.

* en sortie A et X sont changés, le bit de mode décimal du 6502 est nul

\$FF3F IOREST Restaurer les registres. Opération inverse de 65343 IOSAVE. -193

c) Les sous programmes de calcul réel de l'Applesoft

En Applesoft, un nombre réel est représenté par cinq octets, un pour l'exposant et quatre pour la mantisse. Les réels sont stockés

en mémoire comme il a été expliqué au chapitre 2.6. Les sous programmes de calcul, de même que les instructions du 6502, délivrent leurs résultats dans un endroit privilégié. Dans le cas du 6502, le résultat d'une addition, soustraction, etc... se trouve toujours dans le registre A (accumulateur). Pour ce jeu de sous programmes, le résultat se trouve dans les emplacements mémoires \$9D à \$A2. Par analogie, on nomme ces octets "Accumulateur flottant", ou en abrégé, FAC. Tous les sous programmes qui n'utilisent qu'un seul argument (calcul de sinus, cosinus...) prennent pour argument le nombre qu'ils trouvent en FAC et placent le résultat dans ce même FAC.

Il existe une seconde zone mémoire privilégiée, constituée des octets d'adresse \$A5 à \$AA qui doit contenir le second argument, dans le cas des sous programmes à deux arguments (addition, multiplication...). Cette zone est appelée "Argument", ou en abrégé ARG.

Par exemple, pour multiplier deux réels, il faut placer l'un d'eux dans FAC (emplacement mémoires \$9D à \$A2), l'autre dans ARG (\$A5 à \$AA), puis appeler le sous programme de multiplication. Le contenu de ARG n'est pas modifié alors que FAC contient le résultat.

Il existe de plus trois zones \$93 à \$97, \$98 à \$9C et \$8A à \$8E, que l'on nomme TEMP1, TEMP2 et TEMP3 respectivement qui peuvent être utilisées pour stocker des résultats intermédiaires.

Vous avez peut-être remarqué que FAC et ARG sont constitués de 6 octets alors qu'un réel ne comporte que 5 octets. Voir SGN

\$E10C AYINT Convertit le nombre situé en FAC en un 57612 nombre entier à deux octets. Pour convertir un réel en -7924 entier :

- placer le réel dans FAC
- exécuter JSR \$E10C
- le résultat se trouve en \$A0 (octet de poids fort) et \$A1 (octet de poids faible)

\$E2F2 GIVAYF Convertit un entier à deux octets en réel
 58092 - placez l'octet de poids fort dans A (instruction LDA)
 -7438 et l'octet de poids faible dans Y (instruction LDY)
 - exécuter JSR \$E2F2
 - le résultat est dans FAC

\$E7AA FSUBT Calcule ARG-FAC et place le résultat dans
 59306 FAC. ARG n'est pas affecté
 -6930

\$E7A7 FSUB Place le nombre dont l'adresse mémoire est dans
 59303 les registres Y et A dans ARG puis exécute FSUBT
 -6233

Il faut placer la partie basse de l'adresse mémoire du premier octet du nombre dans Y et la partie haute de cette adresse dans A (voir CONUPK)

\$E7C1 FADDT Calcule ARG + FAC et place le résultat dans
 59329 FAC. ARG n'est pas affecté
 -6207

\$E7BE FADD Place le nombre dont l'adresse mémoire est
 59326 dans les registres Y et A dans ARG puis exécute
 -6210 FADDT (voir CONUPK)

\$E941 LOG Calcule le logarithme népérien de FAC et place
 59713 le résultat dans FAC
 -5823

\$E982 FMULTT Calcule ARG × FAC et place le résultat
 59778 dans FAC. ARG n'est pas affecté
 -5758

\$E97F FMULT Place le nombre dont l'adresse mémoire est
 59775 dans les registres Y et A dans ARG puis exécute
 -5761 FMULTT (voir CONUPK)

\$E9E3 CONUPK C'est lui qui place le nombre dont l'adresse
 59699 mémoire est dans les registres Y et A dans ARG.
 -5837

Exemple : Supposons que vous désiriez placer dans ARG le nombre rangé dans les cases mémoires \$0A84 à \$0A88.

Il faut faire : LDY # \$0A
 LDA # \$84
 JSR \$E9E3

\$EA39 MUL10 Calcule $10 \times \text{FAC}$ et range le résultat dans
 59961 FAC
 -5575

\$EA55 DIV10 Calcule $\text{FAC}/10$ et range le résultat dans FAC
 59989
 -5547

\$EA69 FDIVT Calcule ARG/FAC et range le résultat dans
 60009 FAC
 -5527

\$EA66 FDIV Place le nombre dont l'adresse mémoire est dans
 60006 les registres Y et A dans ARG puis exécute FDIVT
 -5530 (voir CONUPK)

Les sous programmes suivants servent à effectuer des transferts de nombres réels, comme CONUPK.

En effet, avant d'effectuer un calcul, il faut placer un nombre dans FAC, et éventuellement dans ARG; puis recopier le résultat en mémoire, ce qui justifie l'utilité de ces sous programmes. Attention : les signes ne sont pas transférés. Voir SGN

\$EAF9 MOVFM Place le nombre dont l'adresse mémoire est
 60153 dans les registres Y et A dans FAC. Voir SGN
 -5383

Exemple : Supposons que vous désiriez placer dans FAC le nombre rangé dans les cases mémoires \$0A84 à \$0A88

Il faut faire : LDY # \$0A
 LDA # \$84
 JSR \$EAF9

C'est-à-dire qu'il s'utilise comme CONUPK.

\$EB1E MOV2F recopie TEMP2 dans FAC. Voir SGN

60190
 -5346

\$EB21 MOV1F recopie TEMP1 dans FAC. Voir SGN

60193
 -5343

\$EB23 MOVML recopie FAC en page zéro à l'adresse con-
 tenue dans le registre X. Voir SGN

60195
 -5341

\$EB2D MOVMF Transfert inverse de MOVFM. Recopie le
 FAC en mémoire, à l'adresse dont l'octet de poids faible
 est dans le registre Y et l'octet de poids fort dans X.
 Voir SGN

\$EB53 MOVFA Recopie ARG dans FAC. Voir SGN

60243
 -5293

\$EB63 MOVAF Recopie FAC dans ARG. Voir SGN

60259
 -5277

Les sous programmes suivants permettent diverses manipulations sur FAC

\$EB80 SGN Un nombre réel est constitué de 5 octets. Un
 60288 pour l'exposant, quatre pour la mantisse. Le signe du
 -5248 nombre peut être déterminé à partir de la valeur de

la mantisse, mais cela n'est pas très simple. Aussi, avant d'exécuter un calcul sur un nombre, on exécute SGN qui place dans A une valeur correspondant au signe de FAC (0 si nul, 1 si positif, FF si négatif).

Pratiquement, il faut exécuter JSR \$EB80 chaque fois que l'on a chargé FAC et que l'on va appeler un sous-programme de calcul.

Exemple : Si l'on désire calculer le carré du nombre placé dans l'accumulateur :

JSR \$EB63
 JSR \$EB80
 JSR \$E982
 JSR \$ED2E
 RTS

On exécute MOVAF (JSR \$EB63) qui recopie FAC dans ARG.
 Puis JSR \$EB80 (SGN) calcule le signe de FAC. JSR \$E982 (FMULTT) calcule $FAC \times ARG$ et place le résultat dans FAC.
 On affiche le résultat grâce à PRNTFAC (JSR \$ED2E).

