P.S.I.
EDITIONS DU P.S.I.

l'assembleur de l'ORIC 1 et ORIC-ATMOS

PROGRAMMATION EN LANGAGE MACHINE

marcel henrot

L'assembleur de L'ORIC 1 et ORIC - ATMOS

La collection « **MATERIELS** » s'intéresse à l'utilisation de tel ou tel type d'ordinateur, depuis la première prise de contact jusqu'à l'utilisation la plus pointue. Elle est aujourd'hui constituée de 39 titres.

— Exercices pour Apple II — Frédéric Lévy
— La découverte de l'Apple II — Dominique Schraen et Frédéric Lévy
— La pratique de l'Apple II — volumes I et II — Nicole Bréaud-Pouliquen
— La pratique de l'Apple II — volume III — Nicole Bréaud-Pouliquen et Daniel-Jean David
— La découverte de l'Atari — Daniel-Jean David
— La découverte du Commodore 64 — Daniel-Jean David
— La pratique du Commodore 64 — Daniel-Jean David
— L'assembleur du Commodore 64 — Daniel-Jean David
— Dragon tout feu tout flammes — Trévor Toms et John Phipps — traduit par Olivier Arnaud
— La découverte du FX-702P — Jean-Pierre Richard
— Un ordinateur personnel signé IBM — Pierre Lotigie-Laurent
— La découverte du PB-100 — Pierrick Moigneau
— La découverte du PB-700 — Pierrick Moigneau
— La découverte du PC-1251 — Jean-Pierre Richard
— La découverte du PC-1500 — Jean-Pierre Richard
— Le petit livre du Spectrum — Trévor Toms — traduit par Allan Keil
— Exercices pour ZX-Spectrum — Julien Lévy
— La pratique du ZX-Spectrum — tome 1 — Xavier Linant de Bellefonds
— La pratique du ZX-Spectrum — tome 2 — Marcel Henrot
— La découverte du T07 — Dominique Schraen et Maurice Charbit
— Exercices pour T07 — Maurice Charbit et Dominique Schraen et Frédéric Lévy
— La découvertedu M05 — Dominique Schraen et Maurice Charbit
— Exercices pour M05 — Dominique Schraen et Maurice Charbit
— Les graphiques sur TRS-80 — Donn Inman — traduit par Alain Pinaud
— La découverte de la TI-57 — Xavier de la Tullaye
— La découverte du TI-99/4A — Frédéric Lévy et Dominique Schraen
— Exercices pour TI-99/4A — Frédéric Lévy
— La découverte du VIC — volume 1 — Daniel-Jean David
— La pratique du ZX-81 — tome 1 — Xavier Linant de Bellefonds
— La pratique du ZX-81 — tome 2 — Marcel Henrot
— Le petit livre du ZX-81 — Trévor Toms — traduit par Ghislaine Lapeyre
— La découverte de l'Oric 1 et Atmos — Daniel-Jean David
— L'assembleur de l'Oric — Marcel Henrot
— La découverte de l'Alice et du MC/10 — Maurice Charbit
— La pratique du Vic — Daniel-Jean David
— La découverte du Dragon — Dominique Schraen et Frédéric Lévy
— Macintosh le magnifique — Meri K. Miller et Mary A. Myers — traduit par Raymond Borraz

Autres ouvrages relatifs à l'Oric :

— La découverte de l'Oric — Daniel-Jean David
Collection « MATERIELS »
— Oric pour tous — Jacques Boisgontier et Sophie Brébion
Collection « ...POUR TOUS »
— 52 programmes Oric pour tous
Collection « ...POUR TOUS »
— Clefs pour Oric
Collection « MEMENTOS »
— Cahier du Basic sur Oric
Collection « CAHIERS »
— 102 programmes pour Oric Atmos — Jacques Deconchat
Collection « PROGRAMMES »

RAPPELS

Les collections :

Les ouvrages d'Edition du PSI, actuellement au nombre de 165, sont répartis en collections :

« MATERIELS », « UTILISATIONS DE L'ORDINATEUR », « MEMENTOS », « PROGRAMMES », « GUIDES PRATIQUES », « LANGAGES », « METHODES PRATIQUES », « TECHNIQUES ET METHODES », « LOGIGUIDE », et pour débutants, les collections **« INITIATION », « ...POUR TOUS »**.

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 1er de l'article 40).

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

© Editions du P.S.I. B.P. 86 — 77402 Lagny Cedex 1984
ISBN : 2-86595-168-5

L'assembleur de L'ORIC 1 et ORIC - ATMOS

par

Marcel Henrot

Editions du P.S.I.
1984

SOMMAIRE

	Pages
CHAPITRE I : INTRODUCTION ET RAPPELS	7
Les systèmes de numération	8
La présentation des nombres	10
Les opérations	13
Le système Oric	16
Le microprocesseur 6502A	19
Les codes machine	28
CHAPITRE II : OPERATIONS DE BASE	31
Exploitation des programmes en C.M.	31
Les codes machine de chargement	33
Les codes machine qui réalisent les opérations arithmétiques	41
CHAPITRE III : BOUCLES, COMPARAISONS, TESTS	55
L'écran basse résolution	55
Les branchements conditionnels relatifs	57
Les comparaisons	64
Les opérations logiques	70
Les instructions BIT	74
CHAPITRE IV : HAUTE DEFINITION, ATTRIBUTS	83
Quelques codes machine divers	83
Les glissements et les rotations	85
Les attributs série	88
L'écran haute résolution	95
CHAPITRE V : INTRODUCTION AUX TECHNIQUES AVANCEES	105
Quelques programmes	105
Délai programme	117
Génération de nombres pseudo-aléatoires	119
L'animation	121

CHAPITRE VI : LA MEMOIRE MORTE 131

Quelques routines 132
Points d'entrée de quelques routines 138
Différents moyens d'appeler des codes machine : ! & ET USR 143

ANNEXES

Annexe I : Liste alphabétique des mnémoniques 149

Annexe II : Liste numérique des mnémoniques 156

Annexe III : Influence des mnémoniques sur les indicateurs 157

Annexe IV : Table des nombres avec signe pour les valeurs négatives 158

Annexe V : Conversion Hexadécimal - Décimal 159

CHAPITRE 1
INTRODUCTION ET RAPPELS

La programmation en langage machine est la suite naturelle de la programmation en Basic. On supposera le lecteur familiarisé avec ce langage. La lecture attentive du manuel de l'ORIC ainsi que celle d'un livre comme celui de Jacques Boisgontier "L'ORIC pour Tous", constituent un bon préalable à la lecture de ce livre.

La méthode adoptée consiste à transposer progressivement le langage Basic en langage machine. La connaissance du premier rend plus facile l'étude du second.

Parmi les avantages de la programmation en codes machine, citons :

- la vitesse d'exécution est de loin supérieure à celle du Basic. C'est primordial pour beaucoup de programmes, et c'est d'ailleurs la vitesse la plus rapide que l'on puisse obtenir ;
- la consommation d'emplacements mémoire est moindre qu'en Basic. La mémoire disponible permet des programmes plus longs, et, ou, plus nombreux ;
- la création de nouvelles commandes et fonctions, orientées vers les besoins spécifiques de l'utilisateur, est réalisable totalement.

Quelques rappels sont nécessaires avant d'aborder l'étude des codes machine. Ces rappels portent principalement sur les systèmes de numération, le binaire étant celui utilisé par tout ordinateur, sur les opérations arithmétiques et logiques réalisées par le microprocesseur, sur l'organisation interne de l'ORIC et sur

le microprocesseur employé : le 65Ø2A. Le lecteur, déjà au courant de ces notions, passera au chapitre suivant ou au dernier paragraphe de ce chapitre.

LES SYSTEMES DE NUMERATION

L'ordinateur est une machine qui ne comprend, ne stocke et ne réalise des opérations que sur des informations qui n'ont que deux états logiques possibles : l'état Ø ou l'état 1, faux ou vrai, bas ou haut. Ceci implique que le **système binaire** est le plus adéquat pour traiter ces informations. Le système binaire est à la base d'un autre système de numération : le **système hexadécimal**, qui rend plus conçise la représentation binaire de toute information.

La numération binaire

Dès maintenant, nous emploierons le mot bit pour désigner un chiffre binaire, le mot quartet pour désigner un groupe de 4 bits et le mot octet pour désigner un groupe de 8 bits.

Le système binaire, comme le décimal ou l'hexadécimal, est un système positionnel. La position d'un chiffre dans un nombre, soit à gauche, soit à droite de la virgule, confère à ce chiffre une certaine valeur, un certain poids.

Un système de numération positionnel se représente comme suit :

$$\ldots + a_3.B^3 + a_2.B^2 + a_1.B^1 + a_\emptyset.B^\emptyset , + a_{-1}.B^{-1}$$

$$+ a_{-2}.B^{-2} + \ldots$$

où **B** est la base, **2** dans le binaire, et où les coefficients **a** sont pris parmi les chiffres du système, **Ø** et **1** dans le binaire.

Par exemple, le nombre 1ØØ11ØØ1b peut s'écrire :

$$1.2^7 + \emptyset + \emptyset + 1.2^4 + 1.2^3 + \emptyset + \emptyset + 1.2^\emptyset$$
$$= 128 + 16 + 8 + 1 = 153d$$

Nous ferons suivre un nombre binaire par **b**, un nombre

décimal par **d** et un hexadécimal par **h**, sauf quand il n'y a aucune confusion possible.

Le nombre ci-dessus peut être mis dans un octet, qui est la dimension de la cellule mémoire de l'ORIC.

7	6	5	4	3	2	1	Ø	poids (exposant de 2)
1	Ø	Ø	1	1	Ø	Ø	1	nombre

Pour préciser le bit d'un octet, nous employons son poids. Le bit Ø de l'octet ci-dessus est 1, le bit 1 est Ø, etc.

La numération hexadécimale

La numération hexadécimale, de base 16, possède 16 chiffres : de Ø à 9,A,B,C,D,E et F (les 6 derniers considérés comme ayant les valeurs 1Ø,11,12,13,14 et 15 respectivement). Cette numération est plus dense que la binaire, nous l'employons presque constamment.

En examinant le **tableau 1**, vous voyez qu'un quartet nécessite un chiffre hexadécimal. Donc chaque groupe de 4 bits d'un nombre binaire se représente par un chiffre hexadécimal. Par exemple :

1ØØ11ØØ1b = 99h = $9.16^{1} + 9.16^{Ø}$ = 144 + 9 = 153d

Binaire B=2	*Décimal B=1Ø*	*Hexa B=16*	*Binaire B=2*	*Décimal B=1Ø*	*Hexa B=16*
Ø	Ø	Ø	1Ø1Ø	1Ø	A
1	1	1	1Ø11	11	B
1Ø	2	2	11ØØ	12	C
11	3	3	11Ø1	13	D
1ØØ	4	4	111Ø	14	E
1Ø1	5	5	1111	15	F
11Ø	6	6	1ØØØØ	16	1Ø
111	7	7	1ØØØ1	17	11
1ØØØ	8	8	1ØØ1Ø	18	12
1ØØ1	9	9	1ØØ11	19	13

Tableau 1 - correspondance entre les numérations

Un nombre de 16 bits, qui sera le nombre le plus long que nous traiterons, se représente par 4 chiffres hexa. Ainsi 11ØØ ØØØØ ØØØØ ØØØØb = CØØØh = 49152d

Conversion de base

Le **passage de base 2 en base 16** et inversement est facile. On sépare le nombre binaire en quartets et chaque quartet est converti en un chiffre hexadécimal suivant le tableau 1.

1100	1000	en base 2 égale
C	8	en base 16

Il vient naturellement pour le passage inverse :

C	8
1100	1000

Le **passage de base 2 ou 16 en base 10** se fait sans difficultés à l'aide de la formule des numérations positionnelles vue plus haut. Nous y avons fait notamment 10011001b = 153d.

Le passage de base 10 en base 2 peut se faire, entre autres, de la façon suivante :

153 - 128 (1.2^7) = 25 - 16 (1.2^4) = 9 - 8 (1.2^3)
= 1 - 1 (1.2^0) = 0

On retranche chaque fois la puissance de 2 la plus élevée possible pour arriver finalement à 0. Ici les poids 7, 4, 3 et 0 auront le chiffre 1 et les autres poids le chiffre 0.

Le **passage de base 10 en base 16** se fera en 2 étapes : conversion de base 10 en base 2 puis de base 2 en base 16. Les conversions de base 16 se font facilement avec l'ORIC puisque celui-ci contient les routines nécessaires. Il suffit de préfixer le nombre hexa. par le signe dièze # .

PRINT #C000	affiche 49152	C000h=49152d
PRINT HEX$(1280)	affiche #500	1280d=500h

LA REPRESENTATION DES NOMBRES

Quand on manipule des nombres avec une partie fractionnaire, située à droite de la virgule, une erreur de conversion apparaît. Par exemple, pour représenter 0,6d en binaire pure, on peut écrire 0,10011b. Même avec 5

chiffres binaires, on a encore une erreur d'un peu plus de 1%.

$$\begin{aligned} Ø,1ØØ11 &= 1.2^{-1} + Ø + Ø + 1.2^{-4} + 1.2^{-5} \\ &= Ø,5 + Ø,Ø625 + Ø,Ø3125 \\ &= Ø,59375d \text{ au lieu de } Ø,6d \end{aligned}$$

La représentation des nombres en **virgule flottante** élimine cette erreur. Mais, quand les nombres deviennent très grands, cette représentation amène également des erreurs. Or, il peut être nécessaire d'avoir une grande précision. Par exemple, les comptables qui établissent les comptes et les bilans, équilibrent rigoureusement les colonnes Actif et Passif. Pour cette raison, la représentation des nombres en **DCB** a été imaginée. Le microprocesseur de l'ORIC fonctionne très facilement dans ce mode, il suffit de changer un indicateur.

Le code DCB

Le code **DCB** pour **D**écimal **C**odé en **B**inaire (BCD pour les Anglais) code chaque chiffre décimal en binaire. Il faut 4 bits pour représenter un chiffre décimal. IL y aura donc autant de quartets que de chiffres décimaux. Comme un quartet peut s'arranger de 16 façons différentes et que l'on n'en prend que 1Ø, il y a 6 arrangements superflus, c'est un code redondant. Comme conséquence directe, un octet représente seulement deux chiffres décimaux, soit un nombre de Ø à 99, quand il est codé en DCB et un nombre de Ø à 255 en binaire pur. Il y a consommation d'octets mémoire.

Décimal	*Code DCB*	*Décimal*	*Code DCB*
Ø	Ø	1Ø	ØØØ1 ØØØØ
1	1	11	ØØØ1 ØØØ1
2	1Ø	12	ØØØ1 ØØ1Ø
3	11	13	ØØØ1 ØØ11
4	1ØØ	14	ØØØ1 Ø1ØØ
5	1Ø1	15	ØØØ1 Ø1Ø1
6	11Ø	16	ØØØ1 Ø11Ø
7	111	17	ØØØ1 Ø111
8	1ØØØ	18	ØØØ1 1ØØØ
9	1ØØ1	19	ØØØ1 1ØØ1
		2Ø	ØØ1Ø ØØØØ

Tableau 2 - codage des nombres en DCB

1984d s'écrira en DCB :

ØØØ1 1ØØ1 1ØØØ Ø1ØØ

Ø,6d s'écrira en DCB :

Ø, Ø11Ø

La virgule flottante

L'ORIC, comme toute calculatrice scientifique, passe en notation exponentielle quand les nombres deviennent trop grands ou trop petits. La précision reste de 9 chiffres significatifs. Quelles que soient leurs représentations à l'affichage, l'ORIC tient les nombres en mémoire, en virgule flottante, bien sûr en base 2. Les conversions sont programmées dans la mémoire de l'ordinateur.

La représentation en virgule flottante se fait en employant la formule :

$$N = m.2^{e-128}$$

où **m** est la mantisse et **e** l'exposant.

L'ORIC utilise 5 octets pour cette représentation. Le premier octet donne la valeur de l'exposant **e**. Les quatre octets suivants donnent la valeur de la mantisse **m** dont tous les bits sont à droite de la virgule (point décimal pour les Anglais). Le premier bit significatif (toujours un 1 en binaire) sera remplacé par Ø pour un nombre positif et restera 1 pour un nombre négatif.

Ceci amène les **règles de conversion** suivantes :

- écrire les 4 octets de la mantisse en binaire ;
- si le premier bit est Ø, le remplacer par 1 ;
  si le premier bit est 1, mettre - devant le nombre ;
- multiplier la mantisse par 2^{e-128}

Pour vérifier ceci, entrons le petit programme suivant :

```
10 INPUT A
20 LET B=DEEK(156)
30 FOR I=B+2 TO B+6
40 PRINT PEEK(I);
50 NEXT
```

La ligne 2Ø donne l'adresse de départ de la zone varia-

bles. On fait +2 dans la ligne 3Ø car les 2 premiers octets représentent le nom de la variable.

RUN puis entrons 1ØØØ, l'écran affiche :

138 122 Ø Ø Ø

Ecrivons la mantisse en binaire :

Ø,Ø1111Ø1Ø

Le premier chiffre est Ø, c'est donc un nombre positif, remplaçons ce Ø par un 1 :

Ø,11111Ø1Ø

Multiplions par $2^{138-128}$ soit par 2^{10}

Pour multiplier par 2 en binaire, on décale le nombre d'un rang vers la gauche (comme pour multiplier par 1Ø en décimal), il vient :

1111Ø1ØØØb ou 3E8h
ou 3.256 + 14.16 + 8.1 = 1ØØØd

RUN puis entrons -1ØØØ, l'écran affiche

138 25Ø Ø Ø Ø

La mantisse est Ø,1111Ø1
Le premier chiffre est 1, donc c'est un nombre négatif

-Ø,1111Ø1

multiplions par 2^{10}

-1111Ø1ØØØb = -1ØØØd

Remarque : l'ORIC permet la manipulation des nombres entiers grâce au suffixe %. Nous verrons cette représentation après avoir introduit les nombres algébriques (avec signe).

LES OPERATIONS

Arithmétiques

Le **microprocesseur 65Ø2** effectue **l'addition** et la **soustraction.** Cette dernière opération se transforme en addition grâce à la représentation des nombres négatifs ou algèbriques, appelés nombres avec signe, encore appelés nombres signés par certains auteurs (**tableau 3**).

La base étant B, on appelle complément restreint d'un nombre, son complément à B-1, et complément vrai, son complément à B.

Ceci étant défini, pour représenter un nombre avec signe, nous prenons la convention suivante : le bit de poids le plus haut sera un bit de signe : Ø pour un nombre positif, 1 pour un négatif.

Binaire B=2	*Décimal B=1Ø*	*Hexa B=16*	*Binaire B=2*	*Décimal B=1Ø*	*Hexa B=16*
1ØØØØØØØ	-128	8Ø	ØØØØØØØØ	Ø	Ø
1ØØØØØØ1	-127	81	ØØØØØØØ1	1	1
1ØØØØØ1Ø	-126	82	ØØØØØØ1Ø	2	2
1ØØØØ11	-125	83	"	"	"
"	"	"	"	"	"
"	"	"	Ø1111101	125	7D
11111110	-2	FE	Ø1111110	126	7E
11111111	-1	FF	Ø1111111	127	7F

Tableau 3 - représentation des nombres avec signe

Ensuite, chaque nombre négatif est représenté comme le complément vrai du nombre positif correspondant. Un octet pourra représenter un nombre avec signe de -128 à 127 et deux octets un nombre de -32768 à 32767.

Remarque : si, dans le programme du paragraphe précédent, nous remplaçons la ligne 1Ø par celle-ci :

```
10 INPUT A%
```

vous aurez la représentation des nombres entiers par l'ORIC. Les 2 premiers octets représentent le nombre, les 3 octets suivants sont tous nuls.

RUN puis entrons 1ØØØ, l'écran affiche :

```
 3  232  Ø  Ø  Ø
```

le premier octet est celui de poids fort, le 2e de poids faible

```
3.256 + 232 = 1ØØØd
```

Comme l'ORIC n'emploie que 2 octets, les limites sont -32768 à 32767 en représentation avec signe.

La manipulation des nombres entiers est, en principe, plus rapide que celle des nombres fractionnaires.

Logiques

Le complément restreint

C'est le **complément à B-1** (B étant la base) effectué bit à bit. Le complément restreint de A s'écrit $\bar{A}$ et se lit A barre.

Exemple

si A = Ø1111lØ1
alors
$\bar{A}$ = 1ØØØØØ1Ø

Le complément vrai

C'est le **complément à B** ou, plus simplement, c'est le complément restreint plus 1.

Dans l'exemple précédent, le complément vrai de A est

$\bar{A}$ + 1 = 1ØØØØØ11

Si vous préférez c'est le résultat de l'opération Ø - A. Remarquez que A = 125 et que $\bar{A}$ + 1 = -125 selon le tableau 3. Pour passer de l'un à l'autre, on prend son complément vrai.

L'opération ET

Cette opération et les deux suivantes sont réalisées directement par le microprocesseur.

L'opération ET s'effectue sur deux opérandes bit à bit. On écrit AΛB (lire A et B)

Exemple

A = 1Ø1ØØ111
B = ØØØØ1111
AΛB = ØØØØØ111

Il suffit d'appliquer les règles suivantes pour chaque bit :

ØΛØ = Ø
ØΛ1 = Ø
1ΛØ = Ø
1Λ1 = 1

L'opération OU

Cette opération s'effectue également bit à bit, sur deux opérandes. On écrit A V B (lire A ou B).

Exemple

```
        A = ØØ11ØØ11
        B = 1ØØØ1ØØØ
  A V B = 1Ø111Ø11
```

en appliquant les règles suivantes :

```
Ø V Ø = Ø
Ø V 1 = 1
1 V Ø = 1
1 V 1 = 1
```

L'opération OU exclusif

Egalement sur deux opérandes, bit à bit. On écrit A ⊻ B (lire A ou exclusif B).

Exemple

```
        A = Ø1Ø1Ø1Ø1
        B = 1Ø1ØØ1Ø1
  A ⊻ B = 1111ØØØØ
```

en appliquant les règles :

```
Ø ⊻ Ø = Ø
Ø ⊻ 1 = 1
1 ⊻ Ø = 1
1 ⊻ 1 = Ø
```

LE SYSTEME ORIC

La **figure 1** montre les principaux composants de l'ORIC :

- *le microprocesseur 65Ø2A*, piloté par un cristal, fournit 2 signaux d'horloge phi 1 et phi 2, presqu'identiques, pour cadencer et synchroniser les signaux des composants. Le paragraphe suivant lui sera consacré.
- *la mémoire vive* (ou RAM) s'étend sur 16 K ou sur 48 K d'octets suivant les modèles. Un K (abréviation de kilo) signifie, en informatique, 2^{10} ou 1Ø24 emplacements mémoire. Dans les micro-ordinateurs, un emplacement mémoire stocke 8 bits ou un octet. Chaque emplacement mémoire possède une adresse. Pour le modèle 16 K, les adresses vont de Ø à 16383d -Ø à 3FFFh- et pour le modèle 48 K, les adresses vont de Ø à 49151d -Ø à BFFFh -.

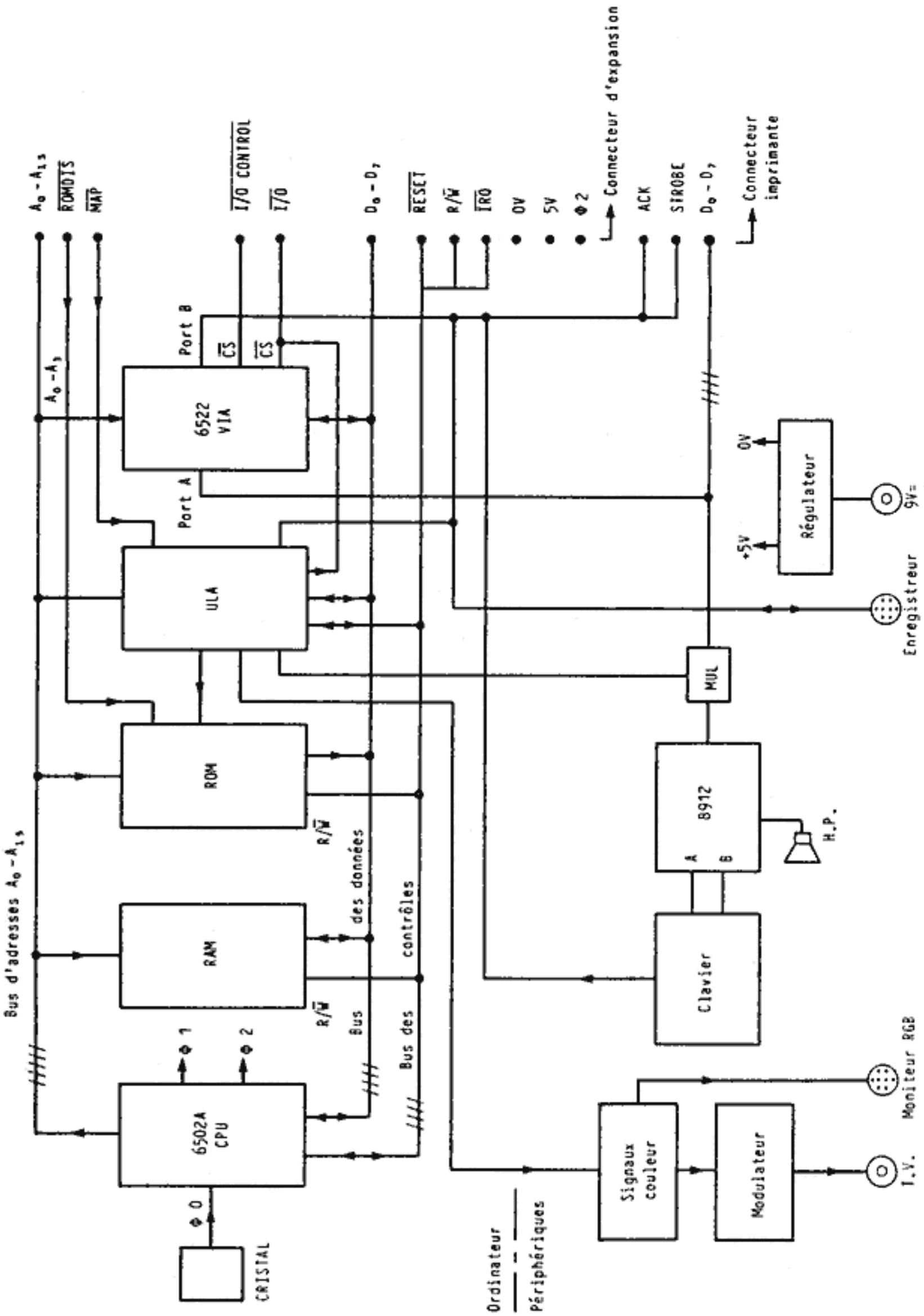


FIGURE 1 - Synoptique de l'ORIC

Nous emploierons des parenthèses pour indiquer le co tenu d'une adresse. Ainsi (1ØØØØd) = 128d signifie que l'emplacement mémoire d'adresse 1ØØØØ contient 128. Nous emploierons également des parenthèses doubles qui signifient : contenu du contenu de. Ainsi ((TXTTAB)) = 85 signifie que (TXTTAB) = x et (x) = 8! TXTTAB est le nom d'une variable système qui correspond à une adresse.