Si l'on avait oublié JSR \$EB80, on aurait obtenu un résultat négatif, ce qui est étonnant pour un carré. De même, l'oubli de SGN peut conduire l'Apple à exécuter des soustractions à la place des additions.

\$EB93 FLOAT Convertit l'entier contenu dans A (-
 60307 $128 \leq A \leq 127$) en un nombre réel qui est placé dans
 -5229 FAC.

\$EBAF ABS Calcule la valeur absolue de FAC et place le
 60335 résultat dans FAC.
 -5201

\$EC23 INT Calcule la valeur entière de FAC, c'est-à-dire le
 60451 plus grand entier relatif inférieur au nombre. Le résultat
 -5085 est placé dans FAC.

\$ED24 LINPTR Affiche le contenu des registres X et A
60708
-4828

\$ED2E PRNTFAC Affiche la valeur contenue dans FAC
60718
-4818

\$ED34 FOUT Convertit la valeur contenue dans FAC en une
60724 chaîne de chiffres représentant cette valeur.

\$EE8D SQR Calcule la racine carrée de FAC. Le résultat
61069 est placé dans FAC
1-467

\$EED0 NEGOP Calcule l'opposé de FAC. Le résultat est
61136 placé dans FAC
-4400

\$EF09 EXP Calcule l'exponentielle de FAC. Le résultat est
placé dans FAC

\$EFAE RND Engendre un nombre aléatoire dans FAC.
61358
-4178

\$EFEA COS Calcule le cosinus du FAC. Le résultat est placé
61418 dans FAC

\$EFF1 SIN Calcule le sinus du FAC. Le résultat est placé
61425 dans FAC
-4111

\$F03A TAN Calcule la tangente du FAC. Le résultat est
61498 placé dans FAC
-4038

\$F09E ATN Calcule l'arc tangente du FAC. Le résultat est
61598 placé dans FAC.
-3938

Note : Pour les fonctions trigonométriques, les valeurs sont exprimées en radians.

2.8. BASIC-ASSEMBLEUR : JONCTION RÉUSSIE

De façon générale, on écrit un programme principal en basic, ce langage permettant un dialogue aisé avec l'utilisateur, grâce à PRINT et INPUT, et l'on utilise des sous-programmes écrits en langage machine pour effectuer les calculs les plus longs. Il se pose trois problèmes.

a) La coexistence pacifique des programmes Basic et Assembleur

En effet, si l'on n'y prend pas garde, il est très probable que le basic place ses variables, ses tableaux ou ses chaînes dans la zone programme assembleur, détruisant ce dernier. Il faut donc penser à initialiser les valeurs P1, P2, LOMEM, T1, T2, C2 et HIMEM (Voir 2.6).

b) Passage du Basic à l'assembleur

Pour exécuter un sous-programme machine depuis un programme basic, ou depuis l'Applesoft en mode immédiat, il existe deux méthodes

- La commande CALL. Elle doit être suivie d'une expression arithmétique dont la valeur est l'adresse mémoire du début du sous-programme. En pratique, on utilise presque exclusivement des constantes. Attention : ces constantes doivent être exprimées en décimal, à l'inverse du moniteur qui ne comprend que l'hexadécimal.

Exemple : Utiliser le sous-programme HCLR, qui efface l'écran en mode haute résolution. Ce programme débute en \$F2F3.

Ce qui, en décimal s'exprime par le nombre $16^3 \times 15 + 16^2 \times 2 + 16 \times 15 + 3 = 62450$.

Comme les adresses sont définies modulo \$10000 (65536 en décimal), on peut aussi utiliser le nombre 62450-65536=-3086.

Ainsi CALL 662450 ou CALL-3086 appellent le sous-programme HCLR situé en \$FF2F3.

- La fonction **USR**. Elle a l'avantage de calculer son argument et de le placer dans FAC avant d'appeler le sous-programme en langage machine. Mais il ne faut pas oublier de placer une instruction JMP en \$0A, car il faut bien, d'une manière ou d'une autre, indiquer à l'Applesoft ou se trouve le sous-programme. Voir **USR**, tome 1.

Pour revenir au programme basic, ou l'Applesoft en mode immédiat, le sous-programme doit exécuter une instruction RTS. Ceci est valable tant pour CALL que pour **USR**. De façon générale, RTS permet de retourner là d'où l'on vient, programme basic, Applesoft ou Integger basic en mode immédiat, ou bien moniteur.

c) Un point délicat : le passage de valeurs entre programmes basic et assembleur

Voici, par ordre de complexité croissante, trois solutions à ce problème

- La fonction **USR**. En effet elle place la valeur de son argument dans FAC (adresses \$9D à \$A2). Ceci permet de passer un nombre réel à un sous-programme machine. En retour de sous-programme, l'Applesoft considère que le nombre qu'il trouve dans FAC est le résultat de la fonction **USR**. Ce qui permet de retourner un autre nombre réel au programme basic.

Avantage : la simplicité

Inconvénients : - ceci ne permet le passage que d'une valeur

- cette valeur ne peut être qu'un nombre réel.

- La méthode "boîte aux lettres". Le programme basic et le sous-programme machine utilisent les mêmes emplacements mémoire pour se communiquer des valeurs. Supposons par exemple, que l'on désire passer des valeurs entières comprises entre 0 et 255. On peut utiliser comme boîte aux lettres un seul octet, prenons celui d'adresse \$06 (6 en décimal).

Tout d'abord, on désire que le programme basic passe la valeur 19 au sous-programme. On fera donc POKE 6,19, ce qui place la valeur dans la boîte aux lettres. Puis on appelle le sous-programme machine, par CALL ou **USR**. Ce dernier "récupère" la valeur en exécutant LDA \$06, ce qui la place dans l'accumulateur A. Supposons qu'à la suite de divers calculs, le sous-programme désire communiquer au basic le nombre contenu dans l'accumulateur. Il suffit d'exécuter STA \$06, ce qui place en \$06 la valeur contenue dans l'accumulateur. RTS retourne au programme basic, qui à son tour récupère la valeur de la boîte aux lettres grâce à PEEK (6).

Voici la liste des adresses des emplacements que l'on peut utiliser comme boîte aux lettres

* en page zéro : \$06 (6 en décimal) à \$09 (9), soit 4 octets, \$CE (206) \$CF (207), ~~\$D6 (214)~~, \$D7 (215), \$EB (235) à \$EF (239) soit 5 octets et \$F9 (249) à \$FF (255) soit 7 octets. On peut aussi utiliser \$18 (24) à \$1F (31) mais avec précautions : ces octets sont utilisés par certains sous-programmes Applesoft (tracés haute résolution).

- On peut enfin utiliser les variables basic stockées en mémoire. Pour retrouver ces variables en mémoire, voir 2.6. On peut bien sûr écrire un sous-programme qui tel l'Applesoft recherche la variable en mémoire en utilisant les deux premières lettres du nom. Il est un peu plus simple de charger le programme basic en mémoire, l'exécuter une fois puis chercher (Voir 2.6.) la variable en mémoire, et noter son adresse. Le programme basic doit dans ce cas toujours rencontrer les variables dans le même ordre, afin qu'elles aient toujours la même adresse, d'une exécution à l'autre.