- *la mémoire morte* (ou ROM) contient 16 K d'octets, au: adresses 49152d à 65535d - CØØØh à FFFFh. Nous trouvons dans cette mémoire figée un programme moniteur qui permet le démarrage du système, un interpréteur Basic qui donne à l'usager un langage Basic évolué, contenant de nombreuses commandes et fonctions, et de nombreuses routines d'exploitation. Ces routines sont implantées en codes machine- nous écrirons C.M. par la suite- que l'interpréteur appelle au fur et à mesure de ses besoins. Ces routines n'ont malheureusement pas les mêmes points d'entrée pour l'ORIC ATMOS et pour l'ORIC-1.

- *l'ULA* est un circuit qui intègre beaucoup de logiques utiles pour le fonctionnement. C'est un circuit intégré fabriqué spécialement pour Tangerine pour ses ordinateurs ORIC.

- *le VIA 6522* est un circuit intégré d'entrées/sorties. Il gère une multitude de fonctions dans l'ordinateur. Ce C.I. contient 16 registres que l'on programme pour l'amener à réaliser les fonctions d'entrées/sorties. Les connecteurs d'expansion et d'imprimante, disponibles à l'arrière de l'ordinateur, permettent à l'usager, nanti de bonnes notions d'électronique, d'employer le VIA pour communiquer avec le monde extérieur. Cet usage ne rentre pas dans le cadre de ce livre. Vous trouverez des exemples et des schémas dans le manuel de l'ORIC ATMOS.
  Il est évident que le connecteur d'imprimante permet le raccordement de l'imprimante ORIC sans interface supplémentaire. Tout le Basic d'exploitation se trouve dans la MEM (ROM).

- *les interfaces* nécessaires pour raccorder un moniteur couleurs et, ou, une T.V. couleurs ; et une prise pour enregistreur pour sauvegarder et reproduire les programmes, tableaux, etc. que vous aurez réalisés.

- *un C.I. 8912* qui permet de synthétiser 3 canaux de sons et 1 de bruit, avec sortie sur ampli et H.P. incorporés ou sur un ampli extérieur. Le 8912 contient

également 2 ports utilisés pour le périphérique suivant.

- *le clavier* qui est de tous le périphérique le plus utile. Il n'y en aurait qu'un, ce serait celui-là.

LE MICROPROCESSEUR 65Ø2A

La **figure 2** donne l'organisation interne de ce microprocesseur très puissant.

Nous trouvons **3 bus** qui sortent du microprocesseur pour communiquer avec les circuits externes. Un bus est une réunion de conducteurs imrimés sur un circuit.

Le **bus d'adresses** comporte 16 conducteurs numérotés **AØ** à **A15.** Une configuration de bits sur ce bus fournit une adresse mémoire. En réalité, ce sont les bits d'un registre double qui sont appelés AØ à A15, mais il n'y a aucune confusion à donner ces mêmes numéros aux conducteurs du bus.

Le **bus des contrôles** comporte 7 conducteurs dont les noms sont **RDY**, **SYNC**, **SO**, **$\overline{\text{NMI}}$**, **$\overline{\text{IRQ}}$**, **$\overline{\text{RESET}}$** et **R/$\overline{\text{W}}$.** Ce bus envoie ou reçoit des signaux qui informent ou modifient le fonctionnement du microprocesseur. La barre indique l'activité au niveau Ø.

RDY sert à s'interfacer avec des mémoires lentes. Il peut rafraîchir des mémoires dynamiques et effectuer des DMA. RDY arrête le 65Ø2 lors de tout cycle de lecture mais non lors d'un cycle d'écriture.

SYNC est en synchronisme avec R/$\overline{\text{W}}$. Il est à 1 lors de tout cycle de recherche d'instruction.

SO limite les possibilités d'entrées série.

$\overline{\text{NMI}}$ interruption non masquable.

$\overline{\text{IRQ}}$ demande d'interruption.

$\overline{\text{RESET}}$ remise à Ø.

R/$\overline{\text{W}}$ signal de lecture/écriture, dirige les directions des transferts sur le bus des données.

Les 3 derniers signaux sont accessibles sur le connecteur d'expansion situé à l'arrière de l'ORIC.

Le **bus des données** comporte 8 conducteurs **DØ** à **D7.** Les données entrent dans le microprocesseur pour y être

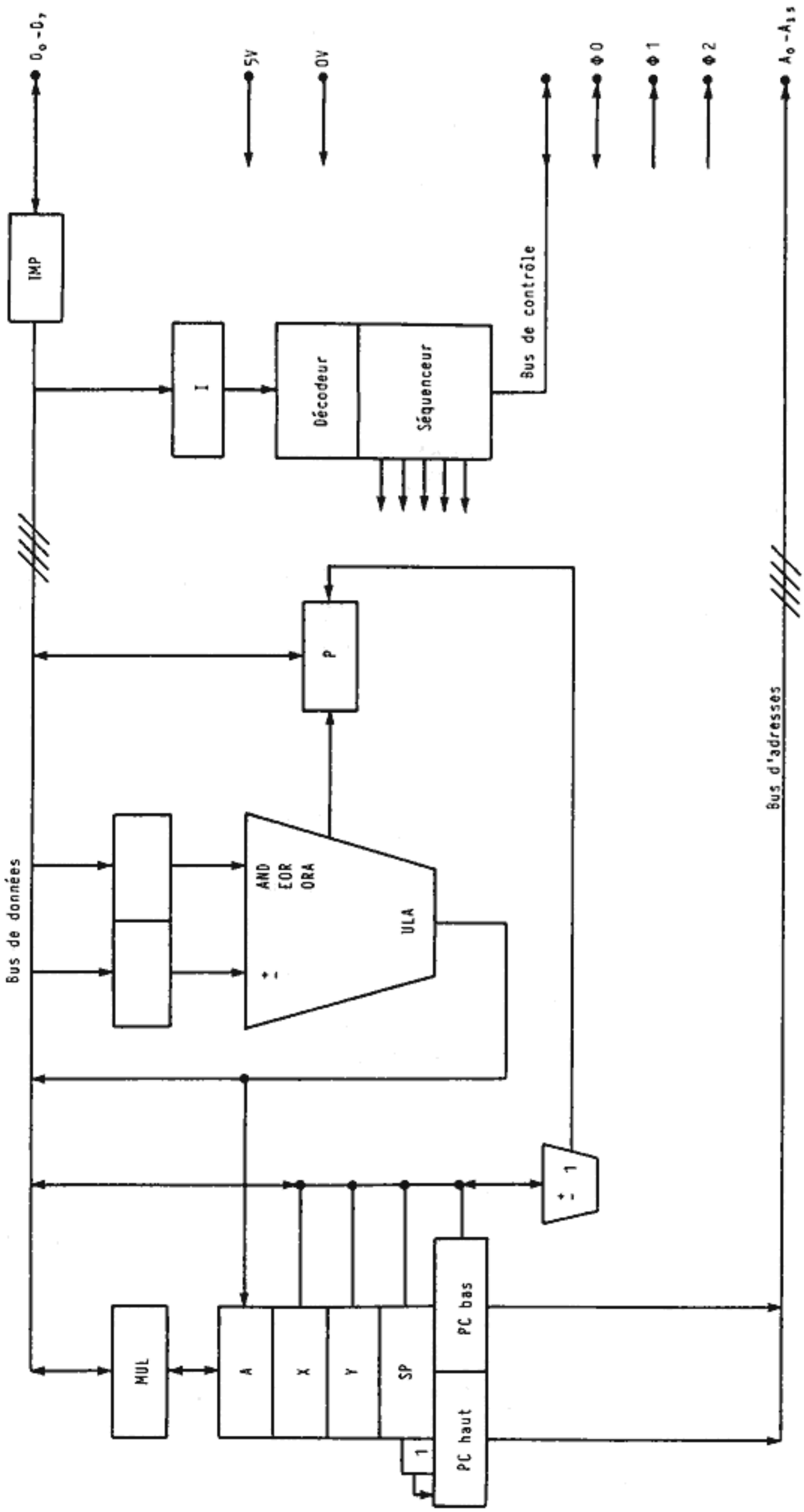


FIGURE 2 - Organisation interne du microprocesseur 6502 A

traitées, elles en sortent pour être stockées en mémoire. Le bus des données est contrôlé par un registre tampon **TMP** qui le met dans 3 états possibles : Ø, 1 et haute impédance. Ce dernier état isole le bus des données pendant que le microprocesseur effectue des opérations internes.

Il y a une **horloge incorporée** qui ne nécessite qu'une entrée phi Ø. En sortie le 65Ø2A fournit phi 1 et phi 2, deux phases à peu près égales qui constituent un cycle. Pendant phi 1, on prépare une adresse et pendant phi 2, on y accède. Le bus des données est en haute impédance pendant phi 1 et actif pendant phi 2.

Nous trouvons également des **tampons** sur les deux entrées de l'**ULA**, Unité Logique et Arithmétique, qui effectue les opérations.

Le **multiplexeur MUL** sélectionne un des registres situés en dessous de lui.

Les **différents registres**, à l'exception du registre PC, contiennent 8 bits. Quel que soit l'arrangement de ces bits (instruction, nombre, caractère, attribut, etc.) on peut toujours le considérer comme un nombre binaire pur.

Voyons l'usage de ces différents registres :

Le registre I

Le registre d'instructions I, reçoit une instruction, la décode puis l'exécute en envoyant des signaux aux différents composants du microprocesseur et également vers l'extérieur par le bus des contrôles. Le nombre de cycles d'horloge T requis pour exécuter l'instruction, dépend de cette instruction, il varie de 2 à 7 cycles d'horloge.

Le registre A

Le registre A ou accumulateur, est le registre le plus utilisé. Son usage est obligatoire pour les opérations arithmétiques et logiques. Il est encore prépondérant pour beaucoup d'autres instructions.

Le registre X

Le registre X s'emploie pour réaliser toutes sortes d'opérations. En premier lieu, il est prévu comme index pour les modes d'adressage indexés. Il s'emploie ensuite pour réaliser des boucles, pour traiter des nombres de deux octets, pour des comparaisons, etc.

Le registre Y

Le registre Y s'emploie de la même manière que le registre X, avec cependant un peu moins de possibilités que celui-ci. Les registres A, X et Y sont les registre. de travail du microprocesseur.

Le registre S ou SP

Le registre S s'emploie comme pointeur de pile. Il pointe sur un emplacement mémoire réservé qui sert de pile au microprocesseur. Cette pile fonctionne suivant le principe LIFO (Last In First Out) : la dernière information entrée sera la première à sortir. Le sommet de la pile possède l'adresse 1FFh -511d- et on empile les informations aux adresses directement inférieures. Donc, quand on pousse un octet dans la pile, SP est décrémenté et, quand on le retire, SP est incrémenté automatiquement.

Le registre PC

Le registre PC ou compteur ordinal est le seul registre de 16 bits. Il dépose une adresse sur le bus d'adresses. Le PC pointe sur l'instruction à exécuter. Après exécution, le PC est incrémenté, plusieurs fois si nécessaire, pour pointer sur l'instruction suivante.

Le registre P

Le registre P est un registre spécial. Les bits qu'il contient servent d'indicateurs d'état. Chaque indicateur est mis à 1 ou à Ø suivant le résultat de l'opération effectuée. La **figure 3** montre l'emplacement et les noms de ces indicateurs.

7	6	5	4	3	2	1	Ø	poids
N	V	-	B	D	I	Z	C	

Figure 3. Indicateurs du registre P

L'indicateur C est l'indicateur de report (addition) ou de retenue (soustraction). Il est mis à 1 quand le résultat est modulo 256.

L'indicateur Z indique la nullité. Il est mis à 1 quand le résultat de l'opération est Ø.

L'indicateur I est l'indicateur d'interruption. Il est mis à 1 quand une interruption survient.

L'indicateur D indique l'emploi du code DCB quand on le met à 1 et du code binaire quand on le met à Ø.

L'indicateur B se met à 1 quand l'interruption est provoquée par un break.

L'indicateur V est l'indicateur de dépassement. Il est mis à 1 quand il y a dépassement de capacité de calcul sur des nombres avec signe. On reviendra sur cet indicateur quand on examinera les codes machine de l'addition.

L'indicateur N est l'indicateur de signe des nombres avec signe. Il est communément la réplique du bit 7 du nombre résultant d'une opération.

Pagination

Puisque le registre PC possède 16 bits, il peut adresser 65536 emplacements mémoire. On peut considérer tout cet espace mémoire comme 256 pages de 256 emplacements ou cellules mémoire. Remarquons que le registre PC tient dans son octet bas, appelé PC bas, l'adresse de 256 cellules dont la page est tenue dans son octet haut, PC haut.

La notion de page est très importante pour le microprocesseur 65Ø2.

La page Ø dans laquelle se trouvent les pointeurs et variables les plus utilisées par le système, possède le privilège d'effectuer plus vite les instructions du 65Ø2 qui la concernent. La page Ø contient également des tampons, des adresses et même une routine importante.

La page 1 est réservée à la pile. Automatiquement, quand une opération de pile est effectuée, le microprocesseur met 1 comme page (son PC haut contient 1). Pour cette raison, nous avons mis 1 à gauche du registre SP dans la figure 2. Ceci implique que la pile ne se compose que de 256 octets, ce qui est bien suffisant. Certaines routines remettent d'ailleurs le pointeur de pile au sommet de celle-ci.

La page 2 contient des variables système, des adresses, etc... nécessaires au système.

La page 3 contient les informations qui dirigent le fonctionnement du VIA 6522, unité d'entrées/sorties et de timers.

La page 4 est réservée en partie pour les besoins du système.

Aux *pages 5* et suivantes commence la zone du programme Basic, suivie de la zone des variables, etc.

Disons encore que certaines instructions indexées prennent plus de temps pour s'exécuter quand on passe d'une page à l'autre.

Adressage

Certaines instructions du microprocesseur ont recours à différentes méthodes d'adressage de la mémoire. Ces différents adressages possibles sont potentiellement fort utiles pour utiliser rationnellement les possibilités des différentes instructions. Il est donc nécessaire de bien les comprendre.

Adressage immédiat

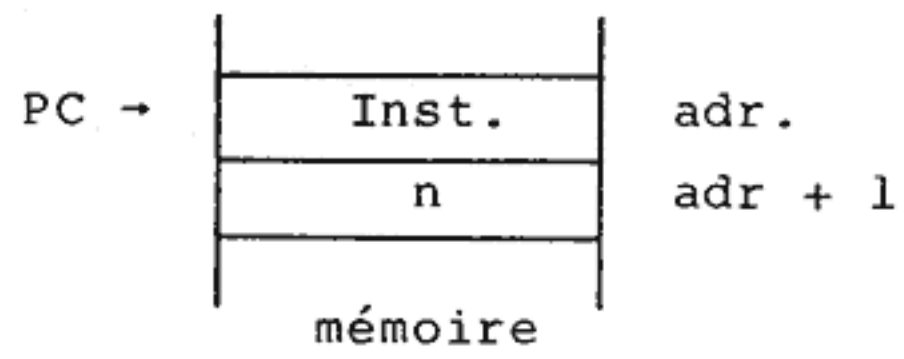


Figure 4

Le PC pointe sur adr. qui contient l'instruction. Celle-ci nécessite une donnée immédiate n située à adr + 1.

Adressage absolu

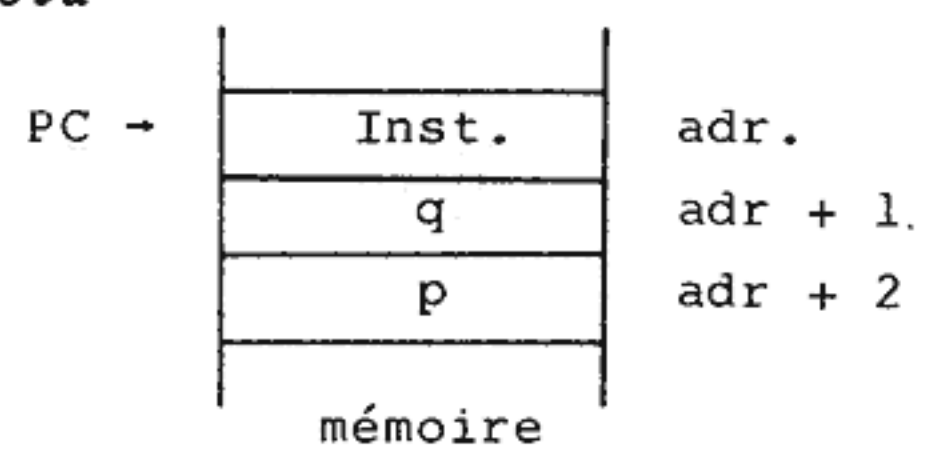


Figure 5

Le PC pointe sur adr. qui contient l'instruction. Celle-ci a besoin d'une adresse pq où se trouve la donnée à traiter.

Adressage page Ø

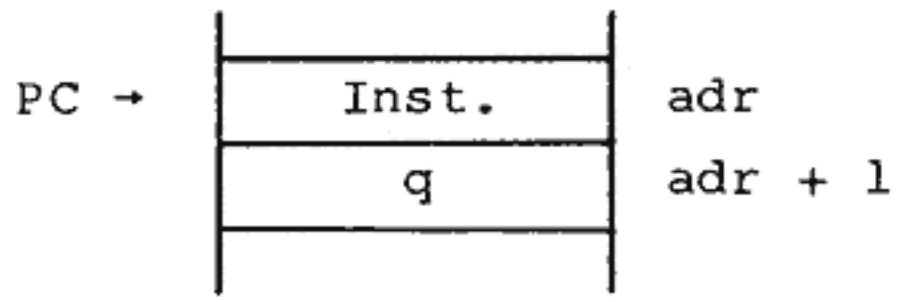


Figure 6

Le PC pointe sur adr pour en extraire l'instruction. Celle-ci a besoin d'une donnée située en page Ø, à l'adresse Oq.

Adressage absolu indexé par X

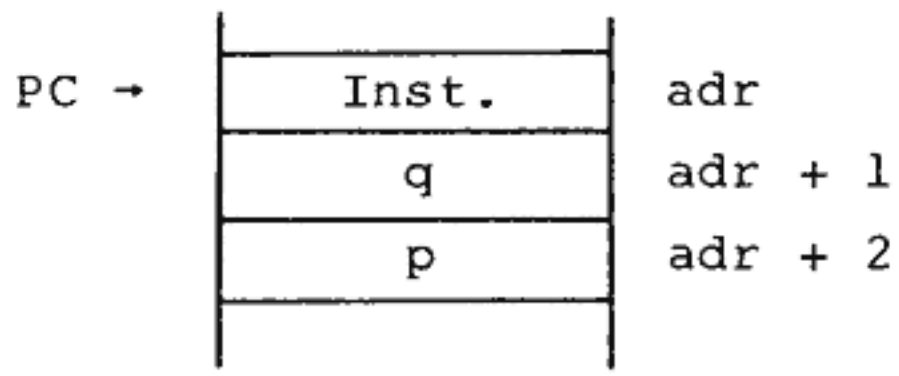


Figure 7

Le PC extrait l'instruction qui a besoin d'une donnée située à pq + X.

Adressage absolu indexé par Y

Comme ci-dessus mais l'indexation se fait par Y. La donnée est située à l'adresse pq + Y.

Adressage page Ø indexé par X

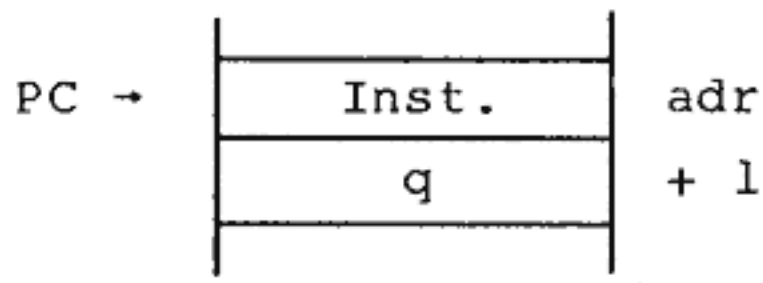


Figure 8

La donnée nécessaire pour réaliser l'instruction est située à l'adresse oq + X.

Adressage indirect

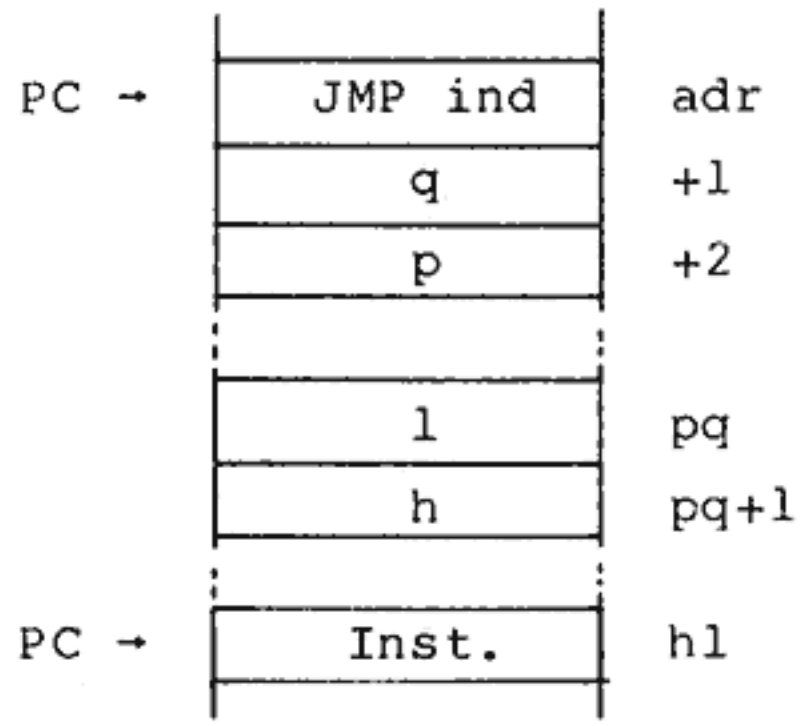


Figure 9

Les 2 octets qui suivent l'instruction, contiennent l'adresse pq. C'est aux adresses pq et pq+1 que se trouve l'adresse à utiliser. Il n'y a qu'une instruction de ce genre : le saut indirect. Le résultat est que PC=hl Le PC pointe sur hl pour extraire l'instruction suivante.

Adressage indirect pré-indexé par X

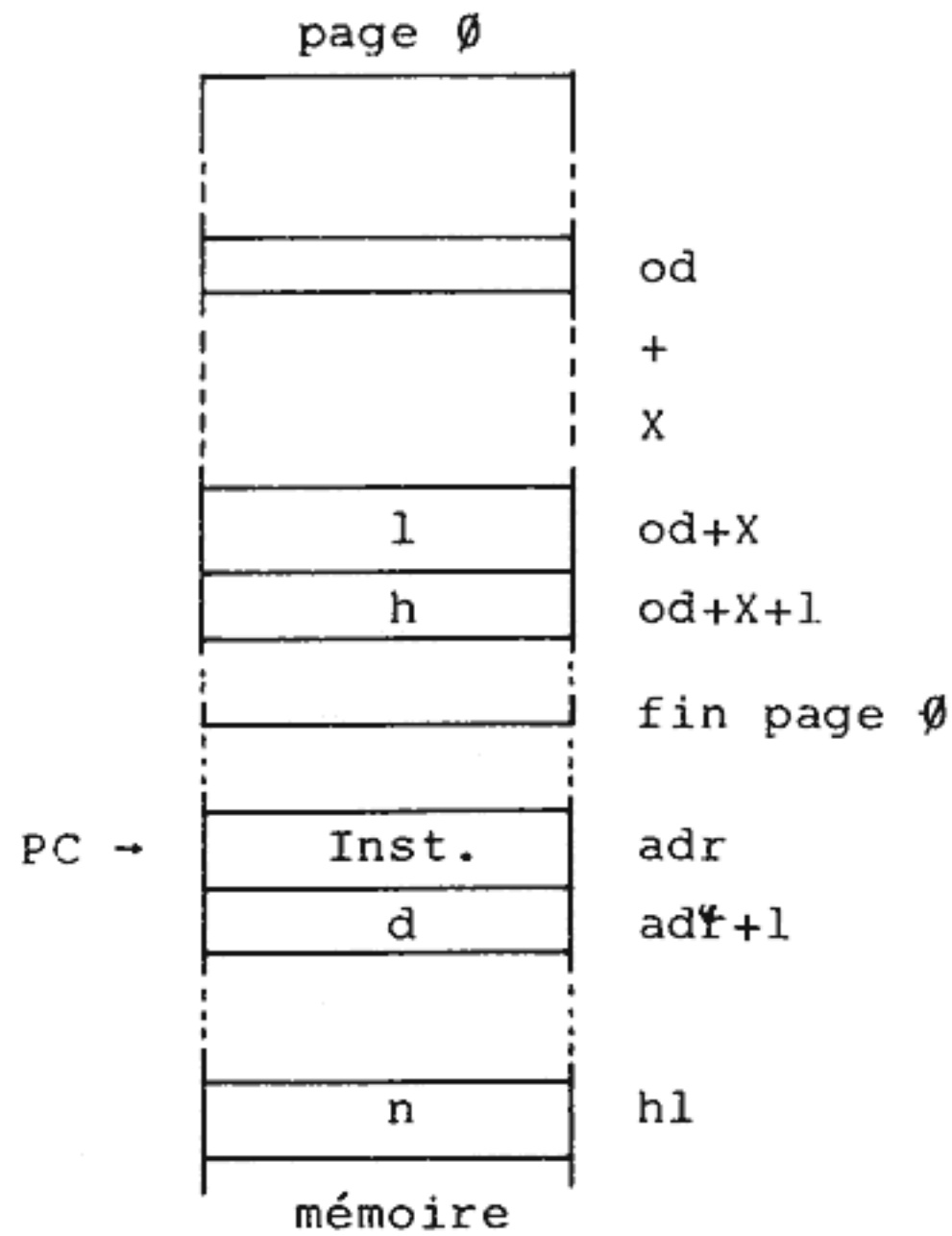


Figure 1Ø

Le PC extrait l'instruction. L'octet suivant contient l'adresse od. On ajoute X à od. Nous avons une nouvelle adresse en page Ø où nous trouvons, sur 2 octets, l'adresse d'indirection hl. C'est à l'adresse hl que se trouve la donnée n à traiter par l'instruction.