2.9. GRAPHIQUE ET ASSEMBLEUR

a) On a déjà donné la liste de sous-programmes graphiques basse résolution, qui font partie du moniteur au chapitre 2.7. Si l'on désire donner une impression de mouvement, cela ne peut se faire efficacement qu'en assembleur. En effet les sous-programmes graphiques, particulièrement en haute résolution, nécessitent des calculs d'adresse complexes. Si de plus l'on "perd" du temps à chercher des variables en mémoire, à vérifier la syntaxe, à interpréter des lignes etc. comme le fait le basic, on aboutit à des temps de calcul si importants que le mouvement obtenu est extrêmement lent.

Avant de donner la liste des sous-programmes graphiques haute résolution, voici quelques adresses "clé" constamment utilisées.

- \$E6 contient l'octet de poids fort de l'adresse de début de page où l'on effectue les tracés, les positionnements de curseur. C'est-à-dire que si \$E6 contient la valeur \$20, HPOSN place le curseur dans la page 1 haute résolution, HCLR efface cette même page, HPLOT, HLIN, DRAW et XDRAW tracent dans cette page. Si \$E6 contient la valeur \$40, les sous-programmes que l'on vient de citer travaillent dans la page 2 haute résolution.

- \$C054 et \$C055 permettent de faire afficher la page 1 ou la page 2.

- \$26, \$27, \$30, \$1C et \$E5 constituent le curseur haute résolution. Disons simplement que ces emplacements contiennent l'adresse mémoire du bit que l'on modifie, ou du dernier bit qui a été modifié.

- \$E4 contient le code de couleur haute résolution à utiliser pour les prochains tracés.

Pour plus de détails, voir le chapitre 1.5.

Un exemple est donné et commenté à la fin de ce chapitre.

b) Liste des sous-programmes graphiques haute résolution

(F3D4)

\$F3D8 HGR2 initialise le mode HGR2, c'est la commande
62424 HGR2 de l'Applesoft
-3112

(62420
-3116)

* pas de paramètre d'entrée

* en sortie - commute la page 2 (switch C055)

- commute le mode non mixte (switch C052)
- commute le mode haute résolution (switch C057)

- commute le mode graphique (switch C050)
- efface la page 2 haute résolution (de \$4000 à \$5FFF)

- range en \$E6 la valeur #\$40. Cette valeur est l'octet de poids fort de l'adresse de début de la page 2, et est utilisée dans les calculs d'adresse de HPOSN, HPLOT, HLIN, DRAW et XDRAW.

(F3DE)

\$F3E2 HGR initialise le mode HGR, c'est la commande HGR
62434 de l'Applesoft
-3102

* pas de paramètre d'entrée.

(62430
-3106)

* en sortie - commute la page 1 (switch C054)

- commute le mode mixte (switch C053)
- commute le mode haute résolution (switch C057)

- commute le mode graphique (switch C050)
- efface la page 1 haute résolution (de \$2000 à \$3FFF)

- range en \$E6 la valeur #\$20. Cette valeur est l'octet de poids fort de l'adresse de début de page, et est utilisée dans les calculs d'adresse de HPOSN, HPLOT, HLIN, DRAW et XDRAW.

(F3EE)

\$F3F2 HCLR efface la page courante haute résolution.

62450
-3086 * en entrée \$E6 contient #\$20 ou #\$40;

(62446
-3090)

- * en sortie - Si on a # $\$20$ en $\$E6$, efface la page 1 haute résolution
- Si on a # $\$40$ en $\$E6$, efface la page 2 haute résolution.

La valeur contenue en $\$E6$ est initialisée par HGR ou HGR2. Si par exemple vous avez exécuté une instruction JSR $\$F3E2$ ou JSR $\$F3D8$ vous n'avez pas besoin de modifier la valeur située en $\$E6$. La seule instruction JSR $\$F3F2$ suffit pour effacer la page courante.

De même, si en Applesoft vous avez exécuté une commande HGR ou HGR2, il suffit d'exécuter CALL-3086 ou CALL 62450 pour effacer la page courante.

(~~F3F0~~)
 $\$F3F4$ BKGND Efface la page courante haute résolution en 62452
 -3084
 (~~62448~~
 -3088)

- * en entrée - $\$E6$ contient # $\$20$ à # $\$40$ (Voir remarque pour HCLR)
- A (accumulateur) contient le code de la couleur désirée. (Voir page 19)
- * en sortie - La page courante est effacée et a la couleur spécifiée.

Exemple : JSR $\$F3E2$ (~~F3DE~~)
 LDA # $\$D5$
 JSR $\$F3F4$ (~~F3F0~~)
 RTS

Ces trois instructions affichent la page 1 haute résolution, qui apparaît uniformément rouge. En effet, JSR $\$F3E2$ appelle le sous-programme HGR qui initialise le mode HGR, puis on charge dans l'accumulateur (LDA # $\$D5$) la valeur # $\$D5$ qui est le code du rouge. On appelle enfin le sous-programme BKGND (JSR $\$F3F4$), qui rend l'écran uniformément rouge.

Dans cet exemple, le sous-programme BKGND est utilisé deux fois. En effet, HGR utilise BKGND pour effacer la page. On efface donc la page 1 une première fois en lui donnant la couleur noire (code # $\$00$) puis une seconde fois en lui donnant cette fois la couleur rouge (code # $\$D5$). Le premier effacement est inutile, on peut obtenir le même résultat que précédemment grâce à :

LDA # $\$20$ ranger en
 STA $\$E6$ $\$E6$ la valeur # $\$20$ indiquant la page 1
 LDA $\$C054$ commuter page 1
 LDA $\$C053$ mixer texte et graphique
 LDA $\$C057$ commuter mode haute résolution
 LDA $\$C050$ commuter mode graphique
 LDA # $\$D5$ charger le code du rouge
 JSR $\$F3F4$ (~~F0~~) et effacer l'écran, qui devient rouge.
 RTS

(~~F452~~)
 $\$F411$
 62481
 -3055
 (~~62546~~
 -2990)

HPOSN Sert à positionner le curseur haute résolution dans la page haute résolution courante. C'est-à-dire que connaissant les coordonnées (x,y) d'un point de l'écran, ce sous-programme calcule la position en mémoire du bit correspondant à ce point.

- * en entrée - $\$E6$ contient # $\$20$ ou # $\$40$ (Voir remarque pour HCLR)
- l'accumulateur contient la coordonnée y. On doit avoir $0 \leq y \leq 191$
- le registre X contient la partie basse de la coordonnée x
- le registre Y contient la partie haute de la coordonnée x ; x doit être compris entre 0 et 279
- $\$E4$ contient le code de la couleur définie par HCOLOR= en basic ou SETHCOL en langage machine.

- * en sortie - \$E2 contient la coordonnée y
- \$E0 contient la partie basse de la coordonnée x
- \$E1 contient la partie haute de la coordonnée x
- \$26 et \$27 contiennent respectivement les parties basses et hautes de l'adresse mémoire du premier octet de la ligne y
- \$E5 et le registre Y contiennent l'adresse dans la ligne de l'octet contenant le point (x,y). Ce nombre est compris entre 0 et 39.
- \$30 contient un masque isolant le bit cherché et le bit de poids fort.
- \$1C contient la même chose que \$E4 si x est pair. Si x est impair, \$1C contient une valeur obtenue formée des mêmes bits que \$E4, mais décalés.

Bien que ce qui précède ne soit pas très simple à comprendre, l'utilisateur est très facile.

Exemple : Supposons que vous désiriez savoir si le point de coordonnées (260, 500) de la page courante est allumé ou éteint. Il suffit d'exécuter les instructions suivantes :

```
LD/A # $32
LD/X # $04
LD/Y # $01
JSR $F411 (52)
LD/A ($26),Y
ANID $30
ANID # $7F
```

260 et 500 valent respectivement \$0104 et \$32 en hexadécimal. On charge donc la coordonnée y, soit # \$32 dans l'accumulateur, on charge les parties basses et hautes de la coordonnée x dans les registres X et Y, puis l'on appelle le sous-programme HPOSN (JSR \$1F411).

HPOSN a placé en \$26, \$27 et dans le registre Y les valeurs adéquates pour que l'instruction LDA (\$26),Y charge dans l'accumulateur l'octet contenant le bit correspondant au point (x,y). De même, HPOSN a placé en \$30 la valeur adéquate pour que l'instruction AND \$30 isole le bit de gauche et le bit correspondant au point cherché. Puis AND # \$7F élimine le bit de gauche. Le nombre se trouvant finalement dans l'accumulateur est :

a) nul si le point (x,y) est éteint.

b) non nul si le point (x,y) est allumé

(F498)

\$F457 HPLLOT Trace un point dans la page courante haute
62551 résolution. Produit exactement le même effet que la
-2985 commande basic HPLOT arg1, arg2. De manière pratique :

(62616
-2920)

* en entrée - placer dans l'accumulateur la coordonnée y du point à tracer
- placer dans les registres X et Y les parties basses et hautes de la coordonnée x du point à tracer.

* en sortie - Le point (x,y) est tracé. Sa couleur est celle définie par la dernière commande basic HCOLOR= ou par le dernier appel à SETHCOL.

Exemple :

```
LDX # $07
JSR $F6EC (c2)
LDA # $32
LDX # $04
LDY # $01
JSR $F457 (98)
```

Les deux premières instructions ont le même effet que HCOLOR=7 en basic, la couleur fixée est le blanc. La coordonnée y, ici 50 est chargée dans l'accumulateur (LDA # \$32), la coordonnée

x, ici 260 est chargée dans les registres X et Y (260 vaut \$0104 en hexadécimal). Enfin JSR \$F457 trace un point blanc en (x,y).

De manière plus détaillée, voici le listing de HPLCOT

```

F457- 20 11 F4 JSR $F411
F45A- A5 1C LDA $1C
F45C- 51 26 EOR ($26),Y
F45E- 25 30 AND $30
F460 51 26 EOR ($26),Y
F462- 91 26 STA ($26),Y
F464- 60 RTS

```

différent sur ITT

On appelle HPOSN (JSR \$F411) qui place en \$26, \$27 et dans le registre Y l'adresse de l'octet contenant le bit correspondant au point (x,y). HPOSN place aussi en \$1C et \$30 deux masques. Les instructions suivantes modifient cet octet afin de faire apparaître le point désiré. Les paramètres d'entrée et de sortie de HPLCOT sont exactement ceux de HPOSN.

(F56C)
 \$F53A HLIN Trace une ligne dans la page courante haute
 62778 résolution. Sans entrer dans les détails, voici comment
 -2758 utiliser HLIN :

(62828
 -2708)

a) Pour obtenir un résultat analogue à une commande basic HPLOT arg1, arg2TO, arg3, arg4 procéder comme suit :

- placer les parties basses et hautes de "arg1" dans les registres X et Y respectivement. Placer « arg2 » dans l'accumulateur.
- exécuter JSR \$F411. Ceci a pour effet de placer le curseur haute résolution en (arg1, arg2)
- placer les parties basses et hautes de « arg3 » dans l'accumulateur et le registre X respectivement. Placer "arg4" dans le registre Y.
- exécuter JSR \$F53A. Ceci trace la ligne.
- exécuter JSR \$F5CB. Ceci réinitialise certaines valeurs (\$E0,\$E1,\$E2).

Exemple : Soit le programme basic.

```

10 HGR
20 HCOLOR=3
30 HPLOT 50,60 TO 270,150

```

Précisons que :

```

50=$0032
60=$3C
270=$010E
150=$96

```

et voyons la traduction en langage machine de ces trois commandes basic.

```

10 JSR $F3E2(3E) passer en mode HGR
20 LDX #$03 fixer la couleur à
JSR $F6EC(C2) la valeur 3 par SETHCOL
30 LDX #$32 charger la partie basse de arg1
LDY #$00 charger la partie haute de arg1
LDA #$3C charger arg2
JSR $F411(52) positionner curseur en (arg1, arg2)
LDA #$0E charger la partie basse de arg3
LDX #$01 charger la partie haute de arg3
LDY #$96 charger arg4
JSR $F53A(6C) tracer la ligne
JSR $F5CB(F8) repositionner curseur en (arg3, arg4)

```

b) Pour obtenir un résultat analogue à une commande basic HPLOT TO arg3, arg4, procéder comme suit :

- Placer les parties basses et hautes de "arg3" dans l'accumulateur et le registre X respectivement. Placer "arg4" dans le registre Y
- Exécuter JSR \$F53A. Ceci trace la ligne

- Exécuter JSR \$F5CB. Ceci réinitialise certaines valeurs (\$E0, \$E1, \$E2)

Exemple : Tracé d'un carré. Le programme basic suivant trace un carré.

```

10 HGR:HCOLOR = 3
20 HPLOT 50,50
30 HPLOT TO 150,50
40 HPLOT TO 150,150
50 HPLOT TO 50,150
60 HPLOT TO 50,50:END

```

La ligne 20 place le curseur en (50, 50) et trace un point. La ligne 30 trace un trait de la dernière position du curseur, c'est-à-dire ici (50, 50), jusqu'à (150, 50). La ligne 40 trace une ligne de (150, 50) à (150, 150), etc.

Ayant précisé que 50 = \$32 et que 150 = \$96 voyons la traduction de ce programme en langage machine

```

10 JSR $F3E2(DE) passer en mode HGR
LDX #$03 fixer la couleur
JSR $F6EC(C2) à la valeur 3 grâce à SETHCOL
20 LDX #$32 charger partie basse abscisse (50)
LDY #$00 charger partie haute abscisse
LDA #$32 charger ordonnée (50)
JSR $F457(98) positionner curseur, tracer un point par HPLOT
30 LDA #$96 charger partie basse abscisse (150)
LDX #$00 charger partie haute abscisse
LDY #$32 charger ordonnée (50)
JSR $F53A(6C) tracer ligne de (50, 50) à (150, 50)
JSR $F5CB(F8) réinitialiser
40 LDA #$96 charger partie basse abscisse (150)
LDX #$00 charger partie haute abscisse
LDY #$96 charger ordonnée (150)
JSR $F53A(6C) tracer ligne de (150, 50) à (150, 150)
JSR $F5CB(F8) réinitialiser

```

```

50 LDA #$32 charger partie basse abscisse (50)
LDX #$00 charger partie haute abscisse
LDY #$96 charger ordonnée (150)
JSR $F53A(6C) tracer ligne de (150, 150) à (50, 150)
JSR $F5CB(F8) réinitialiser
60 LDA #$32 charger partie basse abscisse (50)
LDX #$00 charger partie haute abscisse
LDY #$32 charger ordonnée (50)
JSR $F53A(6C) tracer ligne de (50, 150) à (50, 50)
JSR $F5CB(F8) réinitialiser
RTS retour

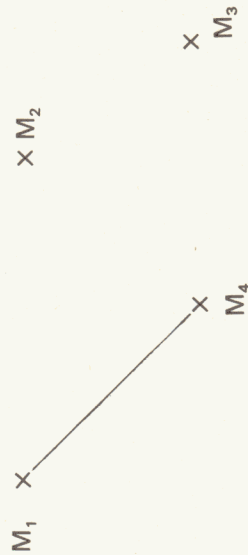
```

De manière plus détaillée, trois points interviennent comme paramètres d'entrée :

M1 (x₁, y₁). En entrée ; \$26, \$27, \$E5 contiennent l'adresse mémoire de l'octet correspondant à M1, tandis que \$1C et \$30 contiennent deux masques pour le bit correspondant à M1.
M2 (x₂, y₂). En entrée \$E2 contient y₂, \$E0 et \$E1 contiennent M2 (x₃, y₃). En entrée, l'accumulateur et le registre X contiennent x₃, le registre Y contient y₃.