Adressage indirect post-indexé par Y

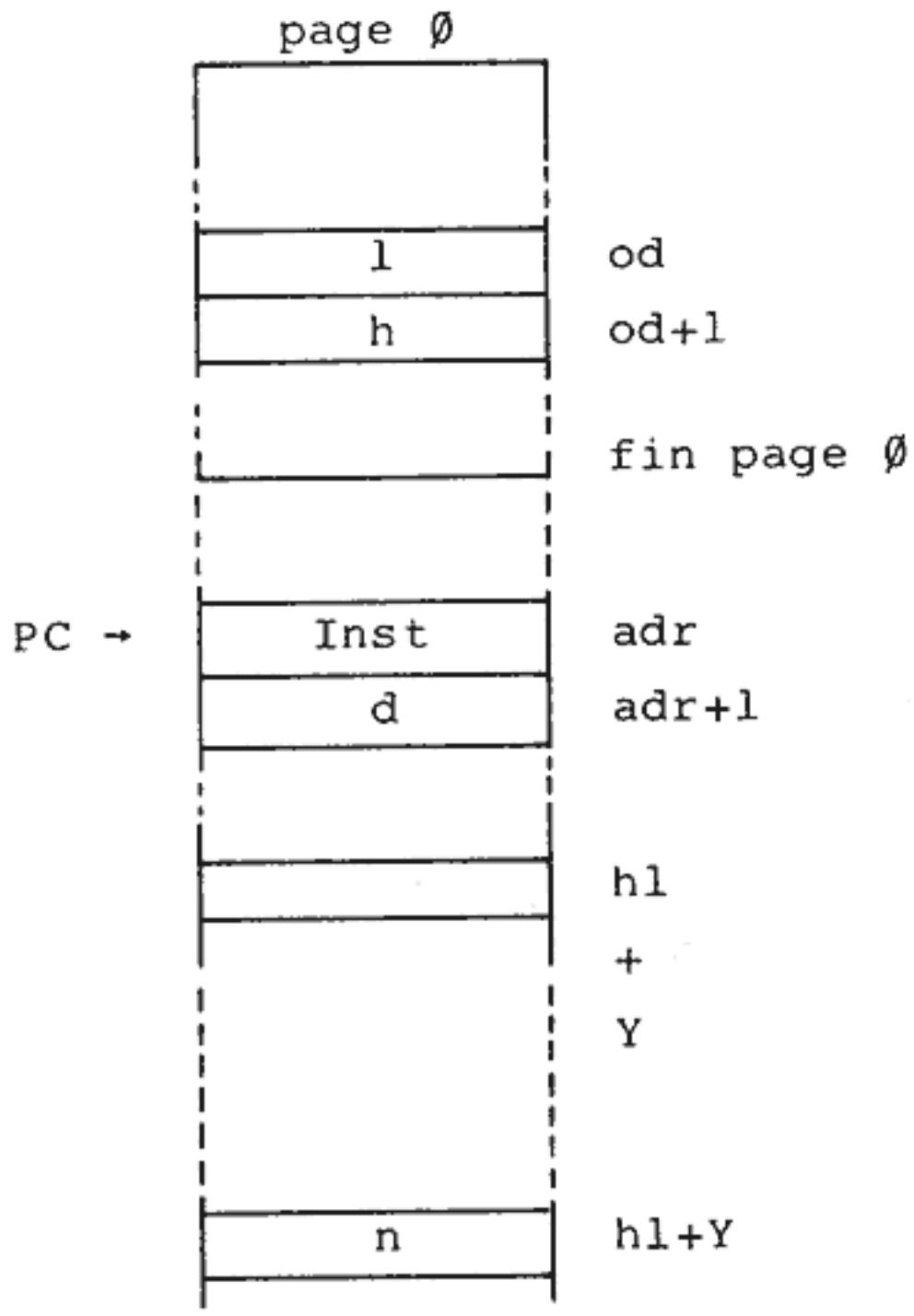


Figure 11

Le PC extrait l'instruction à exécuter. L'octet suivant contient l'adresse d en page Ø. Aux adresses od et od+1 se trouve l'adresse d'indirection hl. Celle-ci est post-indexée par Y, et la donnée n se trouve à l'adresse hl+Y.

Note : Vous avez remarqué que c'est chaque fois le code de l'instruction, qui est transféré dans le registre I, qui déclenche la suite des opérations.

LES CODES MACHINE

Les codes machine représentent des instructions en langage machine que le microprocesseur comprend. Chaque instruction reçue par le microprocesseur lui fait exécuter une ou plusieurs opérations. Chaque instruction est représentée par un mnémonique, utilisé également par les différents programmes assembleur du commerce ; avec cependant des différences de ponctuation d'un auteur à l'autre. Le mnémonique est suivi par un opérande, s'il y a lieu, dont l'écriture et la syntaxe nous est propre. De toute façon, tous les programmes assembleur ont leurs propres syntaxes et si vous en achetez un, vous devez apprendre cette syntaxe-là pour l'utiliser.

A partir du chapitre II, chaque mnémonique est expliqué, en donnant l'instruction qu'il représente, son code machine et son influence sur les indicateurs d'état. Toutes ces explications sont reprises en annexe, par ordre alphabétique et numérique des mnémoniques. Chaque mnémonique comporte 1, 2 ou 3 octets de codes machine, selon l'instruction.

Prenons un exemple : chargeons 5 dans le registre A. L'instruction **A=5** a pour mnémonique **LDA 5** et pour **C.M.**, **A9,Ø5**. Cette instruction nécessite 2 octets. La déclaration Basic 1Ø LET A=5 demande 1Ø octets en zone mémoire programme et 7 octets en zone mémoire variables.

Pour représenter les C.M., nous employons un tableau divisé en 5 colonnes, une ou plusieurs colonnes peuvent rester vierges.

Adresse	*C.M.*	*Etiquette*	*Mnémonique*	*Commentaires*
2ØØØØ	A9,Ø5		LDA 5	

Tableau 4 - présentation des codes machine

La *colonne 1*, adresse, est écrite en décimale, la *colonne 2* en hexadécimale et la *colonne 4* dans l'une ou l'autre des numérations. Les *colonnes étiquette* et *commentaires* seront utilisées si nécessaire.

Seules les 2 premières colonnes sont indispensables : nous devons introduire les codes de la colonne 2 aux adresses de la colonne 1.

Programmes de chargement des codes machine

Programme 1

```
10 LET T=1024
20 INPUT A$
30 IF A$="S" OR T=1056 THEN STOP
40 POKE T,VAL("#"+A$)
50 PRINT T,A$
60 T=T+1
70 GOTO 20
```

Le programme 1 nous servira pour expérimenter quelques petits programmes en C.M..

Pour arrêter ce programme, entrez S.

Programme 2

```
5010 HIMEM 37887
5020 CLS
5030 INPUT "ENTREZ L'ADRESSE";A
5040 PRINT :PRINT "ENTREZ VOS C.M. PAR
     GROUPE DE" :PRINT "2 CHIFFRES HEXA."
5050 PRINT :PRINT "POUSSEZ SUR UNE TOUCHE"
5060 GET A$ :CLS
5070 PRINT :PRINT
5100 PLOT 0,0,23 :PLOT 1,0,0 :PLOT 2,0,12
5110 PLOT 3,0,"     ENTREZ 2 CHIFFRES
     HEXA.      "
5120 PLOT 0,1,8:PLOT 1,1,8
5130 PLOT 2,1,"ADRESSE      HEXA    DEC
     ENTREE"
5140 PRINT TAB(30):INPUT A$
5150 IF A$="S" THEN STOP
5160 IF A$="E" THEN A=A-1:GOTO 5100
5170 IF A$="L" THEN 5240
5180 IF LEN(A$)<>2 THEN 5100
5190 IF LEFT$(A$,1)>"F"OR RIGHT$(A$,1
     )>"F"THEN 5100
5195 LET B=VAL("#"+A$)
5200 POKE A,B
5210 PRINT A;TAB(14);A$;TAB(21);B;
5220 LET A=A+1
5230 GOTO 5100
5240 LET B=PEEK(A)
5250 LET A$=HEX$(B)
5260 GOTO 5210
```

Les lignes Basic ont été chiffrées à partir de 5Ø1Ø pour que vous puissiez mettre votre programme en-dessous. Après la mise au point, vous pouvez effacer ce programme de chargement ou ajouter la ligne **5ØØØ STOP** pour éviter que l'ordinateur ne l'exécute à la suite du vôtre, à moins que vous n'ayez mis **END** ou **STOP** à la fin de votre programme.

Après avoir demandé l'adresse de chargement, l'ordinateur vous prie d'entrer vos C.M.. Si vous entrez un, trois ou plus de caractères, il n'arrive rien. Il en sera de même si vous entrez deux chiffres non hexadécimaux comme ER ou G8.

Pour lister votre programme ou une routine de la mémoire, vous entrez **L** ; vous avez alors le contenu de l'adresse.

Si vous vous êtes trompé en entrant des C.M., entrez **E** pour effacer et revenir en arrière d'un pas. Appuyez plusieurs fois sur **E** pour revenir de plusieurs pas.

Pour arrêter ce programme, entrez **S**.

CHAPITRE 2
OPÉRATION DE BASE

Les deux programmes Basic qui terminent le chapitre I nous permettent d'introduire des C.M. dans la mémoire de l'ORIC. Dans ce chapitre, nous examinons tout d'abord, *comment exploiter les programmes en C.M.*.

Nous présentons ensuite les *instructions de chargement* qui sont les plus importantes et les plus fréquemment utilisées. Ces instructions réalisent l'affectation de valeurs dans les registres du microprocesseur et dans la mémoire de l'ORIC. Nous devons constamment prélever des valeurs, des données de la mémoire, les introduire dans le microprocesseur pour y être traitées et stocker les résultats dans la mémoire.

Nous terminons le chapitre en décrivant les *opérations de traitement les plus usuelles* qui sont l'addition, l'incrémentation, la soustraction et la décrémentation. Des exemples et des exercices sont donnés et proposés pour vérifier les connaissances acquises.

EXPLOITATION DES PROGRAMMES EN C.M.

Lorsque un programme a été introduit ou, que l'on utilise un programme de la MEM (ou ROM), pour lancer son exécution, nous employons l'instruction Basic **CALL** suivi de l'adresse mémoire où commence le programme. Le paramètre de CALL est donc compris entre Ø et 65535. Ceci montre également que nous avons besoin d'une ligne Basic

chaque fois que l'on exécute un programme en C.M.. Il existe trois autres fonctions et instruction pour exploiter les C.M.. Nous les verrons dans le chapitre VI, quand nous aurons acquis un bagage suffisant.

Pour que l'ordinateur sache où se termine notre programme en C.M., la dernière instruction de celui-ci sera une instruction de retour.

Le mnémonique de cette instruction est **RTS** et son C.M. est 6Øh ou 96d.

Les opérations que ce C.M. déclenche dans le microprocesseur seront expliquées plus loin (il fait appel à des notions non encore précisées). Sachons pour l'instant qu'il est nécessaire. Voici un programme, très court, qui l'utilise :

```
1Ø24    6Ø                  RTS
```

Il est inutile d'employer un programme de chargement pour introduire ce C.M., faites simplement la commande :

```
POKE 1Ø24,96
```

pour l'introduire à l'adresse 1Ø24.

Pour appeler ce programme, faites la commande :

```
CALL 1Ø24
```

Evidemment il ne se passe rien ; à peine appelé, le programme se termine déjà et vous avez le message habituel Ready.

Seules les deux premières colonnes comptent pour l'introduction des C.M., le reste rend compréhensible la ou les opérations effectuées ; le mnémonique pour le C.M. employé et les commentaires pour le but recherché.

RTS signifie, en fait, retour à celui qui a appelé. Ici c'est une commande Basic qui a appelé, et par conséquent le retour se fait dans le système Basic.

Cette instruction n'a aucune influence sur les indicateurs d'état.

LES CODES MACHINE DE CHARGEMENT

Chargement d'une constante dans un registre

A9,nn	LDA n	LET A=n

Le mnémonique LDA n réalise l'opération :

A=n

ce que nous montrons dans la correspondance Basic LET A=n où n est une constante numérique.

Nous mettons A9,nn pour indiquer qu'il y a deux octets à entrer. Nous représentons le deuxième octet par nn pour signaler deux chiffres hexadécimaux à entrer. Prenons cette habitude dès le début.

Nous avons de même :

A2,nn	LDX n	LET X=n

qui réalise X=n

AØ,nn	LDY n	LET Y=n

qui réalise Y=n

Ces trois instructions influencent les indicateurs d'état **N** et **Z**. Ainsi :

A9,1Ø	LDA 16d	LET A=16

met N et Z à Ø

A2,ØØ	LDX Ø	LET X=Ø

met l'indicateur Z à 1

AØ,8Ø	LDY 128d	LET Y=128

met l'indicateur N à 1

Chargement d'un registre par un registre

AA	TAX	LET X=A

réalise X=A

Nous disposons de même :

A8	TAY	LET Y=A
8A	TXA	LET A=X
98	TYA	LET A=Y

Ces instructions réalisent le transfert d'un registre dans un autre. Elles influencent les indicateurs d'état **N** et **Z**.

Si vous voulez charger A et X, tous deux avec la valeur 5, faites :

```
A9,Ø5          LDA 5
AA             TAX
```

Si vous employez LDX 5, vous utilisez un octet de plus, sans autre préjudice.

Chargement d'un registre par le contenu d'une adresse

- ***Absolu***

```
AD,qq,pp       LDA pq          LET A=PEEK(pq)
réalise        A=(pq)
```

charge **A** avec le contenu de **pq**. Les parenthèses employée dans le Basic correspondant n'ont pas la même signification que celles de l'instruction en C.M., elles sont nécessaires pour l'argument de PEEK.

IMPORTANT : Dans les C.M., il faut **introduire d'abord** l'octet bas qq puis l'octet haut pp de l'adresse. C'est toujours de cette manière qu'une adresse est stockée dans la mémoire. Ainsi :

```
AD,ØØ,Ø5       LDA (5ØØh)      ;(128Ød)
```

charge A avec le contenu de la cellule d'adresse 5ØØh-128Ød.

Nous avons de même :

```
AE,qq,pp       LDX pq          LET X=PEEK(pq)
AC,qq,pp       LDY pq          LET Y=PEEK(pq)
```

Pour l'adressage en page Ø, on n'emploie que l'octet bas, l'ordinateur sait que l'octet haut est ØØ.

```
A5,qq          LDA q           LET A=PEEK(q)
réalise        A=(Oq)

A6,qq          LDX q           LET X=PEEK(q)
A4,qq          LDY q           LET Y=PEEK(q)
```

Ces instructions influencent les indicateurs **N** et **Z**.

- ***Indexé***

```
BD,qq,pp       LDA pq,X        LET A=PEEK(pq+X)
réalise  A=(pq+X)

B9,qq,pp       LDA pq,Y        LET A=PEEK(pq+Y)
```

```
     BE,qq,pp          LDX pq,Y              LET X=PEEK(pq+Y)
     BC,qq,pp          LDY pq,X              LET Y=PEEK(pq+X)
```

pour la page Ø, nous avons

```
     B5,qq             LDA q,X               LET A=PEEK(q+X)
```

ici pas d'indexation par Y

```
     B6,qq             LDX q,Y               LET X=PEEK(q+Y)
     B4,qq             LDY q,X               LET Y=PEEK(q+X)
```

Ces instructions influencent les indicateurs **N** et **Z.**

- Indirect indexé

```
     A1,dd             LDA d,X               HL=DEEK(d+X)
réalise plusieurs opérations                 A=PEEK(HL)
        l=(od+X)
        h=(od+X+1)
        A=(hl)

     B1,dd             LDA d,Y               HL=DEEK(d)
réalise plusieurs opérations                 A=PEEK(HL+Y)
        l=(od)
        h=(od+1)
        A=(hl+Y)
```

Ces deux instructions indirectes pré- et post-indexées ne travaillent qu'avec la page Ø. Elles nécessitent la présence d'une adresse en page Ø dans 2 octets consécutifs. Elles influencent les indicateurs **N** et **Z.**

Notez que ces deux instructions réalisent chacune trois opérations.

Chargement d'une cellule mémoire par un registre

- Absolu

```
     8D,qq,pp          STA pq                POKE pq,A
réalise                (pq)=A
```

Nous avons de même

```
     8E,qq,pp          STX pq                POKE pq,X
     8C,qq,pp          STY pq                POKE pq,Y
```

et pour la page Ø

```
     85,qq             STA q                 POKE q,A
     86,qq             STX q                 POKE q,X
     84,qq             STY q                 POKE q,Y
```

Ces instructions n'ont aucune influence sur les indicateurs.

Exemple 1 : Nous expliquerons l'écran **basse résolution** dans le chapitre III et **haute résolution** dans le chapitre IV. Comme nous l'employons, dès à présent, en basse résolution, disons que l'affichage commence à 48ØØØd -BB8Øh- pour un ORIC 48 K et à 15232d -3B8Øh- pour un 16K. (Les adresses hautes d'un 16K sont celles d'un 48K moins 32768). Comme la première ligne est réservée, ainsi que les deux premières colonnes (nous y logerons des C.M. plus tard), nous affichons à partir de la position 48Ø42, le mot SALUT. Employez le programme 1 du chapitre I pour entrer les C.M. qui suivent (ce sont les C.M. de la deuxième colonne que vous devez introduire).

```
1024    A9,53          LDA 83          ;code de S
1026    8D,AA,BB       STA (48042)     ;1iere position PRINT
1029    A9,41          LDA 65          ;code de A
1031    8D,AB,BB       STA (48043)
1034    A9,4C          LDA 76          ;code de L
1036    8D,AC,BB       STA (48044)
1039    A9,55          LDA 85          ;code de U
1041    8D,AD,BB       STA (48045)
1044    A9,54          LDA 84          ;code de T
1046    8D,AE,BB       STA (48046)
1049    60             RTS
```

entrez **S** pour arrêter.

Puisque nous ne faisons pas appel aux routines de la ROM (MEM) qui, elles, effectuent toutes sortes de contrôles et de mises à jour de variables système, nous ne modifions pas la position **PRINT** du Basic. Nous devons toujours tenir compte de ce fait dans nos programmes.

Faites les commandes :

```
CLS
CALL 1Ø24
```

et **SALUT** s'affiche en haut à gauche.

Exercice 2-1 : *Ecrivez un programme qui affiche ORIC en haut à droite, donc aux adresses 48076 à 48079.*

- Indexé

9D,qq,pp	STA pq,X	POKE pq+X,A	
réalise	(pq+X)=A		

```
     99,qq,pp          STA pq,Y              POKE pq+Y,A
```

et pour la page Ø

```
     95,qq             STA q,X               POKE q+X,A
     96,qq             STX q,Y               POKE q+Y,X
     94,qq             STY q,X               POKE q+X,Y
```

Ces instructions n'ont pas d'influence sur les indicateurs.

- ***Indirect indexé***

```
     81,dd             STA d,X               HL=DEEK(d+X)
réalise plusieurs opérations                 POKE HL,A
        l=(od+X)
        h=(od+X+1)
        (hl)=A

     91,dd             STA d,Y               HL=DEEK(d)
réalise les opérations suivantes             POKE HL+Y,A
        l=(od)
        h=(od+1)
        (hl+Y)=A
```

Ces instructions n'ont pas d'influence sur les indicateurs. Comme les précédentes et comme toutes les suivantes, les instructions qui utilisent l'adressage indirect indexé réalisent trois opérations.

Manipulation des registres P et SP

```
     48                PHA                  POKE SP,A
réalise 2 opérations :
        (SP)=A
        SP=SP-1
```

pas d'effet sur les indicateurs.

```
     68                PLA                  LET A=PEEK(SP)
réalise 2 opérations :
        A=(SP)
        SP=SP+1
```

Cette instruction influence les indicateurs **N** et **Z.**

```
     Ø8                PHP                  POKE SP,P
effectue les opérations :
        (SP)=P
        SP=SP-1
```

pas d'influence sur les indicateurs.

```
     28              PLP                    LET P=PEEK(SP)
```

réalise les 2 opérations :

P=(SP)
SP=SP+1

Cette instruction influence évidemment **tous** les indicateurs puisque **P** prend une nouvelle valeur.

Il n'y a que **A** et **P** qui puissent être stockés provisoirement dans la pile. Nous disons provisoirement, car ceci est très **important.** La pile est constamment sollicitée par les routines de l'ORIC, ne fut-ce que pour y stocker les adresses de retour. Nous devons donc laisser cette pile dans l'état où on l'a trouvée. Dans nos programmes, si on emploie un **PHA** ou un **PHP**, nous devrons les compenser par un **PLA** ou un **PLP** avant le mnémonique **RTS.**

Du fait que l'on dispose des instructions **TXA** et **TYA** vues ci-dessus, on peut également pousser les registres X et Y sur la pile. Par exemple :

```
                TXA                 ;A=X
                PHA                 ;pousse X dans
                                    la pile
            instruction
              "
              "
              PLA                   ;extrait X de la
                                    pile
              TAX                   ;X=A
          suite avec la valeur de X sauvegardée
```

Donc, règle importante, autant de **PL** qu'on aura mis de **PH.** Notez que si on inverse ces instructions :

```
                PLA                 ;extrait un
                                    octet de la pile
                PHA                 ;rétablit la pile
```

dans l'intention d'examiner l'octet situé dans la pile, puisque nous le mettons dans **A**, nous disposons d'un moyen de comprendre le fonctionnement de la pile.

Si nous voulons examiner les deux octets qui sont en tête de pile (nécessaire pour une adresse), alors :

```
                PLA                 ;extrait le ler
                                    octet
                TAX                 ;X=A
                PLA                 ;extrait le 2e
                                    octet
                TAY                 ;Y=A
```

```
PHA                 ;introduit le
                    2e octet
TXA                 ;A=X
PHA                 ;introduit le ler
                    octet
```

et nous avons l'adresse **YX** (si c'est une adresse) avec une bonne compensation de la pile.

Ces instructions permettent également des échanges entre registres.

Exercice 2-2 : *Nous vous proposons d'échanger les registres A et X. Ceci peut se faire par un stockage intermédiaire en mémoire :*

```
STA pq              ;(pq)=A
TXA                 ;A=X
LDX pq              ;X=(pq)
```

mais essayez de le faire uniquement avec des mnémoniques de transferts T et de manipulation de pile PH et PL. N'écrivez que les mnémoniques. Cet exercice vous familiarisera avec ces instructions importantes.

Terminons ce paragraphe par les deux dernières instructions de transfert :

```
    BA              TSX
réalise             X=SP
```

influence les indicateurs N et Z

Ceci permet de connaître le pointeur de pile à tout moment.

```
    9A              TXS
réalise             SP=X
```

aucune influence sur les indicateurs.

C'est la seule manière de charger le registre **SP**.

L'ORIC, tout au début de son initialisation, exécute :

```
    A2,FF           LDX 255
    9A              TXS
```

et le pointeur de pile pointe sur 511. N'oubliez pas, voir chapitre I, que le microprocesseur ajoute 1 à gauche du registre SP pour que celui-ci utilise la page 1.

N'employez **jamais TXS.** Nous l'avons donné pour être complet.

Chargement des indicateurs du registre P

```
     18              CLC                    ;réalise C=Ø
     D8              CLD                    ;réalise D=Ø
met le mode binaire pour ADC et SBC (voir plus loin)
     58              CLI                    ;réalise I=Ø
pour autoriser les interruptions
     B8              CLV                    ;réalise V=Ø
     38              SEC                    ;réalise C=1
     F8              SED                    ;réalise D=1
met le mode décimal (DCB) pour ADC et SBC
     78              SEI                    ;réalise I=1
pour masquer les interruptions.
```

Evidemment, chacune de ces instructions influence l'indicateur correspondant.

Il n'y a pas d'instruction directe qui fasse V=1 ; nous verrons plus loin qu'on peut le réaliser avec une instruction **BIT.** On peut le réaliser également en chargeant **A** d'une manière appropriée (que nous voyons également plus loin), puis de faire

```
PHA
PLP
```

qui réalisent **P=A**

L'indicateur **B** est utilisé d'une façon interne pour distinguer une interruption provoquée par un break d'une autre. Ne pas confondre un break programmé par le mnémonique **BRK**, que nous verrons plus loin, avec le BREAK du Basic.

Les indicateurs **N** et **Z** sont mis uniquement suivant le résultat de l'instruction effectuée.

Notez que les indicateurs **N** et **V** sont mis quand on manipule des nombres avec signe, mais c'est uniquement vous qui savez si le nombre a un signe ou pas. L'ordinateur vous le signale simplement. Ainsi,

```
A9,EØ          LDA 224d
```

et l'ordinateur met N à 1, comme pour tout nombre supérieur à 127. Il y a EØh dans A, il représente 224 en

binaire pur ou -32 en nombre avec signe, suivant l'usage que vous avez décidé.

LES CODES MACHINE QUI REALISENT LES OPERATIONS ARITHMETIQUES

L'addition

Le microprocesseur effectue l'addition plus le report éventuel (indicateur C). Pour effectuer une addition sans inclure l'addition de C, on doit au préalable mettre C à Ø avec l'instruction **CLC.**

Les additions ne se réalisent qu'avec le registre A, et, la somme est stockée dans ce même registre A.

Addition d'une constante

69,nn	ADC n	LET A=A+n+C

réalise l'opération A=A+n+C

Cette instruction, comme toutes celles d'addition, influence les quatre indicateurs **N**, **V**, **Z** et **C.** Voyons comment, ne pas oublier qu'un octet ne contient que 8 bits et que, par conséquent, le résultat est toujours modulo 256.

```
1111 1111b +1 = 1 ØØØØ ØØØØb
   F    Fh +1 = 1    Ø    Øh
      255d +1 = Ø modulo 256
```

car l'octet contient Ø, le 9e bit se retrouve dans C.

Exemple 2 : Ecrivons un programme pour effectuer **X+Y**, en prévoyant **l'affichage du registre P** pour contrôler les indicateurs.

```
1024    18              CLC             :C=Ø
1025    A9,00           LDA x           :A=x
1027    69,00           ADC y           :A=x+y
1029    8D,1A,04        STA (1050)      :range A dans 1050
1032    08              PHP
1033    68              PLA             :A=P
1034    8D,1B,04        STA (1051)      :range P dans 1051
1037    60              RTS
```

Entrons ensuite le programme Basic d'exploitation. Au préalable, faites **NEW** pour effacer votre programme de chargement.

```
10 CLS
20 INPUT "ENTREZ X ET Y";X,Y
30 POKE 1026,X :POKE 1028,Y
40 CALL 1024
50 PRINT "X+Y=";PEEK(1050)
55 LET A=PEEK(1051)
60 PRINT :PRINT "LE REGISTRE P EST ";A
70 FOR T=7 TO 0 STEP -1
80 IF A>2^T-1 THEN B(T)=1 :A=A-2^T
90 NEXT T
100 PRINT "N=";B(7),"U=";B(6),"Z=";B(
    1),"C=";B(0)
```

Les lignes 70 à 90 convertissent le nombre décimal A en binaire. Ce nombre binaire tient dans la matrice B : le bit 7 dans B(7), ... le bit Ø dans B(Ø).

a- entrez 255,1 somme = Ø mod
N=Ø car la somme est plus petite que 128
V=Ø voir explication plus loin R6=1
Z=1 car la somme est nulle
C=1 car il y a report dans l'addition des bits 7

b- entrez 255,128 somme = 127 mod
N=Ø
V=1 voir plus loin R6=Ø
Z=Ø somme non nulle
C=1 il y a report des bits 7

c- entrez 65,65 somme = 13Ø
N=1 plus grand que 127
V=1 plus loin R6=1
Z=Ø somme non nulle
C=Ø pas de report des bits 7

d- entrez 64,36 somme = 1ØØ
N=Ø
V=Ø plus loin R6=Ø
Z=Ø non nulle
C=Ø pas de report des bits 7

Nous possédons les **quatre cas** pour expliquer **l'indicateur V.** Nous vous conseillons d'effectuer les quatre additions ci-dessus en binaire, pour savoir s'il y a un

report dans l'addition des bits 6. Nous avons d'ailleurs indiqué R6=1 s'il y a report. Comme exemple, prenons la 1ère addition :

```
 11111111        reports
  11111111       255
 C0000000 1       +1
 ---------      ----
 100000000
```

Il y a un report pour l'addition de chaque colonne de bits. Le report des bits 6 est 1, donc écrivons R6=1. Le report des bits 7 est 1, ici le report va dans C, donc C=1.