Le sous programme HLIN trace le trait M₁ M₄ tel que

$$\overrightarrow{M1 M4} = \overrightarrow{M2 M3}$$



en sortie, - \$26, \$27, \$E5 contiennent l'adresse mémoire de l'octet correspondant à M4 tandis que \$1C et \$30 contiennent deux masques pour le bit correspondant à M4.
- \$E0, \$E1, \$E2 sont inchangés

- les valeurs de l'accumulateur des registres X et Y sont détruites.

Dans le cas a), c'est-à-dire une commande HPLOT arg1, arg2 TO arg3, arg4, les opérations effectuées sont les suivantes :

- On a placé "arg1" dans les registres X et Y, "arg2" dans l'accumulateur, on appelle le sous programme HPOSN (JSR \$F411). HPOSN a placé "arg1" en \$E0-\$E1, "arg2" en \$E2, alors que les valeurs correspondant au point (arg1, arg2) ont été rangées en \$26, \$27, \$E5, \$1C et \$30.

C'est-à-dire que les points M1 et M2 sont confondus et se trouvent en (arg1, arg2)

- On met arg3 dans l'accumulateur et le registre X, "arg4" dans le registre Y. C'est-à-dire que (arg3, arg4) est le point M3. On appelle HLIN (JSR \$F53A), qui trace le trait M1M4 tel que $\overrightarrow{M1M4} = \overrightarrow{M2M3}$. Mais comme M1 et M2 sont confondus, M3 et M4 sont donc aussi confondus.

en sortie de HLIN * \$26, \$27, \$E5, \$1C, \$30 contiennent les adresses et les masques correspondant au point (arg3, arg4), puisque $M3 = M4$.

* \$E0, \$E1, \$E2 sont inchangés, ils contiennent toujours "arg1" et "arg2"

- Les valeurs de "arg3" et "arg4" qui se trouvaient dans les registres ont été détruites. On appelle HFIND (JSR \$F5CB), qui recalcule ces valeurs à partir des nombres se trouvant en \$26, \$27, \$E5, \$30, puis range arg3 en \$E0-\$E1 et arg4 en \$E2.

(F562)

\$F530 HVECT Tracé d'un vecteur. Trace le vecteur en partant pour origine la position initiale du curseur, puis le curseur est positionné sur l'extrémité du vecteur tracé.

Pour tracer une chaîne de vecteurs V (v_1, v_2), W (w_1, w_2)... en partant d'une origine O (x_0, y_0) procéder comme suit.

- Placer les parties basses et hautes de x_0 dans les registres X et Y respectivement. Placer y_0 dans l'accumulateur.

- Exécuter JSR \$F411. Ceci place le curseur haute résolution en (x_0, y_0) grâce à HIPOSN.

- Placer les parties basses et hautes de V_1 dans l'accumulateur et le registre X respectivement. Placer V_2 dans le registre Y. Exécuter JSR \$F530. Ceci trace W.

- Placer les parties basses et hautes de W_1 dans l'accumulateur et le registre X respectivement. Placer W_2 dans le registre Y. Exécuter JSR \$F30. Ceci trace W.

.....

- Lorsque tous les vecteurs ont été tracés, exécuter JSR \$F5CB. Ceci initialise certaines valeurs (\$E0, \$E1, \$E2).

De manière plus détaillée : HVECT remet à zéro \$E0, \$E1 et \$E2 puis exécute HLIN.

()

\$F65D DRAW C'est la commande DRAW du basic.

Pour tracer une forme en partant du point O (x_0, y_0), procéder ainsi :

- Placer les parties basse et haute de x_0 dans les registres X et Y respectivement

Placer y_0 dans l'accumulateur

- Exécuter JSR \$F411. Ceci a pour effet de placer le curseur haute résolution en O (x_0, y_0)

- placer en \$E7 le facteur d'échelle. (En basic, ceci se fait par la commande SCAL#=#)

- placer dans l'accumulateur le facteur de rotation. (En basic, ROT=#)

- placer les parties basses et hautes de l'adresse du premier octet de la forme dans les registres X et Y respectivement

- Exécuter JSR \$F65D.

En sortie, le curseur est positionné sur le dernier point tracé. C'est-à-dire que \$26, \$E5, \$1C et \$30 contiennent les valeurs correspondant à ce dernier point.

() XDRAW C'est la commande XDRAW du basic. S'utilise comme DRAW, le JSR \$F65D étant remplacé par JSR \$F601.

c) Un exemple de programme graphique en assembleur

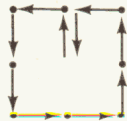
Le programme utilise le mode haute résolution. Il affiche cinq carrés de tailles différentes, de même centre, dont la taille augmente sans cesse. Lorsqu'un carré est trop grand pour la taille de l'écran, il disparaît puis réapparaît, tout petit au centre de l'écran.

Il y a quatre blocs de code :

- de \$300 à \$325, le programme principal qui commute les deux pages graphiques. En effet, pour donner l'impression de mouvement, on affiche l'une des pages tandis que l'on trace dans l'autre, puis l'on permute les rôles des deux pages, etc.

- de \$330 à \$361 un sous-programme qui trace les cinq carrés. Pour cela on utilise DRAW.

- de \$376 à \$37B la table de brme utilisée par DRAW. Elle correspond au tracé suivant :



Less codes de la forme sont \$21 \$3F \$36 \$2D \$1C \$00 (Voir 1.6)

- de \$371 à \$375 les cinq facteurs d'échelle-que l'on utilise pour le tracé des cinq carrés par DRAW. Ces facteurs valent initialement \$02, \$11, \$20, \$2F et \$3E.

Ils sont incrementés par le programme, ce qui permet de faire croître la taille des carrés, tracés par DRAW.