Comme nous avons appris le **OU** exclusif dans le chapitre I, nous constatons que :

V= R6 ⊻ C

quand R6 et C sont différents, V est mis à 1 pour indiquer un dépassement dans l'addition des nombres avec signe.

En effet, si on considère les nombres ci-dessus comme étant des nombres avec signe, nous avons pour

a-	-1 + 1 = 0	vrai	V=0
b-	-1 + -128 = 127	faux	V=1
c-	65 + 65 = -126	faux	V=1
d-	64 + 36 = 100	vrai	V=0

Vous trouverez les nombres avec signe en annexe, tout au moins ceux d'un octet.

Entrez les nombres que vous désirez pour approfondir la connaissance de cette importante instruction.

Addition du contenu d'une cellule mémoire

- Absolu

6D,qq,pp	ADC pq	LET A=A+PEEK(pq)+C
réalise	A=A+(pq)+C	

et pour la page 0

65,qq	ADC q	A=A+PEEK(q)+C
réalise	A=A+(oq)+C	

Exemple 3 : Ecrivons un programme pour effectuer **x+y** avec des **nombres qui tiennent en 2 octets**, donc de 0 à

65535. Le résultat sera modulo 65536. Nous prévoyons également l'affichage du registre P pour ceux intéressés par l'addition de nombres avec signe qui sont, dans ce cas, des nombres allant de -32768 à 32767 (voir chapitre I).

```
1024    18              CLC             ;C=0
1025    AD,1A,04        LDA (1050)      ;A=octet bas de x
1028    6D,1C,04        ADC (1052)      ;ajoute l'octet bas de y
1031    8D,1E,04        STA (1054)      ;range l'octet bas
1034    AD,1B,04        LDA (1051)      ;A=octet haut de x
1037    6D,1D,04        ADC (1053)      ;ajoute l'octet haut de y
1040    8D,1F,04        STA (1055)      ;range l'octet haut
1043    08              PHP
1044    68              PLA
1045    8D,19,04        STA (1049)      ;range P a 1049
1048    60              RTS
```

Reprenez le Basic d'exploitation de *l'exemple 2* et changez les trois lignes 30, 50 et 55 :

```
30 DOKE 1050,X :DOKE 1052,Y
50 PRINT "X+Y=";DEEK(1054)
55 LET A=PEEK(1049)
```

```
entrez  65535,1         somme 0
entrez 32400, 28197     somme 60597
etc.
```

Cette méthode peut s'étendre à l'addition de nombres contenus dans trois octets et plus.

- ***Indexé***

```
    7D,qq,pp        ADC pq,X        LET A=A+PEEK(pq+X)+C
réalise             A=A+(pq+X)+C

    79,qq,pp        ADC pq,Y
réalise             A=A+(pq+Y)+C
```

et pour la page Ø

```
    75,qq           ADC q,X
réalise             A=A+(oq+X)+C
```

Toutes ces instructions indexées : chargement, rangement, addition, etc. s'utilisent, entre autres, pour traiter des tableaux, des matrices situées en mémoire. Nous en verrons des exemples par la suite.

- Indirect indexé

```
     61,dd              ADC d,X          LET HL=DEEK(d+X)
réalise les opérations                   LET A=A+PEEK(HL)+C
        l=(od+X)
        h=(od+X+1)
        A=A+(hl)+C

     71,dd              ADC d,Y          HL=DEEK(d)
réalise les opérations                   A=A+PEEK(HL+Y)+C
        l=(od)
        h=(od+1)
        A=A+(hl+Y)+C
```

Les opérations indirectes indexées s'emploient fréquemment dans la ROM (MEM) puisqu'elles font appel à une adresse tenue en page Ø. C'est dans cette page que l'ordinateur range les adresses et les variables souvent utilisées car c'est dans cette page que la vitesse d'exécution d'une instruction est la plus rapide.

Remarque : ADC n signifie en Basic LET A=A+n+C mais aussi :

```
IF (A+n+C)>255 THEN LET C=1
IF (A+n+C)=256 THEN LET Z=1
IF (A+n+C)>127 THEN LET N=1
```

La mise à **1** de **V** est plus délicate à transcrire en Basic. La transcription d'une instruction C.M. en Basic peut nécessiter plusieurs lignes Basic et réciproquement. Cette transcription est toujours circonstanciée.

Les incrémentations

L'incrémentation est **l'addition de 1.** Elle peut s'effectuer dans une cellule mémoire ou sur les registres X ou Y.

L'incrémentation influence les indicateurs N et Z.

```
     E8                 INX              LET X=X+1
réalise                 X=X+1

     C8                 INY
réalise                 Y=Y+1

     EE,qq,pp           INC pq           POKE PQ,PEEK(PQ)+1
réalise                 (pq)=(pq)+1
```

et en page Ø

```
    E6,qq           INC q
réalise             (oq)=(oq)+1
```

et avec l'adressage indexé

```
    FE,qq,pp        INC pq,X
réalise             (pq+X)=(pq+X)+1

    F6,qq           INC q,X
réalise             (oq+X)=(oq+X)+1
```

Pour incrémenter le registre **A**, on peut faire :

```
CLC
ADC 1
```

Si vous savez qu'à cet endroit du programme, C=1, faites alors

```
ADC Ø
```

D'autre part, si un registre d'index est disponible, alors :

```
TAX
INX
TXA
```

Les instructions d'incrémentation sont constamment employées. Elles servent tout d'abord comme compteur et, à ce titre, contrôlent des boucles quand elles sont associées à des instructions de branchement que nous verrons dans le chapitre suivant.

En ce qui concerne les registres **X** et **Y**, plus particulièrement, les incrémentations contrôlent toutes les opérations où l'adressage indexé et l'adressage indirect pré- et post-indexé sont concernées.

La soustraction

Tout comme pour l'addition, le microprocesseur n'effectue la soustraction d'un nombre que du registre **A**, et range la différence dans ce même registre A. La soustraction est toujours effectuée avec la retenue contenue dans l'indicateur **C**. De ce fait, la soustraction d'un nombre sans retenue doit être précédée d'une instruction **SEC** qui met C à 1, parce que le microprocesseur retranche le complément de C.

Soustraction d'une constante

E9,nn	SBC n	RET=1-C
effectue l'opération $A=A-n-\overline{C}$		LET A=A-n-RET

Toutes les instructions de soustraction influencent les quatre indicateurs principaux **N**, **V**, **Z** et **C**.

Exemple 4 : Programme pour effectuer x-y en faisant apparaître également les quatre indicateurs principaux pour étudier leurs fonctionnements.

```
1024    38              SEC             ;C=1
1025    A9,00           LDA x           ;A=x
1027    E9,00           SBC y           ;A=x-y
1029    8D,1A,04        STA (1050)      ;range A
1032    08              PHP
1033    68              PLA
1034    8D,1B,04        STA (1051)      ;range P
1037    60              RTS
```

Entrons ensuite le programme Basic d'exploitation, en employant les mêmes numéros de ligne que le programme de chargement pour remplacer celui-ci.

```
10 CLS
20 INPUT "ENTREZ X ET Y";X,Y
30 POKE 1026,X :POKE 1028,Y
40 CALL 1024
50 PRINT "X-Y=";PEEK(1050)
55 LET A=PEEK(1051)
60 PRINT :PRINT "LE REGISTRE P EST ";A
70 IF (A AND 128)=128 THEN N=1
80 IF (A AND 64)=64 THEN V=1
90 IF (A AND 2)=2 THEN Z=1
100 IF (A AND 1)=1 THEN C=1
110 PRINT "N=";N,"V=";V,"Z=";Z,"C=";C
```

Les lignes 7Ø à 11Ø vous montrent un autre moyen d'extraire les quatre indicateurs principaux de P que celui employé dans l'exemple 2. Cela introduit le **AND** du Basic qui, dans ce cas, est semblable au mnémonique AND que nous expliquons plus loin.

Essayez différentes soustractions.

Vous voyez que 1ØØ - 55 = 45 et les indicateurs C=1 et V=Ø.

La soustraction 2 - 3 = 255 mod 256, amène C=Ø et V=Ø ; car 255, considéré comme nombre avec signe, représente -1.

En réalité, le microprocesseur effectue l'addition des compléments restreints (complément à 255 pour un octet et à 1 pour un bit), donc **SBC n** réalise l'opération

A=A+n+C

Dans le premier cas

```
   1ØØ
+  2ØØ   soit 255-55
+    1   soit C
  ----
   3Ø1   ou 45 mod 256 qui met C à 1
```

Dans le 2e cas

```
     2
+  252   soit 255-3
+    1   soit C
  ----
   255   ce qui met C=Ø
```

Nous nous sommes étendus sur le fonctionnement des instructions d'addition et de soustraction pour ceux qui veulent approfondir la connaissance du microprocesseur. En première lecture, passez les explications données et effectuez simplement des additions et des soustractions pour assimiler la notion de modulo 256. Les conducteurs ont déjà une notion de ce modulo. Sur le tableau de bord d'une voiture, on trouve un compteur kilométrique journalier, ce compteur décimal est modulo 1ØØØ.

Soustraction du contenu d'une cellule mémoire

- ***Absolu***

ED,qq,pp	SBC pq	RET=1-C
effectue	A=A-(pq)-$\overline{C}$	LET A=A-PEEK(pq)-RET

et en page Ø

E5,qq	SBC q	
soit	A=A-(oq)-$\overline{C}$	

Exercice 2-3 : *Ecrivez un programme pour réaliser la soustraction de nombres tenus dans deux octets mémoire, allant de Ø à 65535.*

Le résultat est modulo 65536.
Changez 3 C.M. de l'exemple 3.

- Indexé

FD,qq,pp	SBC pq,X	RET=1-C
réalise	A=A-(pq+X)-$\overline{C}$	LET A=A-PEEK(pq+X)-RET
F9,qq,pp	SBC pq,Y	
réalise	A=A-(pq+Y)-$\overline{C}$	

et en page Ø

F5,qq	SBC q,X	
réalise	A=A-(oq+X)-$\overline{C}$	

- Indirect indexé

E1,dd	SBC d,X	HL=DEEK(d+X)
réalise les opérations		RET=1-C
l=(od+X)		A=A-PEEK(HL)-RET
h=(od+X+1)		
A=A-(hl)-$\overline{C}$		
F1,dd	SBC d,Y	HL=DEEK(d)
réalise les opérations		RET=1-C
l=(od)		A=A-PEEK(HL+Y)-RET
h=(od+1)		
A=A-(hl+Y)-$\overline{C}$		

Exemple 5 : Ecrivons un programme en C.M. pour connaître la longueur d'un programme Basic.

L'adresse de départ du programme Basic tient, en page Ø, dans les octets 9A et 9B et celle de départ des variables Basic tient dans 9C et 9D. Nous devons réaliser DEEK(#9C)-DEEK(#9A)

```
1024    38              SEC             ;C=1
1025    A5,9C           LDA (9C)        ;octet bas de var
1027    E5.9A           SBC (9A)        ;- octet bas de BASIC
1029    8D,1E,04        STA (1054)
1032    A5,9D           LDA (9D)        ;octet haut de VAR
1034    E5,9B           SBC (9B)        ;- BASIC haut
1036    8D,1F,04        STA (1055)
1039    60              RTS
```

Entrons le Basic d'exploitation suivant :

```
10 CLS
20 PRINT "LE PROGRAMME BASIC COMPREND";
30 CALL 1024
40 PRINT DEEK(1054);"OCTETS"
50 END
```

Les décrémentations

La décrémentation est la **soustraction de 1.** Elle s'effectue soit sur le contenu d'une cellule mémoire soit sur les registres X ou Y. Les décrémentations influencent les indicateurs N et Z.

CA	DEX	LET X=X-1
réalise l'opération	X=X-1	
88	DEY	
réalise	Y=Y-1	
CE,qq,pp	DEC pq	POKE PQ,PEEK(PQ)-1
réalise	(pq)=(pq)-1	
C6,qq	DEC q	
réalise	(oq)=(oq)-1	

et avec l'indexation

DE,qq,pp	DEC pq,X
réalise	(pq+X)=(pq+X)-1
D6,qq	DEC q,X
réalise	(oq+X)=(oq+X)-1

Les décrémentations ont les mêmes usages que les incrémentations ; comme compteur pour contrôler les boucles **FOR** et **REPEAT** et comme index pour les instructions indexées. Les résultats sont modulo 256.

Pour décrémenter le registre **A**, on fait :

```
CLC
ADC 155
```

ou en employant la soustraction :

```
SEC
SBC 1
```

ou éliminer une instruction si on connaît l'état de l'indicateur C.

Si un des registres d'index n'est pas employé, on fait :

```
TAY
DEY
TYA
```

avec de l'influence uniquement sur N et Z.

REPONSES AUX EXERCICES DU CHAPITRE II

Exercice 2-1

```
1024    A9,4F          LDA 79          ;code de O
1026    8D,CC,BB       STA (48076)
1029    A9,52          LDA 82          ;code de R
1031    8D,CD,BB       STA (48077)
1034    A9,49          LDA 73          ;code de I
1036    8D,CE,BB       STA (48078)
1039    A9,43          LDA 67          ;code de C
1041    8D,CF,BB       STA (48079)
1044    60             RTS
```

Faites la commande :

```
CALL 1Ø24
```

Exercice 2-2

```
TAY             ;sauve A dans Y
TXA             ;met X dans A
PHA             ;pousse X dans la pile
TYA             ;récupère A
TAX             ;met A dans X
PLA             ;met X dans A
```

Ceci est un moyen, vous en avez peut-être trouvé un autre. Il y a toujours plusieurs voies pour réaliser les opérations.

Exercice 2-3

```
1024    38             SEC             ;C=1
1025    AD,1A,04       LDA (1050)      ;A=oct. bas de x
1028    ED,1C,04       SBC (1052)      ;soust. oct. bas de y
1031    8D,1E,04       STA (1054)      ;rangement
1034    AD,1B,04       LDA (1051)      ;A=octet haut de x
1037    ED,1D,04       SBC (1053)      ;moins oct. haut de y
1040    8D,1F,04       STA (1055)      ;rangement
1043    08             PHP
1044    68             PLA
1045    8D,19,04       STA (1049)      ;range P
1048    60             RTS
```

Les trois lignes suivantes du Basic d'exploitation de *l'exemple 4*, changent :

```
30 DOKE 1050,X :DOKE 1052,Y
50 PRINT "X-Y=";DEEK(1054)
55 LET A=PEEK(1049)
```

Essayez différentes soustractions modulo 65536.

CHAPITRE 3
BOUCLES, COMPARAISONS, TESTS

Le chapitre précédent vous a familiarisé avec les *opérations de chargement* et de *lecture des registres* du microprocesseur et des *emplacements mémoire* représentés par leurs adresses. Ces instructions sont très importantes, elles s'emploient constamment. Imagineriez-vous un programme Basic sans variables ? Les *opérations d'addition* et de *soustraction* ont ensuite été étudiées. En plus de leurs caractères arithmétiques, ces instructions influencent les indicateurs du registre P. Ceci est également très important, car ces indicateurs amènent des décisions conditionnelles qui, elles aussi, s'emploient constamment et sans lesquelles l'ordinateur n'existerait pas. Le programme enregistré (en ROM ou en RAM pour le vôtre) et la rupture de séquence sont les deux concepts de base de tout ordinateur.

Ce chapitre aborde les **tests** et les **comparaisons** qui interrompent le déroulement séquentiel d'un programme et provoquent ainsi des boucles et des sauts. Une bonne compréhension des matières de ce chapitre est indispensable.

L'ECRAN BASSE RESOLUTION

Cet écran est employé par l'ORIC dans les modes **TEXT**, **LORES Ø** et **LORES 1**. Il se divise en 28 lignes de 4Ø colonnes. Il occupe par conséquent, 112Ø octets de la mé-

moire vive que l'on remplit de codes de caractères ou d'attributs.

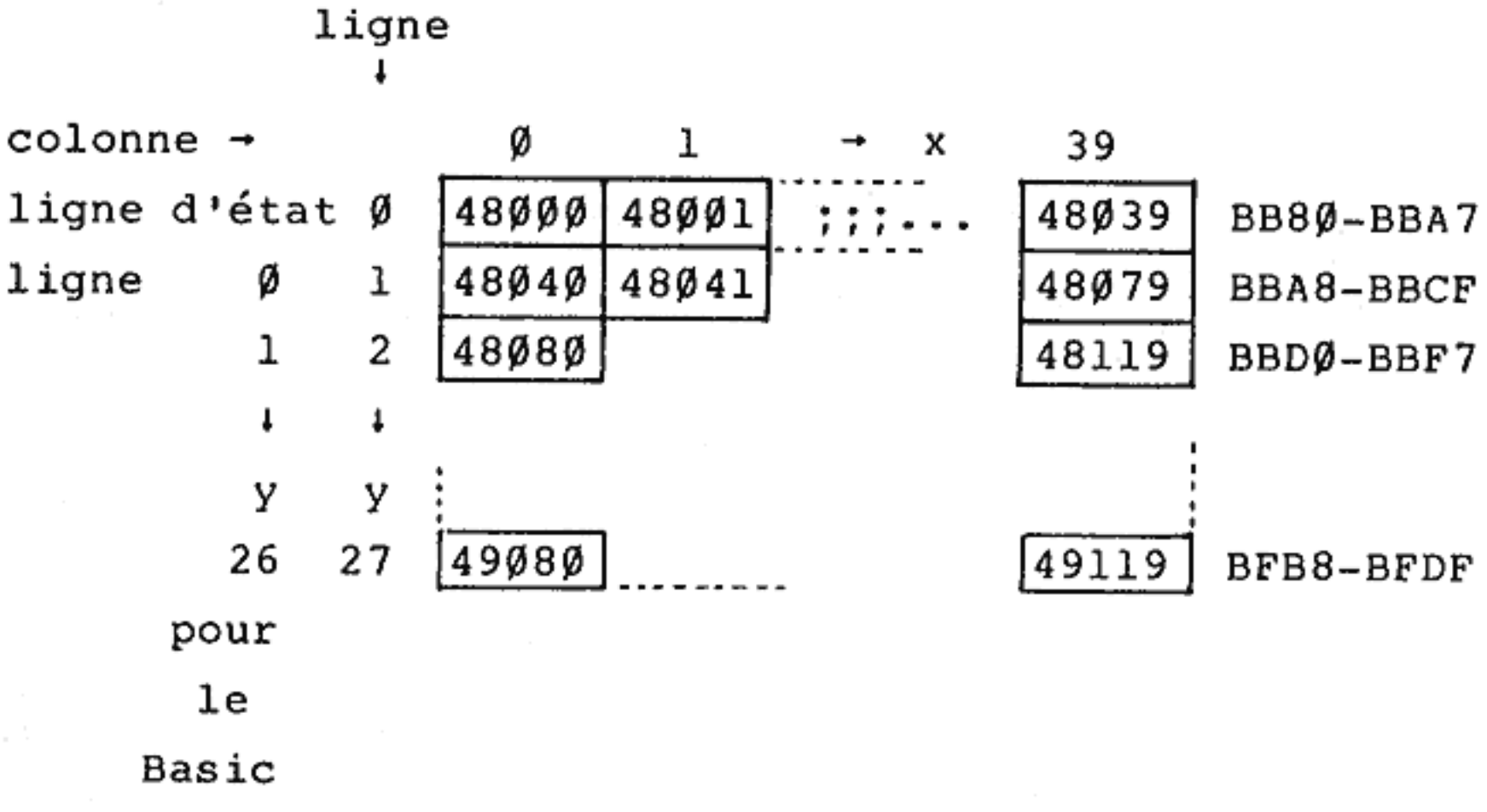


Figure 1- Implantation mémoire de l'écran en LORES

L'écran présenté ici (**figure 1**) correspond à un ORIC de 48K. Pour un 16K, retranchez 32768d - 8ØØØh des adresses mentionnées : l'écran ira de 15232 à 16351 (3B8Ø à 3FDF). Il en sera de même pour toutes les adresses hautes rencontrées par la suite, sauf pour les adresses de la ROM (49152 à 65535 - CØØØ à FFFF) qui est située au même endroit quel que soit l'ORIC employé.

Pour charger les programmes rencontrés dans la suite de cet ouvrage, nous employons le *programme 2* du chapitre I. Pour les possesseurs d'un 16K, changez la ligne 5Ø1Ø qui devient 5Ø1Ø HIMEM 5119 et les programmes chargés à 37888 le seront à 512Ø.

La première ligne, ou ligne Ø, est réservée par l'ORIC comme ligne d'état pour y afficher des messages lorsqu'on emploie l'enregistreur et, en fin de ligne, pour afficher CAPS quand on se trouve en mode majuscules qui est l'état normal. Les commandes Basic **PRINT @** (l'ORIC 1 ne possède pas cette facilité) et **PLOT**, ainsi que la fonction **SCRN** admettent une abscisse de Ø à 39 et une ordonnée de Ø à 26. L'ordonnée correspondante dans la figure 1 va de 1 à 27 : Ayez ceci à l'esprit, car avec les C.M. nous allons n'importe où.

Ajoutons que les deux premières colonnes sont réservées pour des attributs. Le Basic en tient compte,

notamment quand vous entrez des lignes de programme ou quand vous employez PRINT.

LES SAUTS INCONDITIONNELS

4C,qq,pp	JMP pq	GOTO pq
réalise	PC=pq	
6C,qq,pp	JMP pq,IND	HL=DEEK(pq)
réalise trois instructions		GOTO HL
l=(pq)		
h=(pq+1)		
PC=hl		

Cette dernière instruction est la seule instruction indirecte. Les autres instructions indirectes sont indexées.

Les deux JMP n'ont pas d'influence sur les indicateurs.

LES BRANCHEMENTS CONDITIONNELS RELATIFS

Les branchements conditionnels testent l'un des quatre indicateurs principaux **N**, **V**, **Z** et **C**. Suivant que l'indicateur testé est à 1 ou à Ø, le branchement s'effectue ou pas. Les branchements possibles vont de -126 à 129. En effet le deuxième octet des instructions de branchement est toujours considéré par l'ordinateur comme un nombre avec signe pour avoir la faculté d'exécuter des branchements vers l'avant ou vers l'arrière du programme.

Les instructions de branchement n'ont pas d'influence sur les indicateurs.

1Ø,ee	BPL e		
réalise l'opération	PC=PC+2+e		si N=Ø
	ou PC=PC+2		si N=1

e est un nombre avec signe.

Cette instruction peut se représenter en Basic par :

```
2Ø IF N=Ø THEN 1ØØ
21 suite du programme si  N=1
1ØØ suite du programme si N=Ø
```

```
     3Ø,ee            BMI e
réalise l'opération   PC=PC+2+e        si N=1
                   ou PC=PC+2          si N=Ø
```

e est un nombre avec signe.

Correspond au Basic suivant :

```
2Ø IF N=1 THEN 1ØØ
21 suite du programme si  N=Ø
1ØØ suite du programme si N=1
```

Ces deux instructions sont fort employées car **l'indicateur N** est influencé par beaucoup d'instructions. Il est presque toujours le reflet du bit 7 d'un registre ou d'une cellule mémoire. Par exemple :

```
     AD,qq,pp         LDA pq
```

influence l'indicateur N. Si A est plus grand que 127, N est mis à 1 et, si A est plus petit que 128, N est mis à Ø.

Supposons que l'on charge A avec un code de caractère. S'il s'agit d'un caractère affichable, N=Ø, si non N=1. A l'aide de 1Ø,ee ou de 3Ø,ee, on décide de la suite du programme dans une voie ou dans une autre.

```
     5Ø,ee            BVC e
réalise               PC=PC+2+e        si V=Ø
     ou               PC=PC+2          si V=1

     7Ø,ee            BVS e
réalise               PC=PC+2+e        si V=1
     ou               PC=PC+2          si V=Ø
```

L'indicateur V n'est influencé que par les instructions **ADC** et **SBC.** Comme nous l'avons vu dans le chapitre précédent, l'addition ou la soustraction peut amener un dépassement de capacité de calcul sur des nombres avec signe et l'indicateur V intervient. Son emploi est peu fréquent en ce sens.

A la fin du chapitre, nous verrons l'instruction **BIT** qui emploie **V** d'une autre manière. A ce moment le test de V montre toute sa valeur.

```
     DØ,ee            BNE e
réalise               PC=PC+2+e        si Z=Ø
     ou               PC=PC+2          si Z=1

     FØ,ee            BEQ e
réalise               PC=PC+2+e        si Z=1
     ou               PC=PC+2          si Z=Ø
```

Puisque **l'indicateur Z** est influencé par beaucoup d'instructions (comme l'indicateur N), ces deux tests sont très fréquemment employés.

Exemple 1 : Soit à **incrémenter une adresse.**
Cette opération se retrouve partout dans la ROM de l'ORIC. Incrémentons l'adresse contenue dans les octets situés à E9 et EA.

```
1028    E6,E9          INC (E9)
1030    D0,02          BNE 2+2          ;saut si Z=0
1032    E6,EA          INC (EA)
1034                   suite du programme
```

E9 contient l'octet bas de l'adresse et **EA** l'octet haut.

Nous commençons par incrémenter l'octet bas et tant que celui-ci n'atteint pas Ø, Z se met à Ø et il y a branchement. Dès que (E9)=Ø, Z se met à 1 et on incrémente alors l'octet haut.

Par exemple si l'adresse contenue dans le couple E9-EA était 5ØØh, elle devient 5Ø1h. Si elle était 5FFh, elle devient 6ØØh.