\$CALL-151

*300LLL

0300-	20	E2	F3	JSR	\$F3E2	→ F3DE
0303-	20	D8	F3	JSR	\$F3D8	→ F3D4
0306-	A9	20		LDA	#\$20	
0308-	8D	55	D0	STA	#\$C055	
030B-	85	E6		STA	#\$E6	
030D-	20	30	03	JSR	#\$0330	
0310-	A9	40		LDA	#\$40	
0312-	8D	54	D0	STA	#\$C054	
0315-	85	E6		STA	#\$E6	
0317-	20	30	03	JSR	#\$0330	
031A-	AD	00	D0	LDA	#\$C000	
031D-	10	E7		BFL	#\$0306	
031F-	AD	51	D0	LDA	#\$C051	
0322-	AD	54	D0	LDA	#\$C054	
0325-	60			RTS		
0326-	00			BRK		
0327-	00			BRK		
0328-	00			BRK		
0329-	00			BRK		
032A-	00			BRK		
032B-	00			BRK		
032C-	00			BRK		
032D-	00			BRK		
032E-	00			BRK		
032F-	00			BRK		
0330-	20	F2	F3	JSR	#\$3F2	→ F3EE
0333-	A9	5F		LDA	#\$5F	
0335-	A2	8B		LDX	#\$8B	
0337-	A0	00		LDY	#\$00	
0339-	20	11	F4	JSR	#\$411	→ F452
033C-	A0	05		LDY	#\$05	
033E-	84	06		STY	#\$06	
0340-	B9	70	03	LDA	#\$0370,Y	
0343-	85	E7		STA	#\$E7	
0345-	A9	00		LDA	#\$00	
0347-	A2	76		LDX	#\$76	
0349-	A0	03		LDY	#\$03	
034B-	20	5D	F6	JSR	#\$F65D	→ F75B ?
034E-	18			CLC		
034F-	A5	E7		LDA	#\$E7	
0351-	69	03		ADC	#\$03	

sous-programmes HGR et HGR2, dont les actions sont identiques aux commandes basic du même nom.

Les trois instructions suivantes placent la valeur # \$20 en \$C055 et \$E6. Le fait d'écrire quelque chose en \$C055 provoque l'affichage de la page 2, tandis que la valeur \$20 en \$E6 force l'utilisation de la page 1 pour les tracés. On appelle le sous-programme \$330 qui trace cinq carrés, dans cette page 1.

Puis, à partir de \$310, on place la valeur \$40 en \$C054 et \$E6. Le fait d'écrire quelque chose en \$C054 entraîne l'affichage de la page 1, les cinq carrés que l'on a tracés apparaissent donc. La valeur \$40 en \$E6 force l'utilisation la page 2 pour les tracés, effectués par le sous-programme \$330 que l'on appelle de nouveau.

En \$31A, on regarde si une touche n'a pas été frappée : La valeur située à l'adresse \$C000 est chargée dans l'accumulateur. Si cette valeur est positive (bit de poids fort à 0) aucune touche n'a été frappée, on se branche en \$306 et on recommence. Sinon on commute le mode texte (LDA \$CO51), on affiche la page 1 (LDA \$C054) et on termine (RTS).

Notez que dans ce cas l'instruction LDA \$C054 est inutile, car la dernière sélection de numéro de page (STA \$C054, en \$312) sélectionnait le même numéro 1 de page.

Néanmoins, il ne faut jamais oublier de commuter la page 1 en référant l'adresse \$C054 lorsque l'on revient en mode texte, car lorsque l'on frappe une touche, le caractère correspondant est affiché dans la page 1 basse résolution.

Sous programme

Il efface la page courante (1 si \$E6 contient # \$20, 2 si l'on a \$40 en \$E6) en appelant HCLR (JSR \$F3F2).

Les quatre instructions suivantes (de \$333 à \$33B), positionnent le curseur haute résolution sur le point de coordonnées \$008B et \$5F, c'est-à-dire 139 et 95 en décimal. Ce point est situé au milieu de l'écran.

0353-	C9	4D	CMP	#4D
0355-	30	02	BMI	#0359
0357-	A9	02	LDA	#02
0359-	A4	06	LDY	#06
035B-	99	70	STA	#0370,Y
035E-	88		DEY	
035F-	D0	DD	BNE	#033E
0361-	60		RTS	
0362-	00		BRK	
0363-	00		BRK	
0364-	FF		???	
0365-	FF		???	
0366-	00		BRK	
0367-	00		BRK	
0368-	FF		???	
0369-	FF		???	
036A-	00		BRK	
036B-	00		BRK	
036C-	00		BRK	

376-1 4 4 21 3F 36 2D 1C 0

Utilisation du programme :

- Exécuter CALL-151 pour appeler le moniteur. Tapez les codes des quatre blocs, qui débutent en \$300, \$330, \$371 et \$376.
- Pour vérifier le programme principal, exécutez *300 L "return"
- Pour vérifier le sous-programme, exécutez *330 L "return" puis *35BL "return"
- Pour faire exécuter le programme *300G "return"
- Pour cesser : frappez une touche quelconque (sauf reset !)
- Pour recommencer l'exécution *300G "return"

Explications détaillées

Programme principal (\$300 à \$325)

On commence par initialiser les deux pages haute résolution, par JSR \$F3E2 et JSR \$F3D8. C'est-à-dire que l'on appelle les

En \$33C et \$33E, on initialise un compteur de boucle, \$06 à la valeur #05. En effet les instructions suivantes doivent être exécutées 5 fois, afin de tracer 5 carrés. Puis les instructions LDA \$0370, Y et STA \$E7 initialisent le facteur d'échelle de DRAW, car ce sous-programme utilise la valeur contenue en \$E7 comme facteur d'échelle (SCALE en basic). On charge dans l'accumulateur un facteur de rotation nul (ROT=0 en basic) en \$345. Il n'y a plus qu'à placer les parties basses et hautes de l'adresse de la table de forme (\$0376) dans les registres X et Y avant d'appeler DRAW (JSR \$F65D).

De \$345 à \$352, on ajoute 3 au facteur d'échelle du carré que l'on vient de tracer. Ainsi, lors du prochain appel au sous-programme, ce carré sera tracé plus grand, ce qui donnera l'impression qu'il a grossi.

On vérifie que le facteur d'échelle n'est pas trop important (\$353). Si tel est le cas, on le remplace par la valeur 2 (en \$357). Puis l'on récupère dans le registre Y la valeur du compteur de boucle, on range le facteur d'échelle (STA \$0370, Y) et on décrémente le compteur de boucle. S'il n'est pas nul, on se branche en \$33E pour tracer un nouveau carré. Sinon, on a tracé les cinq carrés, retour au programme principal.

Modification : Remplacez l'instruction LDA #000 en \$345 par deux instructions NOP (code \$EA)

*345:EA EA "return"

Lorsque l'on exécute JSR \$F65D (appel de DRAW) en \$34B, l'accumulateur contient le facteur d'échelle, chargé par LDA \$0370, Y en \$340.

Or DRAW utilise la valeur qu'il trouve dans l'accumulateur comme facteur de rotation. Les carrés grossissent donc en tournant.

Annexe A

Les instructions du 6502

Le microprocesseur 6502, qui est celui du système Apple II, possède six registres.

7 6 5 4 3 2 1 0	[]	Accumulateur	A
7 6 5 4 3 2 1 0	[]	Registre d'index	X
7 6 5 4 3 2 1 0	[]	Registre d'index	Y
15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	Compteur ordinal	PC
7 6 5 4 3 2 1 0	[]	Pointeur de pile	S
[N Z C I D V]	[]	Registre d'état	P

Notations :

- A désigne l'accumulateur
- B désigne le second opérande pour les instructions à deux opérandes
- C désigne le bit de retenue (carry)
- D désigne le bit indicateur de mode décimal
- I désigne le bit indicateur d'interruptions masquables
- N désigne le bit indicateur de nombres négatifs
- P désigne le registre d'état, qui contient les bits indicateurs
- S désigne le registre pointeur de pile
- X désigne le registre d'index X
- Y désigne le registre d'index Y
- Z désigne le bit indicateur de résultat nul.

Lorsqu'une instruction ne modifie pas un bit indicateur, un caractère * est placé dans la colonne correspondante. Un • signifie que le bit est modifié suivant la valeur du résultat.

Pour plus de détails, on se reportera utilement à l'ouvrage : "L'assembleur facile du 6502", dans la même collection.

Instruction	actions	modes d'adressage	Mnémoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
ADC	A+B+C-A,C Addition avec la retenue	immédiat Page zéro ADC val ADC val·X Absolu Absolu·X Absolu·Y (indirect·X) ADC (val·X) ADC (val)·Y	ADC #val ADC val ADC val·X ADC val·Y Absolu Absolu·X Absolu·Y (indirect·Y)	69 65 101 75 117 7D 125 79 121 61 97 71 113	• • • • • • • • • • • • • • • • • • • •	2 2 2 3 3 3 2 2 2 2
AND	A1B-A "et" logique	immédiat Page zéro Page zéro·X Absolu Absolu·X Absolu·Y (indirect·X) AND (val·X) AND (val)·Y	AND #val AND val AND val·X AND val·Y Absolu Absolu·X Absolu·Y (indirect·Y)	29 41 25 37 53 2D 45 3D 61 39 57 21 31 49	• • • • • • • • • • • • • • • • • • • •	2 2 2 2 2 3 3 3 3 2 2 2
ASL	décalage d'un bit vers la gauche. Le bit B ₇ va dans C	Accumulateur Page zéro Page zéro·X Absolu Absolu·X	ASL ASL val ASL val·X ASL val·X Absolu·X	0A 10 06 6 16 22 14 30	• • • • • • • • • • • • • • • • • • • •	1 2 2 3 3 3 2
BCC	branche si C=0	relatif	BCC val	90	• • • • • • • • • •	2
BCS	branche si C=1	relatif	BCS val	B0	• • • • • • • • • •	2
BEO	branche si Z=1	relatif	BEO val	F0	• • • • • • • • • •	2

Instruction	actions	modes d'adressage	Mnémoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
CPX	Positionne N, Z et C suivant la valeur de X-B	Immédiate Page zéro Absolu	CPX #val CPX val	E0 224 E4 228 EC 236*	2 2 3
CPY	Positionne N, Z et C suivant la valeur de Y-B	Immédiate Page zéro Absolu	CPY #val CPY val	C0 192 C4 196 CC 204*	2 2 3
DEC	B-1-B	Page zéro Page zéro X Absolu Absolu X Absolu Y Absolu X (indirect X) (indirect Y)	DEC val DEC val X DEC val EOR val EOR val X EOR val Y EOR val X EOR val Y	C6 198 D6 214 CE 206 DE 222*	2 2 3 3 3 2 2
DEX	X - 1-X	implicite	DEX	CA 202*	1
DEY	Y - 1-Y	implicite	DEY	88*	1
EOR	A+B-A "ou exclusif" de A et B	Immédiate Page zéro Page zéro X Absolu Absolu X Absolu Y Absolu X (indirect X) (indirect Y)	EOR #val EOR val EOR val X EOR val EOR val X EOR val Y EOR val X EOR val Y	49 73 45 69 55 85 4D 77 5D 93 59 89 41 65 51 81*	2 2 2 3 3 3 2 2
INC	B+1-B	Page zéro Page zéro X Absolu Absolu X	INC val INC val X INC val	E6 230 F6 246 EE 238 FE 254*	2 2 3 3 3
INX	X+1-X	implicite	INX	E8 232*	1
INY	Y+1-Y	implicite	INY	C8 200*	1

Instruction	actions	modes d'adressage	Mnémoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
BIT	A B positionne C B ₆ →V, B ₇ →N	Page zéro Absolu	BIT val BIT val	24 36 2C 44	B ₇B ₆	2 3
BMI	branche si N=1	Relatif	BMI val	30 48*	2
BNE	branche si Z=0	Relatif	BNE val	D0 208*	2
BPL	branche si N=0	Relatif	BPL val	10 16*	2
BRK	break.empile PC empile P	implicite	BRK	00 0	*...*1...*	1
BVC	branche si V=0	Relatif	BVC val	50 80*	2
BVS	branche si V=1	Relatif	BVS val	70 112*	2
CLC	0-C	implicite	CLC	18 24	*...*0...*	1
CLD	0-D	implicite	CLD	D8 216	*...*0...*	1
CLI	0-I	implicite	CLI	58 88	*...*0...*	1
CLV	0-V	implicite	CLV	B8 184	*...*0...*	1
CMP	Positionne N, Z et C suivant la valeur de A-B	Immédiate Page zéro X Absolu Absolu X Absolu Y Absolu X (indirect X) (indirect Y)	CMP #val CMP val CMP val X CMP val CMP val X CMP val Y CMP val X CMP val Y	C9 201 C5 197 D5 213 CD 205 DD 221 D9 217 C1 193 D1 209*	2 2 2 3 3 3 3 2

Instruction	actions	modes d'adressage	Mnémomoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
LSR	décalage d'un bit vers la droite. Le bit Bo va dans C	Accumulateur Page zéro Page zéro·X Absolu	LSR LSR val LSR val·X LSR val·X	4A 46 56 4E 5E 74	∅	1 2 2 3 3 3
NOP	Aucune action (bouche-trou)	implicite	NOP	EA 234	*	1
ORA	AVB-A "ou" de A et B Immédiat Page zéro Page zéro·X Absolu Absolu·X Absolu·Y (indirect·X) (indirect·Y)	ORA #val ORA val ORA val·X ORA val ORA val·X ORA val·Y ORA (val·X) ORA (val·Y)	09 05 15 0D 1D 19 01 11	09 05 15 21 13 29 25 1 17	*	2 2 2 3 3 3 2
PHA	empile A	implicite	PHA	48 72	*	1
PHP	empile P	implicite	PHP	08 8	*	1
PLA	dépile A	implicite	PLA	68 104	*	1
PLP	dépile P	implicite	PLP	28 40	ce qui était empilé	1
ROL	Rotation d'un bit à gauche. C→Bo B7→C	Accumulateur Page zéro Page zéro·X Absolu Absolu·X	ROL ROL val ROL val·X ROL val ROL val·X	2A 26 36 2E 3E 42 46 54 62	*	1 2 2 3 3 3

Instruction	actions	modes d'adressage	Mnémomoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
JMP	branchement	Absolu Indirect	JMP val JMP (val)	4C 6C 76 108	*	3 3
JSR	branche à un sous programme empilé adresse instruction suivante	Absolu	JSR val	20 32	*	3
LDA	B-A charge l'accumulateur Immédiat Page zéro Page zéro·X Absolu Absolu·X Absolu·Y (indirect·X) (indirect·Y)	LDA #val LDA val LDA val·X LDA val·X LDA val·Y LDA (val·X) LDA (val·Y)	A9 A5 B5 AD BD B9 A1 B1	169 165 181 173 189 185 161 177	*	2 2 2 3 3 3 2
LDX	B-X charge le registre Immédiat Page zéro Page zéro·Y Absolu Absolu·Y	LDX #val LDX val LDX val·Y LDX val·Y LDX val·Y	A2 A6 B6 AE BE	162 166 182 174 190	*	2 2 2 3
LDY	B-Y charge le registre Immédiat Page zéro Page zéro·X Absolu Absolu·X	LDY #val LDY val LDY val·X LDY val·X LDY val·X	A0 A4 B4 AC BC	160 164 180 172 188	*	2 2 2 3