Exemple 2 : Réalisons un **branchement conditionnel long.**
Du fait que le branchement relatif se fait dans la fourchette -126 à 129, il peut arriver qu'on ait besoin d'un branchement plus grand suite à un test.

```
E8              INX
DØ,Ø3           BNE 2+3          ;si X≠Ø
4C,qq,pp        JMP pq           ;si X=Ø, branchement
                suite             à pq
```

Quand à la suite de l'incrémentation de X, ce registre vient à Ø, il y aura branchement à pq, donc aussi loin que l'on veut en avant ou en arrière.

```
    9Ø,ee           BCC e
réalise             PC=PC+2+e        si C=Ø
                    PC=PC+2          si C=1

    BØ,ee           BCS e
réalise             PC=PC+2+e        si C=1
                    PC=PC+2          si C=Ø
```

L'indicateur C est influencé par moins d'instructions que N et Z mais par plus d'instructions que V et, notamment par les instructions de comparaison, de glissements et de rotations qui n'influencent pas V. Les tests sur C seront donc plus fréquents que ceux sur V.

Avant d'aborder quelques exemples et exercices, faisons remarquer que le branchement relatif peut s'interpréter de deux manières, selon le goût de chacun. Reprenons *l'exemple 1* et examinons la lère colonne, celle d'adresses. A 1Ø3Ø se trouve l'instruction de branchement qui branche à 1Ø3Ø+2+2 soit à 1Ø34. Nous comptons le branchement à partir de 1Ø3Ø, c'est pour cette raison qu'on a ajouté +2. (comme le microprocesseur pour son registre PC). Certains préfèrent compter à partir de l'instruction qui suit celle de branchement ; si c'est votre cas, supprimez +2 dans toutes nos instructions de branchement.

```
1030    DØ,Ø2          ┌─ BNE +2          ;2 est le 2ieme octet
1032    E6,EA          │                  ;   de l'inst.
1034                   └─►
```

Dans ce cas on compte à partir de l'instruction suivante 1Ø32 + 2 soit 1Ø34. C'est selon le goût : le résultat est le même.

Si vous employez un programme assembleur, vous mettez simplement une étiquette et c'est l'assembleur qui calcule le déplacement.

A partir des exemples suivants, nous employons le *programme 2* du chapitre I.

Exemple 3 : Calcul de la **position PRINT** ou d'affichage. Pour connaître la position PRINT selon la *figure 1*, nous devrons entrer 2 paramètres : le n° de colonne et le n° de ligne, avant d'appeler la routine suivante.

Avec le *programme de chargement 2*, à la question ENTREZ L'ADRESSE, répondez 37892 et entrez les codes à partir de cette adresse.

```
37888   ØØ,ØØ                DEF pos.PRINT   ;variable
37890   ØØ                   DEF colonne     ;de 2 a 39
37891   ØØ                   DEF ligne       ;de 1 a 27
37892   A9,BB      ENT.      LDA BB          ;48000 octet haut
37894   8D,Ø1,94             STA (9401h)     ;(37889d)
37897   18                   CLC             ;C=Ø
37898   A9,8Ø                LDA 8Øh         ;48000 octet bas
37900   6D,Ø2,94             ADC (9402h)     ;(37890) colonne
37903   AE,Ø3,94             LDX (9403h)     ;(37891) ligne
37906   FØ,ØB          ┌──── BEQ 2+11        ;si X=Ø
37908   69,28          │┌──► ADC 4Ød         ;pour une ligne
37910   9Ø,Ø4          ││┌─┐ BCC 2+4         ;si C=Ø
37912   EE,Ø1,94       ││└─┼► INC (9401)     ;octet haut mis a jour
37915   18             ││  └─► CLC          ;C=Ø
37916   CA             ││     ►DEX          ;X=X-1
37917   DØ,F5          ├└──── BNE 2-11      ;si X diff. de Ø
37919   8D,ØØ,94       └─────► STA (9400)   ;stocke l'octet bas
37922   6Ø                     RTS           ;retour au Basic
```

Nous employons pour la première fois la colonne étiquette. **ENT.** signifie que l'entrée de la routine se situe à 37892. Pour les 4 octets précédents, nous employons **DEF** (pour définition). DEF n'est pas un mnémonique, il signifie que nous réservons les octets correspondants pour nos variables. Les deux premiers octets contiendront la position **PRINT** calculée par le programme, et les deux suivants devront être chargés par un programme Basic approprié. Nous avons mis des commentaires à toutes les lignes, pour que vous compreniez bien cette routine. Nous employons plusieurs branchements dont un qui provoque une boucle, à 37917.
48ØØØd = BB8Øh = début d'écran.

Voici le programme Basic d'exploitation :

```
5 CLS
10 INPUT "ENTREZ LE NUMERO DE COLONNE";C
20 INPUT"ENTREZ LE NUMERO DE LIGNE";L
30 POKE 37890,C :POKE 37891,L
40 CALL 37892 :PRINT
50 PRINT "LA POSITION PRINT EST";DEEK
   (37888)
5000 STOP
```

- Si vous entrez **Ø** pour **C** et **Ø** pour **L**, la réponse est 48ØØØ
- Si vous entrez **39** pour **C** et **27** pour **L**, la réponse est 49119
- Si vous entrez **Ø** pour **C** et **8Ø** pour **L**, la réponse est 512ØØ.

ce qui est hors limites. Nous n'avons fait aucun test de validité. Pour exploiter ce programme avec le suivant, entrez pour C un nombre de 2 à 39 et pour L de 1 à 27.

Afficher au-delà de 49119 ne porte aucun préjudice, car nous avons d'abord une zone réserve, de 4912Ø à 49151 et ensuite, la ROM ou MEM, de 49152 à 65535, dans laquelle aucune écriture n'est possible.

Exemple 4 : **Affichage de texte.** C'est comme *l'exemple 1* du chapitre précédent, mais en employant une boucle pour l'affichage et la routine précédente pour le calcul de la position **PRINT.**

Laissez la routine précédente en mémoire. Quand vous faites RUN 5Ø1Ø, répondez 37923 à la question d'entrée et, entrez tous les octets jusqu'à, et y compris, 37963.

```
37923   AD,00,94  ENT2    LDA (9400)      ;transfert de la pos.
37926   AE,01,94          LDX (9401)      ;   PRINT en page 0
37929   85,0C             STA (0C)        ;   dans les octets
37931   86,0D             STX (0D)        ;   0C-0D
37933   A0,00             LDY 0           ;initialise Y
37935   B9,3B,94          LDA (37947+Y)   ;charge le lier carac.
37938   F0,06             BEQ 2+6         ;si A=0
37940   91,0C             STA (0C0,INDY   ;l'affiche
37942   C8                INY             ;carac. suivant
37943   4C,2F,94          JMP 37935       ;saut incondi.
37946   60                RTS             ;retour au Basic
37947   4C,45,43          DEF LEC         ;message
37950   54,45,55,52,2C    DEF TEUR,       ;  id.
37955   20,4D,4F,4E       DEF  MON        ;  id.
37959   20,41,4D,49       DEF  AMI        ;  id.
37963   00                DEF fin         ;indicateur de fin
                                          ;  de message
```

Le message à afficher commence à 37947 et **doit** se terminer par Ø, car c'est l'indicateur de fin de message que nous avons choisi. Dès qu'il est dans A, la ligne 37938 branche à 37946. Comme nous savons que le message commence à 37947, nous employons, pour charger A, une instruction indexée par Y.

Par contre, pour placer le caractère dans le fichier d'affichage, nous savons que l'adresse de départ est tenue dans les octets 37888-37889, nous devons donc accéder au contenu du contenu de. Pour y accéder nous employons une instruction indirecte indexée par Y. Les quatre premières instructions de la routine **ENT2** servent à transférer l'adresse de la position **PRINT** ou d'affichage dans les octets ØC-ØD, en page Ø. Les octets ØC à ØF sont employés pour cet usage par la ROM. N'employez pas les octets 12-13h car c'est une variable système.

Etudiez cette routine pour bien comprendre son fonctionnement. Elle correspond au Basic PLOT.

Voici le Basic d'exploitation :

```
5 CLS
10 INPUT "ENTREZ LE NUMERO DE COLONNE";C
20 INPUT"ENTREZ LE NUMERO DE LIGNE";L
30 POKE 37890,C :POKE 37891,L
40 CLS
50 CALL 37892 :CALL 37923
5000 STOP
```

Entrez un numéro de colonne de 2 à 24 (puisque notre message comporte 16 caractères) et un numéro de ligne

de 1 à 27. Entrez, par exemple, 11 pour C et 14 pour L.

Exercice 3-1 : *Ecrivez un programme pour afficher BONJOUR à partir de la position PRINT 48575. Donc la position de départ est connue. N'employez que des instructions indexées. Les caractères à afficher sont situés à partir de 379Ø3 (si vous faites le même programme que nous).*

Entrez vos codes à partir de 37888.

Le Basic d'exploitation est :

```
5 CLS
10 CALL 37888
5000 STOP
```

Exemple 5 : Réalisons la **commande CLS**

```
37888   A9,A9      CLS     LDA A9          ;adr. B de l'ecran
37890   A2,BB              LDX BB          ;adr. H de l'ecran
37892   8D,11,94           STA (37905)
37895   8E,12,94           STX (37906)
37898   A2,1B              LDX 27d         ;27 lignes
37900   A9,20        ┌►    LDA 32d         ;code 'espace'
37902   A0,26        │     LDY 38d         ;38 colonnes
37904   99,A9,BB     │  ┌►STA (48041+Y)   ;efface
37907   88           │  │  DEY            ;   une
37908   D0,FA        │  └─BNE 2-6         ;   ligne
37910   18           │     CLC
37911   AD,11,94     │     LDA (37905)    ;octet B
37914   69,28        │     ADC 40d
37916   90,03        │  ┌─ BCC 2+3
37918   EE,12,94     │  │  INC (37906)    ;octet H
37921   8D,11,94     │  └►STA (37905)
37924   CA           │     DEX            ;ligne suivante a
37925   D0,E5        └──── BNE 2-27       ;   effacer
37927   60                 RTS
```

Cette routine vous montre deux **boucles imbriquées**, l'une, avec Y comme compteur, pour effacer une ligne et l'autre, avec X comme compteur, pour effacer les 27 lignes successives. Cet exemple vous montre également un programme qui se modifie lui-même. L'adresse de départ de l'écran, d'où l'on veut effacer, est tenu dans les octets 379Ø5-379Ø6. A chaque bouclage de X, cette adresse est augmentée de 4Ø par les lignes 3791Ø à 37921.

Les 4 premières lignes rétablissent l'adresse originale pour utilisation ultérieure.

Cette routine n'est pas relogeable ailleurs ; à moins que vous ne changiez cinq adresses absolues aux lignes 37892, 37895, 37911, 37918 et 37921.

Pour voir cette routine en action, faites :

```
LIST          pour avoir l'écran rempli, puis
CALL 37888
```

Vous constaterez que le curseur reste en bas, ce qui est évident puisqu'on ne manipule pas les variables système concernées.

Exercice 3-2 : *Pour contrôler si vous avez compris cette routine, faites plusieurs petits programmes Basic, sans changer la routine, pour réaliser :*

1- effacer la moitié supérieure de l'écran
2- effacer la moitié inférieure de l'écran
3- effacer la moitié gauche de l'écran
4- effacer la moitié droite de l'écran.

LES COMPARAISONS

On peut comparer un nombre ou un octet mémoire soit à A, soit à X ou soit à Y.

Les comparaisons influencent les trois indicateurs principaux N, Z et C.

Comparaison d'une constante à A

```
     C9,nn          CMP n
réalise             A-n
```

pour influencer les trois indicateurs N,Z et C. Le résultat de la soustraction n'est pas stocké. Le registre A ne change pas.

La traduction Basic donne ceci :

```
IF A=n THEN Z=1 ELSE Z=Ø
IF (A-n)>127 THEN N=1 ELSE N=Ø
```

modulo 256, bien sûr ; 1ØØ-11Ø fait N=1 car le résultat serait 246 (-1Ø)

```
IF A<n THEN C=Ø ELSE C=1
```

Toutes les comparaisons sont d'ailleurs des :

```
IF ... THEN ... ELSE ...
```

Comparaison d'un octet mémoire à A

- Absolu

```
     CD,qq,pp          CMP pq
réalise                A-(pq)
```

et pour la page Ø

```
     C5,qq             CMP q
réalise                A-(oq)
```

- Indexé

```
     DD,qq,pp          CMP pq,X
réalise                A-(pq+X)
     D9,qq,pp          CMP pq,Y
réalise                A-(pq+Y)
     D5,qq             CMP q,X
réalise                A-(oq+X)
```

- Indirect indexé

```
     C1,dd             CMP d,X
réalise                1=(od+X)
                       h=(od+X+1)
                       A-(h1)
     D1,dd             CMP d,Y
réalise                1=(od)
                       h=(od+1)
                       A-(h1+Y)
```

Comparaison à X

```
     EØ,nn             CPX n
réalise                X-n
     EC,qq,pp          CPX pq
réalise                X-(pq)
     E4,qq             CPX q
réalise                X(-oq)
```

Comparaison à Y

CØ,nn	CPY n
réalise	Y-n
CC,qq,pp	CPY pq
réalise	Y-(pq)
C4,qq	CPY q
réalise	Y-(oq)

Nous constatons que si la soustraction ne se fait qu'avec A (la différence est restockée dans A), les comparaisons se font également avec X et Y. Ceci nous sera utile pour comparer des adresses.

Supposons qu'une adresse soit tenue en X et Y. Après une série d'incrémentation, dont le procédé a été décrit plus haut, nous comparons à une adresse connue pour savoir si celle-ci est atteinte.

Si nous recherchons l'occurrence d'un caractère dans la zone programme Basic, on termine par :

```
┌→traitement
│    "
│  CPX (9C)
├─ BNE +2-
│  CPY (9D)
└─ BNE +2-
   suite
```

9C-9D étant l'adresse de début de la zone variable. Nous pouvons également comparer à (9C) et à (9D) directement comme dans l'exemple suivant.

Rappelons comment un programme Basic est tenu dans la mémoire de l'ORIC. Ce programme commence à 128Ød.

à 5ØØh-128Ød	il y a toujours Ø
à 5Ø1 et 5Ø2h	pointeur de la ligne Basic suivante (octet bas d'abord)
à 5Ø3 et 5Ø4h	numéro de ligne courante (octet bas d'abord)
à 5Ø5 et seq.	on trouve le contenu de la ligne Basic courante. Les instructions et les fonctions sont contractées en mots-clé qui ont chacun un code plus grand que 127. Vous trouverez ces codes dans le dernier chapitre.
à la fin de la ligne Basic courante, il y a toujours un Ø	
à la fin du programme Basic il y a un second Ø, puis la zone variables commence.	

Exemple 6 : Programme pour **remplacer un caractère par un autre.**

```
37888   A9,05     ENT.    LDA 5             ;charge la zone
37890   85,0C             STA (0C)          ;   programme
37892   85,0D             STA (0D)          ;   dans 0C-0D
37894   A0,00             LDY 0
37896   B1,0C             LDA (0C),INDY     ;extrait un caractere
37898   D0,04             BNE 2+4           ;si A diff. de 0
37900   A2,05             LDX 5             ;pour passer 5 carac.
37902   D0,0A             BNE 2+10          ;saut inconditionnel
37904   C9,41             CMP 65 d          ;code de A
37906   D0,04             BNE 2+4           ;si A diff. de 65
37908   A9,4F             LDA 79            ;code de O
37910   91,0C             STA (0C),INDY     ;chamge de caractere
37912   A2,01             LDX 1             ;pour passer 1 carac.
37914   E6,0C             INC (0C)          ;incremente
37916   D0,02             BNE 2+2           ;   la zone
37918   E6,0D             INC (0D)          ;   programme
37920   A5,0C             LDA (0C)          ;compare a
37922   C5,9C             CMP (9C)          ;   l'adresse
37924   D0,06             BNE 2+6           ;   de la
37926   A5,0D             LDA (0D)          ;   zone
37928   C5,9D             CMP (9D)          ;   variables
37930   F0,05             BEQ 2+5           ;si adr. Basic=adr. Var.
37932   CA                DEX
37933   D0,EB             BNE 2-21
37935   F0,D7             BEQ 2-41
37937   60                RTS
```

Nous avons choisi de **remplacer** toutes les lettres **A** par des lettres **O.**

On extrait de la zone programme chaque caractère successivement que l'on place dans le registre A. Si ce caractère est le code de A, on le remplace par le code de O. On incrémente l'adresse de la zone programme, tenue en OC-OD et on compare à l'adresse de la zone variables, tenue en 9C-9D ; dès qu'il y a coïncidence, on saute à 37937 pour le retour au Basic.

Quand le caractère extrait est O, la partie incrémentation et comparaison est effectuée 5 fois de suite, car le pointeur de ligne suivante et le numéro de ligne peuvent contenir 65 qu'il ne faut pas changer.

Nous employons des branchements obligatoires pour réaliser des sauts inconditionnels ; avec JMP il eut fallu 3 octets. Ce n'est pas pour cette raison que nous l'avons fait, mais à titre exemplatif.

Le fait d'employer des instructions indirectes indexées, avec manipulation de OC et OD, et d'éviter l'instruction JMP, rend cette routine relogeable à n'importe quelle adresse.

Ce genre de programme permet de rechercher les occurrences d'un caractère ou d'un groupe de caractères (avec des modifications) dans un texte, donc dans une zône mémoire déterminée.

Voici un programme Basic d'exploitation :

```
5 CLS :LET X=0
10 PRINT "AIMEZ-VOUS LE VIEUX FRANCAIS?"
   :PRINT
20 IF X=1 THEN STOP
30 CALL 37888
40 LET X=1
50 GOTO 10
```

Les deux lignes s'affichent : l'originale et l'altérée. Pour recommencer, recréez la ligne 1Ø avec un autre texte.

Réalisons le transfert d'un bloc d'octets, d'un fichier, d'une zone mémoire dans une autre.

Exemple 7 : Transfert d'octets dont le **nombre est inférieur à 256.**

```
37888   A2,00          LDX 0
37890   BD,80,BB       LDA (BB80+X)   ;adr. source
37893   9D,10,BD       STA (BD10+X)   ;adr. destination
37896   E8             INX
37897   E0,A0          CPX 160d       ;4 lignes
37899   D0,F5          BNE 2-11
37901   60             RTS
```

Avec ce programme, nous transférons quatre lignes de BB8Ø-48ØØØ (lignes Ø, 1, 2 et 3) dans BD1Ø-484ØØ (lignes 1Ø, 11, 12 et 13). Pour le voir, faites les commandes :

```
LIST
CALL 37888
```

Si vous désirez ne pas voir votre écran défiler, remontez le curseur de trois lignes avant la commande 37888. Cette remarque est inutile car le lecteur sait manipuler son ORIC et son Basic résident.

Quand vous serez aguerri à la programmation en C.M., imaginez la puissance potentielle des transferts. Vous pouvez faire apparaître des "esprits" ou des "lutins" n'importe où sur l'écran ; et à l'aide du programme suivant, changer tout l'écran, plusieurs fois si vous possèdez un 48K.

Exemple 8 : Transfert de **plus de 256 octets** Pour réaliser ce transfert, nous employons deux boucles imbriquées, l'une pour contrôler le nombre de pages et l'autre pour traiter le reliquat éventuel. Les deux premières colonnes de l'écran sont changées également, pour voir... tout en évitant la ligne Ø. Nous transférons donc 1Ø79 octets soit quatre pages plus 55 octets.

```
37888   A9,D0          LDA D0          ;pour remettre les
37890   8D,11,94       STA (37905)     ;   adresses
37893   A9,BB          LDA BB          ;   aux lignes
37895   8D,16,94       STA (37910)     ;   37903 et 37908
37898   A0,04          LDY 4           ;4 pages
37900   88             DEY
37901   A2,00          LDX 0           ;pour 256 octets
37903   BD,00,D0       LDA (D000+X)    ;source dans la ROM
37906   29,EF          AND EF          ;voir explication
37908   9D,A8,BB       STA (BBA8+X)    ;destination ecran
37911   CA             DEX
37912   D0,F5          BNE 2-11
37914   EE,11,94       INC (37905)     ;augmente source et
37917   EE,16,94       INC (37910)     ;   desti. d'une page
37920   88             DEY
37921   10,EC          BPL 2-20        ;si y est positif
37923   C0,FF          CPY 255
37925   D0,04          BNE 2+4         ;si y est negatif et < 255
37927   A2,37          LDX 55d         ;reliquat
37929   D0,E4          BNE 2-28        ;saut oblige
37931   60             RTS
```

Faites la commande

```
CALL 37888
```

et voyez l'écran rempli de caractères divers. Pour voir tout l'écran, faites les 2 commandes :

```
CTRL-L
CALL 37888
```

Les quatre premières lignes remettent les adresses à leurs places car ces adresses sont manipulées en cours de programme.

On charge Y avec le nombre de pages à transférer. Changez l'octet 37899 pour un autre nombre de pages . A 379Ø3 on extrait un octet de la ROM que l'on charge dans l'écran à 379Ø8. Entre ces deux instructions, se trouve un mnémonique **AND**, dont vous aurez l'explication au paragraphe suivant. (c'est une légère anticipation).

Quand on ne transfère pas sur l'écran, cette instruction doit disparaître.

Après avoir transféré 256 octets, on incrémente les octets hauts des adresses source et destination, puis on décrémente Y. Nous avons employé le branchement **BPL** pour que, quand Y devient négatif la lère fois, donc avec la valeur 255, l'on charge X avec le reliquat à transférer ; puis quand Y est décrémenté à nouveau, il prend la valeur 254, ce qui provoque le renvoi au Basic. Changez l'octet 37928 pour un autre reliquat.

Exercice 3-3 : *Ecrivez un programme afin de transférer le fichier d'écran de 48Ø4Ø à 49119 dans la mémoire aux adresses 38144 à 39223 (95ØØh à 9937h). Supprimez l'instruction AND.*

LES OPERATIONS LOGIQUES

Toutes les opérations logiques ne se font qu'avec le registre **A**. Elles influencent les indicateurs **N** et **Z**.

L'opération ET avec une constante

29,nn	AND n	LET A=A AND n
réalise	A=A ∧ n	

est équivalent au Basic **AND** quand cette instruction est employée avec des nombres.

L'opération ET avec le contenu d'une adresse mémoire

- ***Absolu***

2D,qq,pp	AND pq
réalise	A=A ∧ (pq)
25,qq	AND q
réalise	A=A ∧ (oq)

- ***Indexé***

3D,qq,pp	AND pq,X
réalise	A=A ∧ (pq+X)

```
     39,qq,pp          AND pq,Y
réalise                A=A ∧ (pq+Y)

     35,qq             AND q,X
réalise                A=A   (oq+X)
```

- Indirect indexé

```
     21,dd             AND d,X
réalise plusieurs opérations
        l=(od+X)
        h=(od+X+1)
        A=A ∧ (hl)

     31,dd             AND d,Y
réalise l=(od)
        h=(od+1)
        A=A ∧ (hl+Y)
```

Les instructions **AND** sont fréquemment employées pour masquer certains bits de A pour les éliminer.

AND 7 élimine les bits 3 à 7 et laisse inchangés les bits Ø, 1 et 2. Cette opération réalise ainsi A modulo 8.

AND 6 élimine les bits 3 à 7 et également le bit Ø. Le résultat de cette opération donne les chiffres Ø, 2, 4 ou 6 suivant le contenu de A.

AND 7F élimine le bit 7, donc si A était plus petit que 128, A ne change pas et, si A était plus grand, le résultat est A=A-128.

Pour comprendre ces exemples, écrivez A (au choix) et le nombre en binaires, et appliquez les règles vues au chapitre I. Nous ne saurions trop vous conseiller de faire ceci.

Explication de l'exemple 8

AND EF masque le bit 4 de A, qui a le poids 16. Ceci revient à retrancher 16 de A, pour autant que A ait son bit 4 à 1. Si A=32, AND EF laisse A inchangé ; si A=31, AND EF fait A=15.

Pourquoi cette instruction est-elle nécessaire quand on charge le fichier d'affichage avec des nombres imprévisibles ? Parce que le fichier d'affichage ne supporte pas d'emmagasiner les nombres 24, 25, 28, 29, 3Ø, 31 qui déterminent les fonctions de ESCAPE, comme tous les

nombres de Ø à 31 d'ailleurs. En supprimant le bit 4, on évite ces nombres qui synchronisent sur 6Ø Hz, etc. Evidemment on supprime également d'autres nombres, comme ceux qui donnent la couleur papier, mais il fallait faire un choix, et l'écran était quand même joli... n'est-ce pas ?

L'opération OU
avec une constante

```
     Ø9,nn          ORA n          LET A=A OR n
réalise             A=A V n
```

est équivalent au Basic **OR** quand cette instruction est appliquée à deux nombres.

L'opération OU
avec le contenu d'une adresse

- Absolu

```
     ØD,qq,pp       ORA pq
réalise             A=A V (pq)

     Ø5,qq          ORA q
réalise             A=A V (oq)
```

- Indexé

```
     1D,qq,pp       ORA pq,X
réalise             A=A V (pq+X)

     19,qq,pp       ORA pq,Y
réalise             A=A V (pq+Y)

     15,qq          ORA q,X
réalise             A=A V (oq+X)
```

- Indirect indexé

```
     Ø1,dd          ORA d,X
réalise             l=(od+X)
                    h=(od+X+1)
                    A=A V (hl)

     11,dd          ORA d,Y
réalise             l=(od)
                    h=(od+1)
                    A=A V (hl+Y)
```

Les instructions **ORA** servent à mettre certains bits de A à 1 tout en laissant les autres inchangés.

ORA 12d Si A était Ø, ORA 12 met A à 12. C'est donc A=A+12 mais **seulement** avec deux indicateurs influencés. Si A avait déjà ses bits 2 et 3 à 1, A ne change pas.

ORA 15d et les bits Ø, 1, 2 et 3 de A viennent à 1.

ORA 8Øh met le bit 7 de A à 1. Si A était plus petit que 128, ceci réalise A=A+128, si non A reste inchangé.

Comme pour les opérations **AND**, écrivez A et le nombre en binaires et appliquez les règles du chapitre I.

L'opération OU exclusif avec une constante

49,nn	EOR n
réalise	A=A ⊻ n

n'a pas d'équivalent Basic.

L'opération OU exclusif avec une mémoire

- ***Absolu***

AD,qq,pp	EOR pq
réalise	A=A ⊻ (pq)
45,qq	EOR q
réalise	A=A ⊻ (oq)

- ***Indexé***

5D,qq,pp	EOR pq,X
réalise	A=A ⊻ (pq+X)
59,qq,pp	EOR pq,Y
réalise	A=A ⊻ (pq+Y)
55,qq	EOR q,X
réalise	A=A ⊻ (oq+X)

- ***Indirect indexé***

41,dd	EOR d,X
réalise	l=(od+X)
	h=(od+X+1)
	A=A ⊻ (hl)

51,dd	EOR d,Y
réalise	l=(od)
	h=(od+1)
	A=A ⊻ (hl+Y)

L'opérateur **EOR** est d'un usage difficile. Son emploi est constamment spécifique.