Instruction	actions	modes d'adressage	Mnémoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
STA	A←B	Page zéro Page zéro·X Absolu	STA val STA val·X STA val STA val·Y STA val·X STA val·Y STA (val·X) STA (val)·Y	85 133 95 149 8D 141 9D 157 99 153 81 129 91 145	* *	2 2 3 3 3 2 2
STX	X←B	Page zéro Page zéro·Y Absolu	STX val STX val·Y STX val	86 134 96 150 8E 142	* * * * * * * * * * * * * * *	2 2 3
STY	Y←B	Page zéro Page zéro·X Absolu	STY val STY val·X STY val	84 132 94 148 8C 140	* * * * * * * * * * * * * * *	2 2 3
TAX	A←X	Implicite	TAX	AA 170	* * * * *	1
TAY	A←Y	Implicite	TAY	A8 168	* * * * *	1
TSX	S←X	Implicite	TSX	BA 186	* * * * *	1
TXA	X←A	Implicite	TXA	8A 138	* * * * *	1
TXS	X←S	Implicite	TXS	9A 154	* * * * *	1
TYA	Y←A	Implicite	TYA	98 152	* * * * *	1

Instruction	actions	modes d'adressage	Mnémoniques	Codes hexa. déci.	N Z C I D V	nbre d'octets
ROR	Rotation d'un bit à droite. C←B, Bo←C	Page zéro Page zéro·X Absolu	ROR val ROR val·X ROR val ROR val·X	6A 106 66 102 76 118 6E 110 7E 126	* * * * * * * * * * * * * * *	1 2 2 3 3
RTI	Retour d'inter-ruption dépile puis PC	Implicite	RTI	40 64	ce qui était empilé	1
RTS	Retour de sous programme dépile PC	Implicite	RTS	60 96	* * * * *	1
SBC	A←B+C←A Soustraction avec la retenue	Immédiat Page zéro Page zéro·X Absolu Absolu·X Absolu·Y (indirect·X) (indirect·Y)	SBC #val SBC val SBC val·X SBC val SBC val·X SBC val·Y SBC (val·X) SBC (val)·Y	E9 233 E5 229 F5 245 ED 237 FD 253 F9 249 E1 225 F1 241	* *	2 2 2 3 3 3 2 2
SEC	1←C	Implicite	SEC	38 56	* * * * *	1
SED	1←D	Implicite	SED	F8 248	* * * * *	1
SEI	1←I	Implicite	SEI	78 120	* * * * *	1

Annexe B

Adresses de branchement du système

Adresse		Contenu de l'adresse
hexa.	deci.	
\$0A	10	Contiennent une instruction JMP vers le sous programme machine utilisé par la commande USR du basic.
\$0B	11	
\$0B	12	
\$03F0	1008	Contiennent l'adresse du sous programme à exécuter lorsqu'une instruction BRK est rencontrée. Cette adresse vaut normalement \$FA59
\$03F1	1009	
\$03F2	1010	Contiennent une adresse où l'on se branche après la frappe de "reset". Néanmoins, lors de la première frappe de reset, cette adresse n'est pas utilisée, car elle n'a alors pas encore été initialisée. Normalement cette adresse est celle du basic.
\$03F3	1011	
\$3F5	1013	Contiennent une instruction JMP vers un sous programme qui est exécuté lorsque l'Applesoft rencontre le caractère &, en mode immédiat ou programme. Normalement, il y a \$4C \$58 \$FF qui sont les codes de JMP \$FF58. En \$FF58 se trouve une instruction RTS qui provoque un retour immédiat à l'Applesoft. Vous pouvez remplacer \$FF58 par l'adresse d'un programme machine, qui sera donc exécuté chaque fois que l'Applesoft rencontrera &.
\$3F6	1014	
\$3F7	1015	
\$3F8	1016	Contiennent une instruction JMP vers un sous programme qui est exécuté lorsque le moniteur rencontre le caractère ctrl-Y dans une ligne de commande. Ceci est l'analogue du caractère & de l'Applesoft.
\$3F9	1017	
\$3FA	1018	
\$3FB	1019	Contiennent une instruction JMP vers le sous programme de traitement d'interruptions non masquables (NMI)
\$3FC	1020	
\$3FD	1021	
\$3FE	1022	Contiennent l'adresse du sous programme de traitement d'interruptions masquables (IRQ, interrupt request)
\$3FF	1023	

Annexe C

Le clavier

Le clavier de l'Apple II utilise deux adresses réservées, \$C000 et \$C010. Lorsque l'on frappe une touche, le code Ascii de cette touche est placé à l'adresse \$C000. Un programme peut lire cette valeur, qui est supérieure à 128, grâce à LDA \$C000 ou PEEK(49152).

Puis, le programme doit "acquitter" la touche : il doit effectuer une référence (lecture ou écriture) à l'adresse \$C010. Ceci indique au clavier que la valeur en \$C000 a été lue, et donc qu'il peut y placer une autre valeur, à la prochaine touche frappée. Tant que l'adresse \$C010 n'a pas été référencée, la valeur en \$C000 ne peut être modifiée, même si vous frappez une nouvelle touche.

D'autre part, après la référence en \$C010, la valeur en \$C000 devient inférieure à 128, en effet le bit de gauche de \$C000, qui était à 1, (valeur supérieure à 128) passe à 0 (valeur inférieure à 128, \$80 en hexadécimal).

Voici, en hexadécimal, la liste des codes que l'on peut lire en \$C000
le tableau indique les codes obtenus lorsque l'on frappe la touche seule, lorsque l'on frappe CTRL et la touche, lorsque l'on frappe SHIFT et la touche, lorsque l'on frappe CTRL, SHIFT et la touche

touche seule	CTRL	SHIFT	CTRL	SHIFT	CTRL	SHIFT	CTRL	SHIFT	CTRL	SHIFT	CTRL	SHIFT
seule			seule		seule		seule		seule		seule	
barre	A0	A0	A0	A0	AF	BF	AF	BF	D0	D0	D0	D0
Ø	B0	B0	B0	B0	C1	C1	C1	C1	D1	D1	D1	D1
1 1	B1	A1	A1	A1	C2	C2	C2	C2	D2	D2	D2	D2
2 "	B2	A2	A2	A2	C3	C3	C3	C3	D3	D3	D3	D3
3 #	B3	A3	A3	A3	C4	C4	C4	C4	D4	D4	D4	D4
4 \$	B4	A4	A4	A4	C5	C5	C5	C5	D5	D5	D5	D5
5 %	B5	A5	A5	A5	C6	C6	C6	C6	D6	D6	D6	D6
6 &	B6	A6	A6	A6	C7	C7	C7	C7	D7	D7	D7	D7
7 .	B7	A7	A7	A7	C8	C8	C8	C8	D8	D8	D8	D8
8 (B8	A8	A8	A8	C9	C9	C9	C9	D9	D9	D9	D9
9)	B9	A9	A9	A9	CA	CA	CA	CA	DA	DA	DA	DA
:	BA	AA	AA	AA	CB	CB	CB	CB	return	8D	8D	8D
;	BB	AB	AB	AB	CC	CC	CC	CC	esc	9B	9B	9B
,	AC	BC	BC	BC	CD	CD	CD	CD	→	88	88	88
-	AD	BD	BD	BD	CE	CE	CE	CE	←	95	95	95
•	AE	BE	BE	BE	CF	CF	CF	CF				

Après que \$C010 ait été référencée, la valeur que l'on peut lire est égale à celle lue dans ce tableau moins \$80.