EOR n cette opération effectuée deux fois de suite, laisse A inchangé.

EOR O laisse A inchangé.

EOR 255 donne le complément restreint de A. En ajoutant 1, on obtient le complément vrai de A. Comme A peut se charger avec ce que l'on veut, nous disposons d'un moyen de calculer le complément d'un registre ou d'une cellule mémoire.

En appliquant les règles du chapitre I, vérifiez les trois affirmations précédentes.

LES INSTRUCTIONS BIT

BIT réalise un ET sur six bits et influence trois indicateurs.

2C,qq,pp	BIT pq
réalise	$Z=\overline{A \wedge (pq)}$
	$N=(pq)_7$
	$V=(pq)_6$
24,qq	BIT q
réalise	$Z=\overline{A \wedge (oq)}$
	$N=(oq)_7$
	$V=(oq)_6$

n'a pas d'équivalent Basic.

Si l'opération logique A ∧ (pq) donne Ø comme résultat, la barre, qui est le signe de la complémentation, amène Z=1. Le registre A ne change pas.

N prend la valeur du bit 7 de (pq).

V prend la valeur du bit 6 de (pq).

BIT nous procure le moyen de tester les bits 7 et 6 de n'importe quelle cellule mémoire. L'indicateur V n'est plus ici un indicateur de dépassement de calcul.

Ceci constitue un moyen de mettre V à 1. Il suffit de faire (pq)=64, puis un BIT pq pour avoir V=1.

La première opération réalisée par BIT, l'opération **ET** sans rangement de résultat, permet de tester si un des bits Ø à 5 de A est à Ø en mettant un masque dans (pq). Par exemple, si on fait :

(pq) = 8 = ØØØØ1ØØØb

on teste le bit 3 de A ; si A_3 = Ø, nous avons Z=1 et si A_3 = 1, nous avons Z=Ø.

On peut aussi tester plusieurs bits de A d'un coup, si

(pq) = 12d

BIT amène Z=1 **si** A_2 **et** A_3 sont à Ø.
Ceci est réversible, le test peut porter sur les bits de (pq), on emploie alors A comme masque.

Exemple 9 : Réalisons la **fonction INVERSE** qui consiste à afficher en vidéo inverse tout l'écran. Ceci ne correspond pas à LORES Ø.

```
37888   A0,00          LDY Ø
37890   A9,A7          LDA A7          ;stocke l'adresse
37892   85,0C          STA (ØC)        ;   de depart -1
37894   A9,BB          LDA BB          ;   dans
37896   85,0D          STA (ØD)        ;   ØC-ØD
37898   E6,0C          INC (ØC)        ;incremente
37900   D0,02          BNE 2+2         ;   l'adresse
37902   E6,0D          INC (ØD)
37904   B1,0C          LDA (ØC),INDY   ;extrait un caractere
37906   09,80          ORA 128d        ;l'inverse
37908   91,0C          STA (ØC),INDY   ;le stocke
37910   A5,0C          LDA (ØC)        ;compare
37912   C9,BF          CMP BF          ;   l'adresse a
37914   D0,EE          BNE 2-18        ;   l'adresse finale
37916   A5,0D          LDA (ØD)        ;   du fichier
37918   C9,DF          CMP DF          ;   qui est BFDF
37920   F0,02          BEQ 2+2
37922   D0,E6          BNE 2-26
37924   60             RTS
```

Pour afficher un caractère en vidéo inverse, il suffit d'ajouter 128 au code du caractère, ce que l'instruction **ORA** fait en ligne 379Ø6.

La fin de la routine compare l'adresse variable à l'adresse finale du fichier d'affichage soit **BFDF.** Ceci vous montre une autre possibilité de contrôler une boucle.

Entrez le Basic d'exploitation suivant :

```
10 CALL 37888
20 GET A$
30 POKE 37906,41 :POKE 37907,127
40 CALL 37888
50 POKE 37906,9 :POKE 37907,128
4999 STOP
```

La ligne 2Ø vous permet d'examiner l'écran, après quoi vous poussez sur une touche.

La ligne 3Ø transforme la ligne 379Ø6 comme suit :

379Ø6 29,7F AND 127d

qui remet à Ø tous les bits 7 de tous les caractères pour retrouver la vidéo normale.

La ligne 5Ø rétablit le ORA 128 de la ligne 379Ø6.

Faites LIST puis RUN.

Exemple 1Ø : Réalisons un programme pour **défiler l'écran de gauche à droite** en rotation complète : le dernier caractère revient en première position. Nous évitons les deux premières colonnes.

```
37888   A9,82          LDA 82          ;charge l'adresse du
37890   85,0C          STA (0C)        ;   debut d'ecran +2
37892   A9,BB          LDA BB          ;   dans
37894   85,0D          STA (0D)        ;   0C-0D
37896   A2,1B          LDX 27d         ;pour defiler 27 lignes
37898   18          ┌─►CLC             ;C=0
37899   A5,0C       │  LDA (0C)       ;pour augmenter
37901   69,28       │  ADC 40d        ;   l'adresse
37903   85,0C       │  STA (0C)       ;   de l'ecran
37905   90,02       │  BCC 2+2        ;   de 40
37907   E6,0D       │  INC (0D)       
37909   A0,25       │ ►LDY 37d        
37911   B1,0C       │  LDA (0C),INDY  ;prend le dernier carac.
37913   48          │  PHA            ;le sauve
37914   88          │  DEY            
37915   B1,0C       │ ►LDA (0C),INDY  ;prend un caractere
37917   C8          │  INY            
37918   91,0C       │  STA (0C),INDY  ;le stocke a la pos.
37920   88          │  DEY            ;   suivante
37921   88          │  DEY            
37922   10,F7       └─ BPL 2-9        ;cela 37 fois
37924   C8             INY            
37925   68             PLA            ;recupere le dernier carac.
37926   91,0C          STA (0C),INDY  ;le stocke debut de ligne
37928   CA             DEX            ;nouvelle ligne
37929   D0,DF       └─ BNE 2-33       
37931   60             RTS
```

En 37911, le caractère de la 39e colonne est saisi et stocké dans la pile. Ensuite, dans une boucle effectuée 37 fois, on décale chaque caractère d'une colonne vers la droite, puis on récupère le caractère de la pile et on l'affiche en colonne 2. On effectue la boucle 27 fois à 37898 pour augmenter l'adresse d'écran de 40 et pointer ainsi sur la ligne suivante. Entrez le Basic :

```
10 FOR A=1 TO 19
20 CALL 37888
30 NEXT
```

Faites LIST puis RUN puis RUN

Tous les Basic d'exploitation que nous donnons montrent l'action des routines en C.M.. Essayez ces routines avec vos programmes Basic personnels.

Exercice 3-4 : *En vous inspirant du programme précédent, écrivez une routine pour défiler de droite à gauche, en rotation complète. Le même Basic d'exploitation peut servir.*

Exercice 3-5 : *En vous inspirant de l'exemple 10, écrivez une routine pour défiler de gauche à droite, mais en rotation simple. Les caractères qui sortent de l'écran à droite ne réapparaissent pas à gauche.*

REPONSES AUX EXERCICES DU CHAPITRE III

Exercice 3-1

```
37888   A0,00          LDY 0
37890   B9,0F,94       LDA (37903+Y)
37893   F0,07          BEQ 2+7
37895   99,BF,BD       STA (48575+Y)
37898   C8             INY
37899   4C,02,94       JMP 37890
37902   60             RTS
37903   42,4F,4E       DEF BON
37906   4A,4F,55,52    DEF JOUR
37910   00             DEF fin de message
```

A remarquer que dans cet exercice, comme dans *l'exemple 4*, on peut profiter du fait que Y est différent de Ø et remplacer la ligne 37899 par :

```
37899   D0,F5          BNE 2-11
37901   60             RTS
37902   60             RTS
```

Si on supprime la ligne 379Ø2, il faut changer la ligne 37893 :

```
37893   F0,06          BNE 2+6
```

Faites très attention aux déplacements. Si vous changez une instruction, regardez directement si un branchement n'est pas influencé.

Exercice 3-2

1- effacer la moitié supérieure de l'écran

```
10 POKE 37899,13
20 CALL 37888
5000 STOP
```

LIST puis RUN

2- effacer la moitié inférieure de l'écran

```
10 DOKE 37905,48601 :POKE 37899,13
20 CALL 37898
```

LIST puis RUN

3- effacer la moitié gauche de l'écran

```
10 POKE 37903,19 :POKE 37899,27
20 CALL 37888
```

LIST puis RUN

4- effacer la moitié droite de l'écran

```
10 DOKE 37905,48060
20 CALL 37898
```

LIST puis RUN

Note : pour avoir le temps d'examiner l'écran, ajoutez aux quatre programmes ci-dessus.

```
30 GET A$
```

Note : Exercez-vous à effacer une fenêtre de l'écran.

Exercice 3-3

```
37888   A9,BB          LDA BB
37890   8D,11,94       STA (37905)
37893   A9,95          LDA 95
37895   8D,14,94       STA (37908)
37898   A0,04          LDY 4
37900   88             DEY
37901   A2,00          LDX 0
37903   BD,A8,BB       LDA (BBA8+X)    ;source
37906   9D,00,95       STA (9500h+X)   ;destination
37909   CA             DEX
37910   D0,F7          BNE 2-9
37912   EE,11,94       INC (37905)
37915   EE,14,94       INC (37908)
37918   88             DEY
37919   10,EE          BPL 2-18
37921   C0,FF          CPY 255
37923   D0,04          BNE 2+4
37925   A2,37          LDX 55d
37927   D0,E6          BNE 2-26
37929   60             RTS
```

N'avez-vous pas oublié de changer les branchements en 3791Ø, 37919 et 37927, ni les adresses des mnémoniques en 37895 et 37915 ? Parfait.

Après avoir fait **CALL 37888**, vous pouvez sauver sur cassette les 1Ø79 octets situés aux adresses 38144 et seq., pour un usage ultérieur éventuel.

Dans le *programme 2* de chargement, la lère ligne Basic met HIMEM à 37887 afin de nous réserver quatre pages de mémoire qui commencent à 94ØØh, 95ØØh, 96ØØh et 97ØØh.

A 98ØØh commence une zone disponible pour le Basic à condition d'employer **GRAB**. Pour nos C.M., ceci n'est pas nécessaire.

Exercice 3-4

```
37888   A9,82          LDA 82
37890   85,0C          STA (0C)
37892   A9,BB          LDA BB
37894   85,0D          STA (0D)
37896   A2,1B          LDX 27d
37898   18          ->CLC
37899   A5,0C          LDA (0C)
37901   69,28          ADC 40d
37903   85,0C          STA (0C)
37905   90,02          BCC 2+2
37907   E6,0D          INC (0D)
37909   A0,00          LDY 0              ;pour
37911   B1,0C          LDA (0C),INDY      ;saise du 1ier carac.
37913   48             PHA                ;que l'on sauve
37914   C8             INY
37915   B1,0C       ->LDA (0C),INDY       ;saisie du 2ieme carac.
37917   88             DEY                ;pour le
37918   91,0C          STA (0C),INDY      ;transferer dans le 1ier
37920   C8             INY
37921   C8             INY
37922   98             TYA                ;A=Y
37923   C9,26          CMP 38d            ;pour 37 remplacements
37925   D0,F4          BNE 2-12
37927   88             DEY
37928   68             PLA
37929   91,0C          STA (0C),INDY
37931   CA             DEX                ;pour
37932   D0,DC          BNE 2-36           ;27 lignes
37934   60             RTS
```

Pour défiler de droite à gauche, nous prélevons le caractère de la colonne 2 que l'on stocke. Ensuite on prélève le caractère de la colonne 3 que l'on place dans la colonne 2, puis colonne 4 vers colonne 3 dans la boucle contrôlée par Y. Comme Y s'incrémente, on a ajouté deux instructions pour contrôler cette boucle.

L'instruction 37922 n'est pas nécessaire si vous employez **CPY** dans l'instruction suivante. Enfin, on remet le caractère de la colonne 2 dans la colonne 39.

Exercice 3-5

```
37888   A9,82          LDA 82
37890   85,0C          STA (0C)
37892   A9,BB          LDA BB
37894   85,0D          STA (0D)
37896   A2,1B          LDX 27d
37898   18          ┌─►CLC
37899   A5,0C       │  LDA (0C)
37901   69,28       │  ADC 40d
37903   85,0C       │  STA (0C)
37905   90,02       │  BCC 2+2"
37907   E6,0D       │  INC (0D)
37909   A0,25       │  LDY 37d
37911   88          │┌►DEY
37912   B1,0C       ││ LDA (0C),INDY
37914   C8          ││ INY
37915   91,0C       ││ STA (0C),INDY
37917   88          ││ DEY
37918   10,F7       │└─BPL 2-9
37920   C8          │  INY
37921   A9,20       │  LDA 32d          ;caractere 'espace'
37923   91,0C       │  STA (0C),INDY    ;dans la colonne 2
37925   CA          │  DEX
37926   D0,E2       └──BNE 2-30
37928   60             RTS
```

On n'est pas obligé de sauver le caractère de la colonne 39. Par contre, à la fin on charge A avec le caractère "espace" pour le placer dans la colonne 2.

Remarquez également que la boucle contrôlée par Y, se renvoie plus haut et que, de ce fait, une instruction **DEY** a été supprimée juste avant le **BPL**. Ceci vous montre une autre façon de faire, en gagnant un octet.

CHAPITRE 4
HAUTE DÉFINITION, ATTRIBUTS

Après l'important chapitre III dont la compréhension est nécessaire pour transposer des boucles telles que **FOR ... TO ... STEP ...** ou **REPEAT ... UNTIL ...**, ou des branchements conditionnels tels que **IF ... THEN ... ELSE ...**, nous terminons la revue des C.M. en examinant quelques codes divers, puis les codes de glissements et de rotations.

Nous montrons ensuite l'utilisation de l'écran haute résolution.

Pour terminer nous développons quelques programmes utilitaires, démonstratifs et illustratifs avec des explications sur les attributs série.

QUELQUES CODES MACHINE DIVERS

2Ø,qq,pp	JSR pq	GOSUB pq

qui réalise les opérations suivantes :

```
PC=PC+2
(SP)=PChaut
SP=SP-1
(SP)=PCbas
SP=SP-1
PC=pq
```

Cette importante instruction, qui n'influence aucun indicateur, est équivalente au Basic **GOSUB**. Èlle s'utilise comme telle. Si nous avons recours plusieurs fois

à un sous-programme, une sous-routine, pour traiter une opération, il est préférable d'écrire une seule fois ce programme et de l'appeler plusieurs fois avec **JSR**. De même qu'un sous-programme Basic se termine par un **RETURN**, notre routine en C.M. doit se terminer par un **RTS**, que nous employons depuis le début et dont voici sa fonction :

60 RTS RETURN

réalise les opérations suivantes :

```
PCbas=(SP)
SP=SP+1
PChaut=(SP)
SP=SP+1
PC=PC+1
```

Constatons une incrémentation importante : **JSR** incrémente le PC deux fois, et **RTS** l'incrémente une fois pour pointer sur l'instruction qui suit **JSR**. Si nous avons à un endroit d'un programme :

```
32000   20,33,EC        JSR 60467d
32003                   SUITE
```

quand **JSR** s'exécute PC=32000+2 = 32002 et, à la fin de la routine située à 60467d, nous trouvons une instruction RTS qui fait PC = 32002+1 = 32003 et le programme se poursuit à l'instruction 32003.

En examinant la **mémoire morte** (ROM), nous trouvons certaines séquences où la routine qui s'exécute, pousse une adresse de retour dans la pile pour que le retour se fasse à un autre endroit. Cette nouvelle adresse est l'adresse réelle moins 1 puisque **RTS** incrémente PC.

EA NOP

ne réalise aucune opération.

Comme cette instruction dure 2 cycles d'horloge, elle peut s'employer pour calculer de courts délais quand elle se trouve dans une boucle par exemple.

Un autre usage plus fréquent que nous vous recommandons est d'insérer quelques **NOP** dans vos programmes en cours d'élaboration. Cela vous permettra de les remplacer par une instruction supplémentaire que vous n'aviez pas prévue au départ. De même quand vous supprimez une instruction, remplacez-la par des **NOP**, cela vous évitera de recalculer les déplacements des branchements qui interviennent dans le programme.

ØØ BRK

réalise les instructions suivantes :

```
PC=PC+2
(SP)=PChaut
SP=SP-1
(SP)=PCbas
SP=SP-1
(SP)=P
SP=SP-1
PCbas=(FFFE)
PChaut=(FFFF)
```

Cette instruction met l'indicateur I à 1 et l'indicateur B à 1 dans le registre P sauvé sur la pile pour différencier cette interruption programmée d'une interruption demandée par un périphérique.

Quoique n'ayant qu'un octet, cette instruction sauve le PC+2 sur la pile. Ce fait mérite d'attirer l'attention du lecteur.

4Ø RTI

réalise les instructions suivantes :

```
P=(SP)
SP=SP+1
PCbas=(SP)
SP=SP+1
PChaut=(SP)
SP=SP+1
```

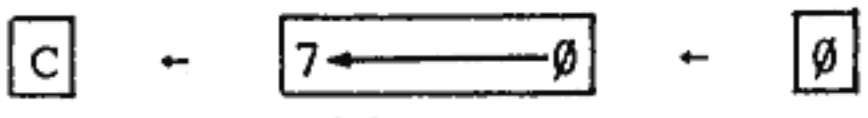
Tous les indicateurs changent puisqu'on récupère le registre P.

Le retour d'interruption termine une interruption soit programmée, soit due à un périphérique.

LES GLISSEMENTS ET LES ROTATIONS

ØA ASL

réalise un glissement de tous les bits de A vers la gauche ; Ø vient dans le bit Ø et le bit 7 va dans l'indicateur C.



sur A

encore appelée décalage arithmétique à gauche

```
     ØE,qq,pp          ASL pq
glissement à gauche des bits de (pq)

     Ø6,qq             ASL q
glissement à gauche des bits de (oq)

     1E,qq,pp          ASL pq,X
glissement à gauche des bits de (pq+X)

     16,qq             ASL q,X
glissement à gauche des bits de (oq+X)
```

Il est évident que ce décalage à gauche correspond à une multiplication par 2, puisque chaque bit qui se trouve à gauche d'un autre possède un poids qui lui est deux fois supérieur.

Si (oq) était

| Ø | Ø | Ø | Ø | 1 | Ø | Ø | 1 |

soit 9

il devient après Ø6,qq

| Ø | Ø | Ø | 1 | Ø | Ø | 1 | Ø |

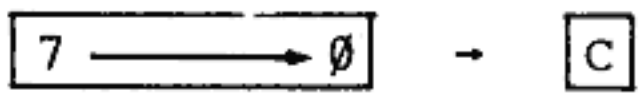
soit 18d

Pour savoir si le résultat est modulo 256, il faut tester l'indicateur C.

Les instructions **ASL** influencent les indicateurs N, Z et C.

```
     4A                LSR
```

réalise un glissement de tous les bits de A vers la droite. Ø entre dans le bit 7 et le bit Ø va dans C.



sur A

appelée également décalage logique à droite.

```
     4E,qq,pp          LSR pq
glissement à droite des bits de (pq)

     46,qq             LSR q
glissement à droite des bits de (oq)

     5E,qq,pp          LSR pq,X
glissement à droite des bits de (pq+X)

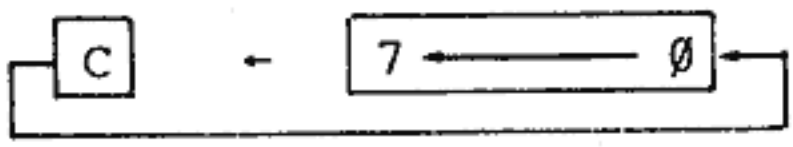
     56,qq             LSR q,X
glissement à droite des bits de (oq+X)
```

Ce n'est plus un décalage arithmétique car si ce glissement réalise bien une division par 2 pour les nombres pairs, cette division est arrondie pour les nombres impairs. Ceux-ci ont toujours le bit Ø à 1, qui entre dans l'indicateur C, et le test sur C renseigne si la division est arrondie.

Si une cellule mémoire contient 17d, après un LSR pq, elle contient 8 et l'indicateur C est à 1.

Les instructions **LSR** influencent les indicateurs Z et C, et mettent N à Ø.

2A ROL
réalise une rotation de 9 bits à gauche sur A. Tous les bits de A glissent vers la gauche, C vient dans le bit Ø et le bit 7 va dans C.



sur A

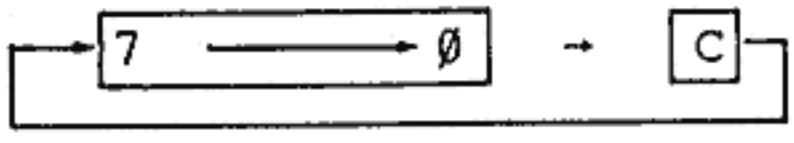
2E,qq,pp ROL pq
rotation de 9 bits à gauche sur (pq)

26,qq ROL q
rotation de 9 bits à gauche sur (oq)

3E,qq,pp ROL pq,X
rotation de 9 bits à gauche sur (pq+X)

36,qq ROL q,X
rotation de 9 bits à gauche sur (oq+X)

6A ROR
réalise une rotation de 9 bits à droite sur A. Tous les bits de A glissent vers la droite, C entre dans le bit 7 et le bit Ø entre dans C.



sur A

6E,qq,pp ROR pq
rotation de 9 bits à droite sur (pq)

66,qq ROR q
rotation de 9 bits à droite sur (oq)

7E,qq,pp ROR pq,X
rotation de 9 bits à droite sur (pq+X)

76,qq ROR q,X
rotation de 9 bits à droite sur (Oq+X)

Toutes les rotations influencent les indicateurs N, Z et C.

Pour **multiplier par 2** un nombre qui tient en 16 bits, on fait :

```
ASL (pq)
ROL (pq+1)
```

suivi d'un test sur C si on veut savoir si le résultat est modulo 65536.

Si l'opération envisagée se fait sur 24 bits :

```
ASL (pq)
ROL (pq+1)
ROL (pq+2)
```

d'où l'intérêt des instructions indexées.

Pour **multiplier A par 3** :

```
STA (pq)
ASL
ADC (pq)
```

à condition que 3 fois A ne dépasse pas 255 ou 511 si on fait un test sur C.

Comme les glissements et rotations agissent au niveau du bit, cela correspond sur l'écran au niveau du pixel, donc en **HIRES.**

LES ATTRIBUTS SERIE

Les attributs modifient la présentation des caractères sur l'écran. Ils occupent chacun une position dans le fichier d'affichage et influencent le reste de la ligne dans laquelle ils se trouvent, d'où leurs noms d'attributs série. Quand il s'agit d'influencer toute la ligne, on introduit l'attribut dans la première position, la colonne Ø, de la ligne. C'est ce que fait la commande **PAPER** pour attribuer une couleur de fond, ou couleur papier, mais à toutes les lignes. La commande **INK** fait de même pour la couleur des caractères, mais cet attribut est placé évidemment dans la colonne 1 puisque la colonne Ø est déjà occupée.

Voyons les codes des attributs (**tableau 1**) que l'on peut introduire dans le fichier d'affichage. Comme nous

travaillons en C.M. cette introduction est réalisée par des instructions **STA**, **STX** et **STY**, soit des instructions qui correspondent aux POKE du Basic, et non aux instructions Basic de la forme PRINT CHR$ (x).

Codes	*Attribut*
Ø	encre noire
1	encre rouge
2	encre verte
3	encre jaune
4	encre bleue
5	encre magenta
6	encre cyan
7	encre blanche
8	caractère standard
9	caractère graphique
1Ø	caractère standard, hauteur double
11	caractère graphique, hauteur double
12	caractère standard clignotant
13	caractère graphique clignotant
14	caractère standard clignotant, hauteur double
15	caractère graphique clignotant, hauteur double
16	papier noir
17	papier rouge
18	papier vert
19	papier jaune
2Ø	papier bleu
21	papier magenta
22	papier cyan
23	papier blanc

Tableau 1- Codes des attributs

Les attributs suivants, de **24 à 31**, s'emploient pour la synchronisation de l'image suivant la norme de télévision employée. (différente aux U.S.A., le réseau étant en 6Ø Hz).

Les codes **32 à 127**, introduits par C.M. ou par POKE, correspondent au jeu de caractères ASCII.

Les codes **128 à 255** sont identiques aux codes Ø à 127, mais l'attribut ou le caractère est inversé puisqu'on a ajouté 128 aux codes concernés (comme l'exemple 9 du chapitre précédent). L'inversion n'étant pas un attribut,

n'influence que l'octet où il apparaît et non le reste de la ligne. L'inversion d'une couleur correspond à sa couleur complémentaire.

Faites les commandes :

```
CTRL-L
POKE 48Ø4Ø,17
```

et la couleur papier de la lère ligne est rouge. Faites :

```
POKE 48Ø4Ø,128+17
```

et vous voyez que la couleur papier est cyan dans la lère position et rouge dans le reste de la ligne.

En **mode TEXT**, les colonnes Ø sont remplies de code 23, et les colonnes 1 de code Ø.

En **mode LORES Ø**, les colonnes Ø sont remplies de code 8, et les colonnes 1 de code 16.

En **mode LORES 1**, les colonnes Ø sont remplies de code 9, et les colonnes 1 de code 16.

Dans les deux **modes LORES**, l'encre est blanche par défaut, de plus, le mode TEXT réapparaît quand l'écran défile, sauf si nous employons des C.M. pour provoquer ce défilement vertical.

Exemple 1 : Faisons **défiler l'écran d'une ligne** vers le haut, en rotation complète, de façon que la première ligne se réaffiche en dernière ligne.

Comme nous ne changeons pas les attributs des deux premières colonnes, nous remplaçons les 4Ø caractères d'une ligne par les 4Ø caractères de la ligne suivante. La première ligne sera mise dans un tampon et, à la fin, nous remettons ce tampon dans la dernière ligne.

```
37888                       DEF tampon de 40 caracteres
37928   A0,27      S.ROU    LDY 39
37930   B1,0C                LDA (0C),INDY  ;source
37932   91,0E                STA (0E),INDY  ;destination
37934   88                   DEY
37935   10,F9                BPL 2-7
37937   60                   RTS
37938   A9,A8      ENT.      LDA A8          -;BBA8 est
37940   85,0C                STA (0C)        ;   l'adresse
37942   A9,BB                LDA BB          -;   de la
37944   85,0D                STA (0D)        ;   liere ligne
37946   A9,00                LDA 0           ;9400h est
37948   85,0E                STA (0E)        ;   l'adresse
37950   A9,94                LDA 94h         ;   du
37952   85,0F                STA (0F)        ;   tampon
37954   20,28,94             JSR 37928       ;transfert 40 carac.
37957   A9,80                LDA 80h         -;8880 est
37959   85,0E                STA (0E)        ;   l'adresse
```

→

```
37961   A9,BB      LDA BB          -;  de la
37963   85,0F      STA (0F)        ;   ligne 0
37965   A2,1A      LDX 26d         ;26 translations
37967   A5,0C      LDA (0C)        ;ajoute
37969   18         CLC             -;   40
37970   69,28      ADC 40d         -;   a
37972   85,0C      STA (0C)        ;   l'adresse
37974   90,02      BCC 2+2         -;   source
37976   E6,0D      INC (0D)        -;
37978   A5,0E      LDA (0E)        ;ajoute
37980   18         CLC             -;   40
37981   69,28      ADC 40 d        -;   a
37983   85,0E      STA (0E)        ;   l'adresse
37985   90,02      BCC 2+2         -;   destination
37987   E6,0F      INC (0F)        -;
37989   20,28,94   JSR 37928       ;transfert
37992   CA         DEX             ;26 fois
37993   D0,E4      BNE 2-28
```

```
37995   A9,00      LDA 0           ;adresse
37997   85,0C      STA (0C)        ;   tampon
37999   A9,94      LDA 94 h
38001   85,0D      STA (0D)
38003   A9,B8      LDA B8          -;adresse
38005   85,0E      STA (0E)        ;   derniere
38007   A9,BF      LDA BF          -;   ligne
38009   85,0F      STA (0F)
38011   20,28,94   JSR 37928       ;transfert
38014   60         RTS
```

Nous faisons appel à une sous-routine qui se situe de 37928 à 37937. Cette sous-routine transfère une ligne d'adresse source vers une ligne d'adresse destination.

La routine commence à 37938. L'adresse source est chargée avec l'adresse de la lère ligne, et l'adresse destination avec celle du tampon, puis on réalise le transfert par appel à la sous-routine.

Avant de construire la boucle pour effectuer 26 transferts, nous changeons l'adresse de destination. La boucle s'exécute alors en ajoutant chaque fois 4Ø aux 2 adresses source et destination avant l'appel à la sous-routine.

En fin de programme, on manipule les adresse pour transférer le tampon dans la dernière ligne.

Voici le Basic d'exploitation :

```
5 CLS
10 PRINT "CECI DEMONTRE LE DEFILEMENT
VERTICAL"
11 PRINT "DE L'ECRAN, A L'AIDE D'UN
PROGRAMME"
```

```
12 PRINT "EN CODES MACHINE."
13 PRINT "POUSSEZ SUR UNE TOUCHE":GET A$
20 FOR P=1 TO 10
30 CALL 37938
40 NEXT
5000 STOP
```

Exercice 4-1 : *Réalisez le défilement de l'écran vers le bas.*

Dans la colonne des mnémoniques de l'exemple 1, nous avons ajouté le signe - pour vous indiquer les instructions qui changent, afin de vous faciliter la tâche.

Exemple 2 : Remplissons l'écran de **bandes de 4 cases** de différentes couleurs.

```
37888   A9,A8          LDA A8          ;adresse de
37890   85,0C          STA (0C)        ;   depart de
37892   A9,BB          LDA BB          ;   l'ecran
37894   85,0D          STA (0D)        ;   BBA8
37896   A0,00          LDY 0
37898   A2,10          LDX 16d         ;papier noir
37900   8A             TXA
37901   29,17          AND 17h         ;voir explications
37903   EA             NOP
37904   EA             NOP
37905   AA             TAX
37906   91,0C          STA (0C),INDY   ;met l'attribut
37908   18             CLC
37909   A5,0C          LDA (0C)        ;ajoute 4
37911   69,04          ADC 4           ;   a l'adresse
37913   85,0C          STA (0C)        ;   du fichier
37915   90,02          BCC 2+2         ;   d'ecran
37917   E6,0D          INC (0D)
37919   E8             INX             ;autre couleur
37920   A5,0C          LDA (0C)        ;compare
37922   C9,E0          CMP E0          ;   avec
37924   D0,E6          BNE 2-26        ;   l'adresse
37926   A5,0D          LDA (0D)        ;  de fin
37928   C9,BF          CMP BF          ;   d'ecran
37930   D0,E0          BNE 2-32
37932   60             RTS
```

Nous utilisons **l'instruction indirecte indexée**, bien connue des lecteurs, en 37906, pour charger l'attribut dans le fichier d'affichage. Ceci nécessite les cinq premières instructions pour l'initialisation.

En 37898, le registre X est initialisé à 16 qui est l'attribut de la couleur papier noir. En 37900, X est

transféré en A pour le traitement. L'instruction AND 17h en 379Ø1 limite les valeurs de X entre 16 et 23. Vous voyez pourquoi ?

Reprenons encore une fois, le fonctionnement de AND :

17h = ØØØ1Ø111b

ce masque ne conserve que les bits de poids 1, 2, 4, 16 et, toutes les fois que X sera incrémenté, il aura pour valeur 16 à 23, puis 16 à 23, ... inlassablement.

En 379Ø5 X reprend la valeur de A. Ensuite on ajoute 4 à l'adresse du fichier d'affichage puis on compare à l'adresse de la fin du fichier pour sortir de la boucle. Cette façon de faire peut être dangereuse, comme en Basic d'ailleurs. Nous avons ici un multiple de 4, le branchement BNE est valable.

Il est préférable de tester si une adresse est supérieure à une autre plutôt que leur égalité.

Pour visualiser, employez le Basic :

```
5 CLS
10 PRINT CHR$(17)
20 CALL 37888
30 GET A$
80 PRINT CHR$(17)
5000 STOP
```

Si on remplace les 2 NOP par l'instruction :

37903 09,80 ORA 128d

la lère position de chaque bande sera inversée.

Ajoutez les lignes suivantes au Basic précédent :

```
40 POKE 37903,9 :POKE37904,128
50 CALL 37888
60 GET A$
70 POKE 37903,234 :POKE 37904,234
```

En poussant sur une touche, vous obtenez la seconde illustration.

Exemple 3 : Remplissons l'écran de **bandes verticales** de différentes couleurs.

```
37888   A2,10          LDX 16          ;papier noir
37890   A0,00          LDY 0
37892   A9,A8          LDA A8          ;adresse
37894   85,0C          STA (0C)        ;    de
37896   A9,BB          LDA BB          ;    depart
37898   85,0D          STA (0D)
37900   A9,1B          LDA 27d         ;pour 27 lignes
37902   85,10          STA (10)        ;compteur de lignes
37904   A9,09          LDA 9           ;pour 9 cases
37906   85,11          STA (11)        ;compteur de cases
37908   8A             TXA
37909   29,17          AND 17h         ;masque
37911   AA             TAX
37912   91,0C          STA (0C),INDY
37914   C6,10          DEC (10)
37916   F0,12          BEQ 2+18
37918   18             CLC
37919   A5,0C          LDA (0C)        ;ajoute 40
37921   69,28          ADC 40          ;    a l'adresse
37923   85,0C          STA (0C)        ;
37925   90,02          BCC 2+2         ;
37927   E6,0D          INC (0D)        ;
37929   C6,11          DEC (11)
37931   D0,E7          BNE 2-25
37933   E8             INX             ;autre couleur papier
37934   D0,E0          BNE 2-32        ;saut inconditionnel
37936   EE,05,94       INC (37893)     ;augmente la colonne
37939   A9,CF          LDA CF          ;pour remplir
37941   CD,05,94       CMP (37893)     ;    40 colonnes
37944   10,CA          BPL 2-54
37946   A9,A8          LDA A8          ;remet l'adresse
37948   8D,05,94       STA (37893)     ;    de depart
37951   60             RTS
```

On ajoute 40 à l'adresse de départ un certain nombre de fois déterminé par le compteur (11) pour charger autant de cases avec la même couleur. La boucle interne réalise ceci. La seconde boucle change de couleur puis recommence la boucle précédente. Quand le compteur de lignes a compté 27 lignes, on incrémente l'adresse de départ pour pointer dans la 2e colonne, puis on renvoie au départ ; c'est le travail de la boucle externe.

Pour sortir de cette boucle, on compare l'adresse de départ avec l'adresse de la 40e colonne, quand celle-ci est atteinte, on restaure l'adresse de départ, puis on retourne au Basic.

Faites CALL 37888 pour voir l'effet.

Faisons varier la longueur des cases verticales à l'aide du Basic suivant :

```
5 CLS
10 FOR P=1 TO 15
```

```
20 POKE 37905,P
30 CALL 37888
40 WAIT 100
50 NEXT
5000 STOP
```

Si vous désirez ne remplir que 2Ø lignes, faites :

```
POKE 379Ø1,2Ø
```

Si vous voulez arrêter le remplissage après la 3Øe colonne, faites :

```
POKE 3794Ø,197 :POKE 379Ø1,27
```

L'ECRAN HAUTE RESOLUTION

Cet écran est employé en mode **HIRES.** La ligne d'état, réservée en basse résolution est employée. Les trois dernières lignes restent en mode **TEXT.** Il nous reste 25 lignes de caractères qui se transforment en lignes de pixels. Chaque caractère est formé par 8 pixels en hauteur sur 6 pixels en largeur, nous avons alors 2ØØ lignes pixels de 4Ø octets, soit 24Ø colonnes pixels. En tout 8ØØØ octets.

Quand nous parlons de lignes et de colonnes en haute résolution, il est entendu qu'il s'agit de lignes et de colonnes pixels.

	lignes pixels ↓				
colonnes →		Ø-5	6-11	x 234-239	
pixels	Ø	4Ø96Ø	4Ø961 ...	4Ø999	AØØØ-AØ27
	1	41ØØØ		41Ø39	AØ28-AØ4F
	↓				
	y				
	199	4892Ø		48959	BF18-BF3F
		4896Ø	ligne de réserve	48999	BF4Ø-BF67
25e l. carac.		49ØØØ	3 lignes	49Ø39	BF68-BF8F
26e l. carac.		49Ø4Ø	de	49Ø79	BF9Ø-BFB7
27e l. carac.		49Ø8Ø	textes	49119	BFB8-BFDF

Tableau 2- Ecran en haute résolution, répartition des octets

Chaque octet peut contenir des attributs série ou des pixels. Les attributs sont les mêmes que ceux du tableau 1, excepté les caractères graphiques et les hauteurs doubles (codes 8 à 11), mais le clignotement agit (codes 12 à 15). Ceci est évident puisqu'on n'affiche ni caractères normaux ni caractères graphiques mais des pixels. Les codes 24 à 31 sont à éviter également.

Pour afficher des pixels, on charge les 6 premiers bits d'un octet (bits Ø à 5).

32	16	8	4	2	1	poids du bit
5	4	3	2	1	Ø	position du bit

Pour afficher le bit Ø, on charge l'octet avec la valeur 1, pour afficher le bit 1, on charge la valeur 2, etc. Pour afficher plusieurs pixels, on charge la somme des valeurs concernées.

ATTENTION : Vous désirez afficher le pixel de position 2, vous chargez donc l'octet avec la valeur 4. Mais ceci donne l'attribut encre bleue. Ici intervient le bit 6 de l'octet : nous le mettons à 1 pour signifier que l'on veut afficher des pixels. Vous devez donc charger l'octet avec la valeur 64+4 soit 68. Ceci est facile en C.M. avec l'instruction **ORA** (de même qu'un **AND** impose un attribut).

Le bit 7 de l'octet a la même fonction qu'en mode **TEXT**, il provoque l'inversion vidéo.

La conséquence de ceci est que, dès que vous passez en mode **HIRES**, tous les octets de 4Ø96Ø à 48959 contiennent la valeur 64.

Exemple 4 : Remplissons les 2ØØ premiers octets de chaque ligne d'un **attribut couleur papier**, en changeant de couleur à chaque ligne.

Nous introduisons **4 NOP** dans le programme pour effectuer une modification ultérieure.

```
37888   A2,10          LDX 16d          ;papier noir
37890   A0,C8          LDY 200d         ;pour 2ØØ lignes
37892   8A          ->TXA
37893   29,17          AND 17h
37895   AA             TAX
37896   8D,00,A0       STA (40960)      ;lier octet liere ligne
37899   AD,09,94       LDA (37897)
37902   18             CLC
37903   69,28          ADC 40d
37905   8D,09,94       STA (37897)
37908   90,03          BCC 2+3
37910   EE,0A,94       INC (37898)
37913   88          ->DEY
```

```
37914   F0,07          BEQ 2+7
37916   EA             NOP
37917   EA             NOP
37918   EA,EA          NOP NOP
37920   E8             INX
37921   D0,E1          BNE 2-31        ;saut oblige
37923   A9,00          LDA 0           ;restaure
37925   8D,09,94         STA (37897)   ;   l'adresse
37928   A9,A0            LDA A0        ;   de
37930   8D,0A,94         STA (37898)   ;   depart
37933   60               RTS
```

Faites HIRES puis CALL 37888

Chaque ligne d'écran se colore d'une des huit couleurs, du noir au blanc, successivement.

Il nous devient possible de colorier un caractère avec huit couleurs différentes horizontalement.

Exercice 4-2 : *Au lieu d'afficher une ligne, remplacez les 4 NOP par 3 instructions pour afficher 2 lignes de même couleur, en changeant de couleur toutes les 2 lignes.*

Exemple 5 : Défilement de l'écran **vers la droite, pixel par pixel.**

```
37888   00                DEF octet de rotation memoire
37889   00                DEF reserve
37890   A9,02     ENT.    LDA 2           ;restaure
37892   8D,1A,94          STA (37914)     ;   l'adresse
37895   8D,2A,94          STA (37930)     ;   de depart
37898   A9,A0             LDA A0          ;   A002 a deux
37900   8D,1B,94          STA (37915)     ;   endroits du
37903   8D,2B,94          STA (37931)     ;   programme
37906   A0,C8             LDY 200d        ;pour 200 lignes
37908   A2,00           LDX 0             ;initialise X
37910   4E,00,94        LSR (37888)       ;pour bit 7=0
37913   BD,02,A0      LDA (40962+X)       ;traitement
37916   4A              LSR               ;   voir texte
37917   6E,00,94        ROR (37888)       ;
37920   09,40           ORA 64d           ;bit 6 a 1
37922   2C,00,94        BIT (37888)
37925   70,02           BVS 2+2
37927   29,DF           AND DF
37929   9D,02,A0      STA (40962+X)
37932   E8              INX               ;pour 38 octets
37933   E0,26           CPX 38d           ;   par ligne
37935   90,E8           BCC 2-24
37937   18              CLC               ;calcule les
```

→

```
37938   AD,1A,94        LDA (37914)     ;   nouvelles
37941   69,28           ADC 40          ;   adresses
37943   8D,1A,94        STA (37914)     ;   a deux
37946   8D,2A,94        STA (37930)     ;   endroits
37949   90,06           BCC 2+6
37951   EE,1B,94        INC (37915)
37954   EE,2B,94        INC (37931)
37957   88              DEY             ;nouvelle ligne
37958   D0,CC           BNE 2-52
37960   60              RTS
```

Entrez le Basic suivant :

```
5 HIRES :PAPER 4:INK 3
10 FOR A=20 TO 60 STEP 20
20 CURSET 80,100,0
30 CIRCLE A,1
40 NEXT
50 FOR B=0 TO 80
60 CALL 37890
70 NEXT
5000 STOP
```

Faites **RUN** et voyez les 80 déplacements de l'écran vers la droite.

Pour n'avoir que le déplacement de la moitié supérieure de l'écran, faites POKE 37907,100 puis RUN.

Explication du procédé employé

Chaque octet de l'écran glisse à droite d'un pixel, nous récupérons le bit 0 de cet octet pour le placer dans le bit 5 de l'octet suivant mais après que celui-ci ait lui-même glissé à droite. Nous devons donc garder cette information pendant 2 glissements. Puisque nous affichons des pixels, le bit 6 de chaque octet doit rester à 1 pour éviter de transformer un arrangement de pixels en un attribut. Pour cette raison également, nous devons effectuer le traitement ailleurs que dans le fichier d'affichage. A cet égard nous employons le registre A et un octet tampon situé à 37888. Voyons la réalisation :

En **37913** nous extrayons un octet d'affichage pour le placer dans le registre A.

En **37916** A subit un glissement logique à droite, ce qui a pour effet d'introduire un 0 dans le bit 7 et de mettre le bit 0 dans l'indicateur C et de faire glisser

tous les bits d'un rang vers la droite : nous avons donc automatiquement un 1 dans le bit 5 puisque le bit 6 était à 1.

En **37917** nous récupérons le bit Ø (indic C), grâce à une rotation à droite de l'octet tampon, et le plaçons dans le bit 7 de ce tampon. Le bit 7 de celui-ci vient dans le bit 6.

En **3792Ø** nous forçons un 1 dans le bit 6 de A.

Quand nous commençons le traitement d'une ligne (par son 3e octet, les 2 premiers étant réservés pour les couleurs papier et encre), l'octet tampon glisse une lère fois à droite (en 3791Ø) pour avoir son bit 7 à Ø. Avec la rotation en 37917, nous avons Ø dans le bit 6. Pour les 37 octets suivants de chaque ligne nous aurons dans le bit 6, le contenu du bit Ø de l'octet précédent.

En **37922** l'instruction **BIT** a pour but de placer le bit 6 du tampon dans l'indicateur V.

En **37925** si V=1 on saute à 37929 où l'on remet l'octet traité à sa place, celui-ci a son bit 5 à 1 (ce qu'a fait la ligne 37916).

Par contre si V=Ø, on force un Ø dans le bit 5 avec l'instruction **AND DF** avant de remettre l'octet traité à sa place.

Le reste du programme ne présente aucune difficulté pour le lecteur attentif. Ce programme n'est pas relogeable sans modifications d'adresses.

Exemple 6 : Réalisons le **défilement de l'écran à gauche**, pixel par pixel. Ceci peut se réaliser à titre d'exercice. Dans ce cas, refermez le livre et écrivez le programme.

```
37888   00                  DEF octet tampon
37889   00                  DEF  reserve
37890   A9,02      ENT      LDA 2             ;restaure
37892   8D,1E,94            STA (37918)       ;   l'adresse
37895   8D,29,94            STA (37929)       ;   de depart
37898   A9,A0               LDA A0            ;
37900   8D,1F,94            STA (37919)       ;
37903   8D,2A,94            STA (37930)       ;
37906   A0,C8               LDY 200d
```

→

```
37908   A2,25       LDX 37d
37910   18          CLC             ;traitement
37911   2C,00,94    BIT (37888)     ;
37914   50,01       BVC 2+1
37916   38          SEC
37917   BD,02,A0    LDA (40962+X)
37920   2A          ROL
37921   8D,00,94    STA (37888)
37924   09,40       ORA 64d         ;bit 6 a 1
37926   29,7F       AND 7F          ;bit 7 a 0
37928   9D,02,A0    STA (40962+X)
37931   CA          DEX             ;pour 38
37932   10,E8       BPL 2-24        ;   rotations
37934   18          CLC
37935   AD,1E,94    LDA (37918)     ;calcul de la
37938   69,28       ADC 40          ;   nouvelle
37940   8D,1E,94    STA (37918)     ;   adresse
37943   8D,29,94    STA (37929)     ;
37946   90,06       BCC 2+6
37948   EE,1F,94    INC (37919)
37951   EE,2A,94    INC (37930)
37954   A9,00       LDA 0           ;remet (37888)
37956   8D,00,94    STA (37888)     ;   a 0
37959   88          DEY             ;ligne suivante
37960   D0,CA       BNE 2-54
37962   60          RTS
```

Même Basic que l'exemple précédent, à l'exception de la ligne 2Ø qui devient :

```
20 CURSET 160,100,0
```

Faites également, après essai, POKE 379Ø7,1ØØ

Le dernier octet d'une ligne subit une rotation à gauche. Son bit 5 vient dans le bit 6 (que l'on stocke dans 37888). L'indicateur C vient dans le bit Ø. Au premier passage C est à Ø. Aux passages suivants C est mis à Ø ou à 1 selon le bit 6 de 37888 qui contient le bit 5 de l'octet précédent. Ceci est réalisé par les lignes 3791Ø à 37916. Le reste du programme va de soi.

Exemple 7 : Réalisons le **défilement de bas en haut**, pixel par pixel.

```
37888   A9,2A       LDA 2A          ;restaure
37890   8D,17,94    STA (37911)     ;   l'adresse
37893   A9,A0       LDA A0          ;   de depart
37895   8D,18,94    STA (37912)     ;
37898   8D,1B,94    STA (37915)     ;
37901   A9,02       LDA 2           ;
37903   8D,1A,94    STA (37914)     ;
37906   A0,C7       LDY 199d
```

```
37908   A2,25       LDX 37d
37910   BD,2A,A0    LDA (41002+X)   ;transfert
37913   9D,02,A0    STA (40962+X)   ;   d'une
37916   CA          DEX             ;   ligne
37917   10,F7       BPL 2-9         ;   a l'autre
37919   18          CLC
37920   AD,17,94    LDA (37911)     ;transfert
37923   8D,1A,94    STA (37914)     ;   d'une adresse
37926   69,28       ADC 40          ;   et calcul
37928   8D,17,94    STA (37911)     ;   d'une
37931   AD,18,94    LDA (37912)     ;   nouvelle
37934   8D,1B,94    STA (37915)     ;
37937   90,03       BCC 2+3         ;
37939   EE,18,94    INC (37912)     ;
37942   88          DEY
37943   D0,DB       BNE 2-37
37945   60          RTS
```

On transfère les 38 octets de la 2e ligne dans la lre ligne, puis ceux de la 3e dans la 2e, et ainsi de suite jusqu'à la dernière.

Pour visualiser, entrez le Basic suivant :

```
5 HIRES
10 FOR A=20 TO 80 STEP 20
20 CURSET 120,100,0
30 CIRCLE A,1
40 NEXT
50 FOR B=0 TO 100
60 CALL 37888
70 NEXT
5000 STOP
```

Exercice 4-3 : *Ecrivez un programme pour défiler l'écran de haut en bas, pixel par pixel.*

REPONSES AUX EXERCICES DU CHAPITRE IV

Exercice 4-1

Reprenez *l'exemple 1* et changez les lignes suivantes :

```
37938   A9,B8          LDA B8          ;adresse derniere
37942   A9,BF          LDA BF          ;   ligne
37957   A9,E0          LDA E0          ;adresse derniere
37961   A9,BF          LDA BF          ;   ligne + 40
37969   38             SEC
37970   E9,28          SBC 40d
37974   B0,02          BCS 2+2
37976   C6,0D          DEC (0D)
37980   38             SEC
37981   E9,28          SBC 40d
37985   B0,02          BCS 2+2
37987   C6,0F          DEC (0F)
38003   A9,A8          LDA A8          ;adresse
38007   A9,BB          LDA BB          ;   liere ligne
```

Les adresses changent et diminuent au lieu d'augmenter.

La dernière ligne va dans le tampon.

La ligne pénultienne va dans la dernière, etc.

Et pour terminer, le tampon va dans la lre ligne.

Exercice 4-2

```
37916   98             TYA
37917   6A             ROR
37918   B0,E4          BCS 2-28
```

Comme Y décroît régulièrement, Y est alternativement pair puis impair. Après avoir chargé A avec la valeur de Y, on effectue une rotation à droite pour amener le bit Ø dans l'indicateur C. Celui-ci sera une fois sur deux à 1 et par conséquent **INX** ne sera exécuté également qu'une fois sur deux.

Cet exercice, un peu difficile pour un néophyte, montre une technique de programmation pour exécuter quelque chose une fois sur deux. On pourra, par exemple, déplacer un objet deux fois plus vite qu'un autre.

Exercice 4-3

```
37888   A9,F0          LDA F0
37890   8D,19,94       STA (37913)
37893   A9,BE          LDA BE
37895   8D,1A,94       STA (37914)
37898   A9,18          LDA 18
37900   8D,1C,94       STA (37916)
37903   A9,BF          LDA BF
37905   8D,1D,94       STA (37917)
37908   A0,C7          LDY 199d
37910   A2,25          LDX 37d
37912   BD,F0,BE       LDA (48880+X)
37915   9D,18,BF       STA (48920+X)
37918   CA             DEX
37919   10,F7          BPL 2-9
37921   38             SEC
37922   AD,19,94       LDA (37913)
37925   8D,1C,94       STA (37916)
37928   E9,28          SBC 40d
37930   8D,19,94       STA (37913)
37933   AD,1A,94       LDA (37914)
37936   8D,1D,94       STA (37917)
37939   B0,03          BCS 2+3
37941   CE,1A,94       DEC (37914)
37944   88             DEY
37945   D0,DB          BNE 2-37
37947   60             RTS
```

Ce programme ne nécessite aucun commentaire.

Même Basic d'exploitation que pour *l'exemple 7*.

Voyez-vous quel beau coucher de soleil pourrait se réaliser ?

CHAPITRE 5
INTRODUCTION AUX TECHNIQUES AVANCÉES

Après avoir expliqué les derniers codes machine du microprocesseur 65Ø2A, le chapitre IV a montré la disposition des octets dans le fichier d'affichage en haute résolution et donné quelques programmes d'illustration.

Ce chapitre sert d'introduction à la programmation avancée. Quelques programmes utilitaires seront décrits. La production des délais programmés et des nombres aléatoires seront ensuite abordées. Ce sont des éléments essentiels de tout programme de jeu. L'animation d'objets et leurs diversités nécessitent l'examen du clavier pour permettre l'intervention et la formation de caractères définis par l'usager pour permettre la création d'objets.

QUELQUES PROGRAMMES

Réalisons la commande **CHAR** pour pouvoir afficher sur l'écran en haute résolution. Le point de départ de l'affichage est le point pixel de coordonnées (x,y). Nous devons calculer ce point dans un programme en lui fournissant les coordonnées (x,y). Ce programme réalise la commande **CURSET.**

Nous construisons ensuite un programme qui donne la matrice de formation du caractère, en fournissant à ce programme un code ASCII.

Le programme **CHAR** appelle ces deux programmes qui, d'autre part, pourront servir à une foule d'autres usages.

Exemple 1 : Réalisons la **commande CURSET.**

Nous réservons l'octet 37888 pour tenir l'abscisse x et l'octet 37889 pour tenir l'ordonnée y.

Nous devons remplir ces 2 octets avant l'appel de la routine : Ø à 239 pour x et Ø à 199 pour y.

```
37888   00                  DEF coord. x
37889   00                  DEF coord. y
37890   00,00               DEF adresse ecran
37892   00                  DEF position pixel (0 a 5)
37893   A9,9F      CURSET   LDA 9F          ;adresse de
37895   8D,03,94            STA (37891)     ;   depart - 40
37898   A9,D8               LDA D8          ;   soit
37900   8D,02,94            STA (37890)     ;   40920
37903   AC,01,94            LDY (37889)
37906   C8                  INY
37907   18                  CLC             ;actualise
37908   69,28               ADC 40          ;   l'adresse au
37910   90,03               BCC 2+3         ;   depart de
37912   EE,03,94            INC (37891)     ;   la yieme ligne
37915   88                  DEY
37916   D0,F5               BNE 2-11
37918   8D,02,94            STA (37890)     ;la stocke
37921   AD,00,94            LDA (37888)
37924   AA                  TAX
37925   E0,06               CPX 6
37927   90,0C               BCC 2+12        ;si x plus petit que 6
37929   E9,06               SBC 6           ;A=A-6
37931   EE,02,94            INC (37890)     ;l'adresse
37934   D0,F4               BNE 2-12        ;   est
37936   EE,03,94            INC (37891)     ;   incrementee
37939   D0,EF               BNE 2-17        ;saut oblige
37941   8D,04,94            STA (37892)     ;stocke pos. pixel
37944   60                  RTS
```

Au retour, l'adresse de l'octet écran se trouve dans 3789Ø et 37891, et la position du pixel dans cet octet se trouve dans 37892.

Sauvez ce programme sur cassette.

Avant de poursuivre, il est toujours bon de contrôler un programme. Entrez le Basic suivant :

```
5 CLS
10 INPUT X
20 INPUT Y
30 POKE 37888,X :POKE 37889,Y
40 CALL 37893
```

```
50 PRINT DEEK(37890),PEEK(37892)
5000 STOP
```

En vous référant au *tableau 2* du chapitre IV, essayez différentes entrées :

```
Entrez successivement   Ø,Ø   vous obtenez 4Ø96Ø  Ø
Entrez successivement  11,Ø   vous obtenez 4Ø961  5
Entrez successivement 239,199 vous obtenez 48959  5
```

Exemple 2 : Calculons la **position de la matrice d'un caractère** dont on donne le code ASCII.

Les matrices de caractères dont les codes ASCII vont de 32 à 127 sont stockées, 8 octets par caractère, aux adresses FC78-64632 à FF77-65399 pour l'ATMOS et de FC7Ø-64624 à FF6F-65391 pour l'ORIC 1, de la ROM. Au cours de l'initialisation, cette police de matrices est transférée dans la RAM (MEV) aux adresses B3ØØ-46336 à B7FF-471Ø3. Quand on passe en **HIRES**, cette police est retransférée aux adresses de 99ØØ-39168 à 9BFF-39935. Quand on revient en **TEXT**, cette police est retransférée aux adresses précédentes. (N'oubliez pas de soustraire 32768 de ces adresses pour un ORIC de 16K).

Notez que la police des caractères graphiques n'est pas stockée en ROM, mais celle-ci contient une routine qui les fabrique et les stocke en RAM au cours de l'initialisation. Cette police est également transférée suivant **HIRES** ou **TEXT.**

Nous ferons pointer notre programme sur les matrices qui se trouvent en ROM. De cette façon nous pourrons loger des caractères graphiques, pour notre usage, dans la RAM aux endroits normalement réservés pour les caractères ASCII, accessibles également par notre programme.

```
38016   00              DEF code ASCII
38017   01              DEF no de la police
38018   78,F8           DEF adresse police 1 en ROM
                                (F870 pour ORIC 1)
38020   00,98           DEF adr. police 2: jeu de caracteres
38022   00,9C           DEF adr. police 3: jeu de carac. graph.
38024   00,B4           DEF adr. police 4: jeu de carac. en TEXT
38026   00,00           DEF adresse matrice
38028   00,00           DEF deplacement
38030   A9,00     MAT.  LDA Ø           ;remet a Ø
38032   8D,8D,94        STA (38029)
38035   AD,80,94        LDA (38016)     ;charge le code
38038   8D,8C,94        STA (38028)     ;   caractere
38041   A2,03           LDX 3
```

```
38043   0E,8C,94        ASL (38028)     ;le multiplie
38046   2E,8D,94        ROL (38029)     ;   par 8 en le
38049   CA              DEX             ;   stockant
38050   D0,F7           BNE 2-9
38052   AD,81,94        LDA (38017)     ;no de police
38055   0A              ASL             ;multiplie par 2
38056   AA              TAX
38057   BD,80,94        LDA (38016+X)   ;ajoute le
38060   18              CLC             ;   deplacement
38061   6D,8C,94        ADC (38028)     ;   a l'adresse
38064   8D,8A,94        STA (38026)     ;   de la
38067   BD,81,94        LDA (38017+X)   ;   police
38070   6D,8D,94        ADC (38029)     ;
38073   8D,8B,94        STA (38027)     ;
38076   60              RTS
```

Sauvez ce programme sur cassette.

Avant d'appeler cette routine, on charge 38Ø16 avec le code du caractère à afficher et 38Ø17 avec le n° de police :

1- police des caractères en ROM, utilisable en TEXT et en HIRES
2- police du ler jeu de caractères, utilisable en HIRES
3- police du 2e jeu de caractères graph. utilisable en HIRES
4- police du ler jeu de caractères, utilisable en TEXT.

Au retour, l'adresse du premier octet de la matrice du caractère se trouve en 38Ø26 et 38Ø27.

Comme d'habitude, vérifions ce programme à l'aide du Basic suivant :

```
5 CLS
10 INPUT X
20 INPUT Y
30 POKE 38016,X :POKE 38017,Y
40 CALL 38030
50 FOR A=0 TO 7
60 PRINT PEEK(DEEK(38026)+A);
70 NEXT
5000 STOP
```

Entrons 65 (code de A) puis 1 (lre police), nous obtenons les 8 octets 8 2Ø 34 34 62 34 34 Ø, qui remis en binaire donnent l'arrangement suivant : (page suivante)

```
Ø  Ø  1  Ø  Ø  Ø          8
Ø  1  Ø  1  Ø  Ø         2Ø
1  Ø  Ø  Ø  1  Ø         34
1  Ø  Ø  Ø  1  Ø         34
1  1  1  1  1  Ø         62
1  Ø  Ø  Ø  1  Ø         34
1  Ø  Ø  Ø  1  Ø         34
Ø  Ø  Ø  Ø  Ø  Ø          Ø
```

c'est bien la lettre A. Chaque octet de la matrice a ses bits 7 et 6 à Ø. Les bits 5 à Ø forment le caractère. Le bit Ø est toujours à Ø pour séparer les caractères successifs d'une ligne. Le 8e octet est souvent Ø pour séparer 2 lignes successives. Nous disons souvent car certains caractères descendants comme la virgule, le j minuscule, etc. utilisent ce 8e octet.

Exemple 3 : Réalisons la commande CHAR.

Nous appelons les deux routines des exemples précédents, rechargez donc ces deux routines avant d'entrer le programme suivant :

```
38144-45                    DEF compteurs
38160   20,05,94  CHAR      JSR 37893       ;curseet
38163   20,8E,94            JSR 38030       ;mat.
38166   AD,8A,94            LDA (38026)     ;adresse
38169   85,0C               STA (0C)        ;    matrice
38171   AD,88,94            LDA (38027)     ;
38174   85,0D               STA (0D)        ;
38176   AD,02,94            LDA (37890)     ;adresse
38179   85,0E               STA (0E)        ;    ecran
38181   AD,03,94            LDA (37891)     ;
38184   85,0F               STA (0F)        ;
38186   A9,08               LDA 8
38188   8D,00,95            STA (38144)     ;pointeur 8 lignes
38191   A9,01               LDA 1
38193   8D,01,95            STA (38145)     ;pointeur hauteur
38196   A0,00               LDY 0
38198   B1,0C               LDA (0C),INDY   ;traitement
38200   0A                  ASL             ;
38201   0A                  ASL             ;
38202   85,11               STA (11)        ;
38204   A9,06               LDA 6           ;
38206   38                  SEC             ;
38207   ED,04,94            SBC (37892)     ;    pos. pixel
38210   AA                  TAX
38211   A9,00               LDA 0
38213   18                  CLC
38214   26,11               ROL (11)
38216   2A                  ROL
38217   CA                  DEX
38218   D0,FA               BNE 2-6
```

```
38220   85,10          STA (10)
38222   46,11          LSR (11)
38224   46,11          LSR (11)
38226   B1,0E          LDA (0E),INDY   ;affichage
38228   05,10          ORA (10)        ;
38230   91,0E          STA (0E),INDY   ;
38232   C8             INY             ;
38233   B1,0E          LDA (0E),INDY   ;
38235   05,11          ORA (11)        ;
38237   91,0E          STA (0E),INDY   ;
38239   A5,0E          LDA (0E)        ;calcul
38241   18             CLC             ;   de la
38242   69,28          ADC 40          ;   ligne
38244   85,0E          STA (0E)        ;   suivante
38246   90,02          BCC 2+2
38248   E6,0F          INC (0F)
38250   CE,01,95       DEC (38145)
38253   D0,C5          BNE 2-59
38255   E6,0C          INC (0C)        ;octet
38257   D0,02          BNE 2+2         ;   suivant
38259   E6,0D          INC (0D)        ;   de mat.
38261   CE,00,95       DEC (38144)
38264   D0,B3          BNE 2-77
38266   60             RTS
```

Sauvez ce programme avec les deux précédents.

Le traitement consiste en :

38198 : extraction d'un octet de la matrice.
382ØØ/1 : deux glissements à gauche pour éliminer les bits 7 et 6 non significatifs.
382Ø2 : stockage dans l'octet 11h.
382Ø4 à 3821Ø : l'emplacement 37892 contient la position pixel de l'octet d'affichage, on retranche cette position de 6 et on transfère le résultat dans X, qui contient alors un nombre de 1 à 6.
38211 à 38218 : au moyen de rotations à gauche on transfère X bits de l'octet 11 dans A.
3822Ø : transfert de A dans l'octet 1Ø.
38221/2 : deux glissements à droite de 11 pour remettre Ø aux bits 7 et 6 de 11.

Nous avons ensuite l'affichage par transferts des octets 1Ø et 11, au moyen de l'instruction ORA, dans le fichier d'affichage.

Pour terminer on calcule la ligne suivante, en ajoutant 4Ø, puis l'adresse suivante pour extraire le 2e octet de la matrice.

A l'aide du compteur 38144, on recommence le procédé 8 fois pour afficher les 8 octets de la matrice.

Comprenez bien ce procédé ; munissez-vous au besoin d'une feuille de papier pour effectuer les glissements, les rotations et les **ORA.**

Ce programme ne peut être exécuté qu'en **HIRES** et après avoir entré les paramètres demandés par les programmes des exemples 1 et 2.

Pour visualiser, entrez le Basic suivant :

```
5 HIRES
10 POKE 37889,92
15 POKE 38017,1
20 LET A$="ORIC"
30 FOR A=1 TO LEN(A$)
40 POKE 38016,ASC(MID$(A$,A,1)) :POKE
37888,96+8*A
50 CALL 38160
60 NEXT
5000 STOP
```

Faites RUN.

Comme nous n'avions que 4 instructions à écrire, lignes 38191, 38193, 3825Ø et 38253, pour ajouter une sophistication, nous les avons introduites.

En 38191, LDA 1 signifie qu'on agrandit le caractère 1 fois en hauteur. Pour agrandir 2, 3, 4 fois, faites POKE 38192,3 pour 3 fois en hauteur, puis RUN.

Faites attention aux numéros de colonne et de ligne, car ce programme ne fait aucun test de validité.

Exemple 4 : Réalisons **l'agrandissement d'un caractère** en largeur et en hauteur.

L'agrandissement en hauteur a été vu dans l'exemple précédent, il suffit de boucler sur l'affichage de la ligne avant d'extraire l'octet suivant de la matrice.

L'agrandissement en largeur est plus délicat. Pour le réaliser nous employons le tampon de 8Ø caractères, situé en page Ø, de 35h à 84h (53d à 132d). Nous en donnons l'usage qu'en fait l'ORIC dans le chapitre suivant.

Voici comment nous procédons :

INIT : initialisation qui consiste en appels des routines CURSET et MAT, suivis des transferts d'adresses.

TRAIT : traitement d'une ligne qui consiste à entrer un octet de matrice dans les différents octets d'affichage en tenant compte de la largeur désirée.

LIG.SUI : calcul de l'adresse de la ligne suivante. Bouclage à l'affichage d'une ligne suivant le compteur de hauteur de caractère.

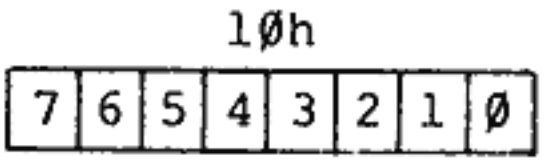
OCT.SUI : calcul de l'adresse de l'octet suivant de la matrice. Bouclage dans l'initialisation jusqu'à ce que les 8 octets aient été traités.

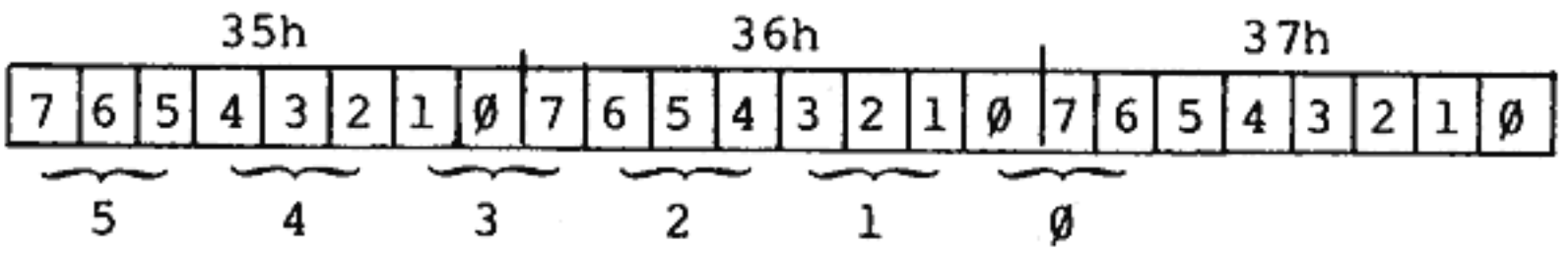
RTS : retour au Basic.

Développons la section **TRAIT** du programme, les autres sections sont identiques à celles de l'exemple précédent.

- PLABIT

Dessinons 3 octets d'adresses 35 36 et 37h. Plus la largeur demandée est grande, plus le nombre d'octets du tampon est grand. Dessinons également l'octet d'adresse 1Øh dans lequel se trouve un octet de la matrice.





Nous extrayons le bit Ø de (1Ø) pour l'introduire dans le bit 7 de (35), en effectuant le nombre de rotations nécessaires des octets 35, 36, 37 etc. eu égard à la largeur. Ces rotations pour faire progresser un bit dans le tampon font l'objet d'une sous-routine appelée SR. On l'appelle toutes les fois qu'on exécute la rotation d'un bloc d'octets.

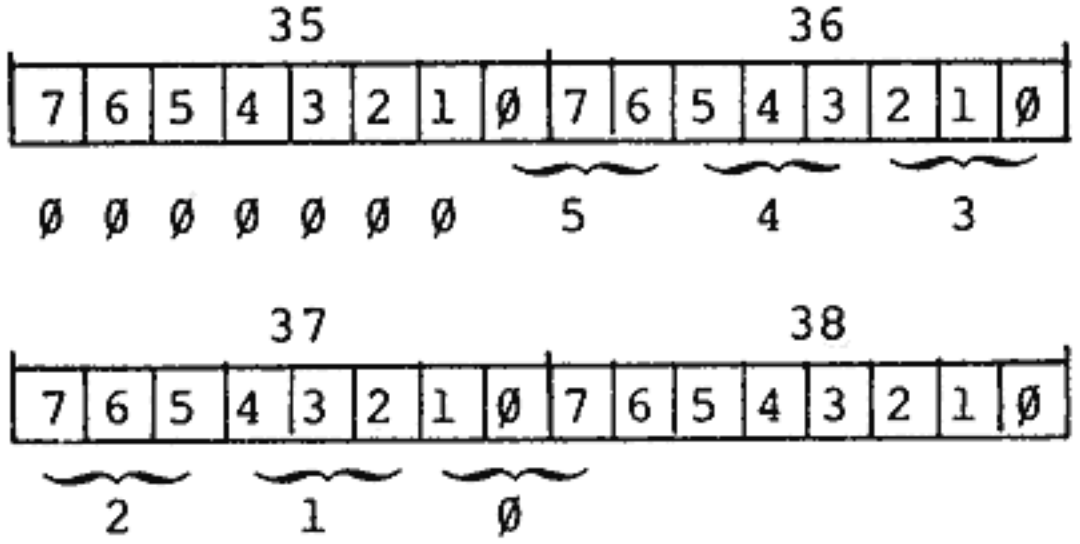
On réintroduit le bit Ø de (1Ø) dans le bit 7 de (35), avec rotation du bloc, autant de fois qu'on agrandit.

Après quoi on passe au bit suivant de (1Ø). Finalement, nous aurons l'aspect de la *figure de la page ci-contre*, pour un agrandissement de trois fois.

La section PLABIT de **TRAIT** réalise ce placement de bits dans le tampon.

• PLAPIX

Il s'agit maintenant de faire reculer ces bits vers la droite. Tout d'abord, tenons compte de la position pixel dans le premier octet. Cette position nous est donnée par le contenu de 37892 de la routine CURSET ; ajoutons 2 à ce nombre pour éviter les bits 7 et 6 de l'octet 35. En supposant que la position pixel est 5, nous effectuons 7 rotations de blocs, d'où la *figure suivante :*

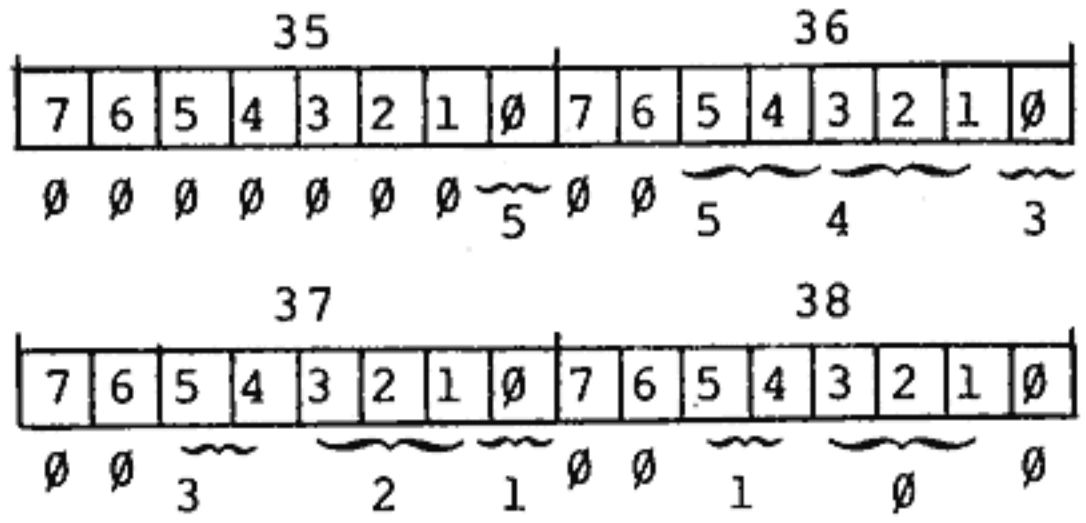


La section PLAPIX réalise ceci

• PLAOC

A présent nous devons effectuer deux rotations pour éviter les bits 7 et 6 des octets suivants, à partir de 36, puis de 37, etc. Pour ces deux rotations, nous employons l'astuce suivante : on met A à Ø, puis l'instruction EOR 1 met A à 1, ce qui provoque une deuxième rotation ; au passage suivant, donc après la deuxième rotation, EOR 1 met A à Ø et on sort de cette boucle.

La section PLAOC réalise ces rotations. On obtient la *figure suivante :*



- AFFI

La section AFFI transfère les octets 35, 36, 37 et 38 dans le fichier d'affichage par un procédé bien connu des lecteurs à présent.

La sous-routine SR

Avant l'appel de cette sous-routine, X doit être chargé à Ø si on veut la rotation à partir de 35, à 1 pour partir de 36, à 2 pour partir de 37, etc... ; Y doit être chargé à un nombre qui correspond au nombre d'octets du bloc qui doivent subir une rotation. Enfin, l'indicateur C doit être à Ø ou à 1 pour introduire Ø ou 1.

Entrez cette sous-routine à 38130 :

```
38130   76,35      SR     ┌►ROR (35+X)
38132   E8                │ INX
38133   88                │ DEY
38134   D0,FA             └ BNE 2-6
38136   60                 RTS
```

Le programme

Entrez ce programme à 38160 :

```
38144   00                 DEF compteur octets de la matrice
38145   00                 DEF LARG: agrandissement largeur
38146   00                 LARG1: compteur
38147   00                 DEF HAUT: agrandissement hauteur
38148   00                 DEF HAUT1: compteur

38160   20,05,94   INIT    JSR 37893       ;routine curset
38163   20,8E,94           JSR 38030       ;routine mat
38166   AD,8A,94           LDA (38026)     ;adresse
38169   85,0C              STA (0C)        ;   mat
38171   AD,8B,94           LDA (38027)     ;
38174   85,0D              STA (0D)        ;
38176   AD,02,94           LDA (37890)     ;adresse
38179   85,0E              STA (0E)        ;   ecran
38181   AD,03,94           LDA (37891)     ;
38184   85,0F              STA (0F)        ;
38186   A9,08              LDA 8           ;compteur octets
38188   8D,00,95           STA (38144)     ;   mat
38191   AD,01,95      ┌──► LDA (38145)     ;compteur
38194   8D,02,95      │    STA (38146)     ;   largeur
38197   AD,03,95      │    LDA (38147)     ;compteur
```

```
38200   8D,04,95            STA (38148)     ;   hauteur
38203   A0,00      TRAIT    LDY 0           ;extrait un
38205   B1,0C               LDA (0C),INDY   ;   octet de la
38207   85,10               STA (10)        ;   matrice
38209   AE,01,95            LDX (38145)     ;(LARG)
38212   E8                  INX
38213   94,34               STY (34+X)      ;RAZ de
38215   CA                  DEX             ;   LARG+1 octets
38216   D0,FB               BNE 2-5         ;   du tampon
38218   A9,01      PLABIT   LDA 1           ;bit 0
38220   A2,00               LDX 0
38222   AC,01,95            LDY (38145)     ;(LARG)
38225   24,10               BIT (10)        ;extraction bit 0
38227   18                  CLC
38228   F0,01               BEQ 2+1
38230   38                  SEC
38231   20,F2,94            JSR 38130       ;rotation bloc
38234   CE,02,95            DEC (38146)     ;(LARG1)
38237   D0,ED               BNE 2-19
38239   8E,02,95            STX (38146)     ;remet LARG1
38242   0A                  ASL             ;bit suivant
38243   C9,40               CMP 64d         ;bit 6
38245   D0,E5               BNE 2-27        ;sort si bit 6
38247   AC,04,94   PLAPIX   LDY (37892)     ;position pixel
38250   C8                  INY             ;   +1
38251   C8                  INY             ;   +1
38252   84,11               STY (11)        ;compteur
38254   A2,00               LDX 0
38256   AC,01,95            LDY (38145)     ;(LARG)
38259   C8                  INY
38260   18                  CLC
38261   20,F2,94            JSR 38130       ;rotation bloc
38264   C6,11               DEC (11)
38266   D0,F2               BNE 2-14        ;sort si (11)=0
38268   E6,11      PLAOC    INC (11)
38270   A9,00               LDA 0           ;pour 2 rotations
38272   A6,11               LDX (11)        ;X=1 au lier passage
38274   AC,02,95            LDY (LARG1)
38277   18                  CLC
38278   20,F2,94            JSR 38130       ;rotation bloc
38281   49,01               EOR 1           ;pour 2 rotations
38283   D0,F3               BNE 2-13
38285   CE,02,95            DEC (LARG1)     ;diminue la longueur
38288   D0,EA               BNE 2-22        ;   du bloc
38290   AC,01,95   AFFI     LDY (LARG)
38293   C8                  INY
38294   B1,0E               LDA (0E),INDY   ;transfert
38296   19,34,00            ORA (34+Y)      ;   pour
38299   91,0E               STA (0E),INDY   ;   affichage
38301   88                  DEY
38302   D0,F6               BNE 2-10
38304   A5,0E      LIGSUI   LDA (0E)        ;calcule
38306   18                  CLC             ;   l'adresse
38307   69,28               ADC 40          ;   ligne
38309   85 0E               STA (0E)        ;   suivante
```

```
38311   90,02              BCC 2+2          ;
38313   E6,0F               INC (0F)        ;
38315   CE,04,95           DEC (38148)      ;(HAUT1)
38318   D0,E2               BNE 2-30
38320   E6,0C      OCTSUI   INC (0C)
38322   D0,02               BNE 2+2
38324   E6,0D               INC (0D)
38326   CE,00,95           DEC (38144)      ;8 fois
38329   F0,03               BEQ 2+3         ;BNE long
38331   4C,2F,95            JMP 38191       ;
38334   60                 RTS
```

Sauvez ce programme ainsi que les routines **CURSET**, **MAT** et **SR** en faisant, par exemple :

```
CSAVE "AGRAN",A37888,E38334
```

A n'employer qu'en **HIRES**

Avant d'appeler cette routine, il faut charger :

```
37888 : abscisse x de Ø à 239  }   suivant
37889 : ordonnée y de Ø à 199  } agrandissement
38Ø16 : code du caractère à agrandir
38Ø17 : n° de la police de caractères
38145 : agrandissement en largeur
38147 : agrandissement en hauteur
```

Voici un Basic d'illustration :

```
5 HIRES
10 POKE 37889,30
20 POKE 38017,1
30 POKE 38145,5 :POKE 38147,5
40 LET A$="ORIC" :B=38 :C=45
50 GOSUB 500
60 POKE 37889,90
70 POKE 38145,2 :POKE 38147,3
80 A$="EDITIONS DU" :B=18 :C=20
90 GOSUB 500
100 POKE 37889,134
110 POKE 38145,10 :POKE 38147,7
120 LET A$="PSI" :LET B=68 :LET C=24
130 GOSUB 500
140 FOR A=42161 TO 43760 STEP 40
150 POKE A,4 :NEXT
160 FOR A=46321 TO 48560 STEP 40
170 POKE A,1 :NEXT
180 POKE 49000,16 :POKE 49040,16 :POKE
    49080,16
190 PRINT CHR$(17) :GET A$
```

```
200 PRINT CHR$(17)
499 STOP
500 FOR A=1 TO LEN(A$)
510 POKE 38016,ASC(MID$(A$,A,1)) :POKE
    37888,B*(A-1)+C
520 CALL 38160 :NEXT
530 RETURN
```

Ce Basic affiche trois textes avec les rapports d'agrandissement 5 x 5, 2 x 3 et 1Ø x 7. Ce Basic met ensuite des couleurs, puis noirçit le reste de l'écran.

Calculez vos coordonnées (x,y) en fonction de l'agrandissement désiré.

DELAI PROGRAMME

L'exemple précédent affiche un caractère agrandi très rapidement. C'est le Basic d'exploitation, pour changer de caractère et de coordonnées, qui donne l'apparence d'une relative lenteur. Aussi, dans des programmes écrits entièrement en C.M., aurons-nous besoin de différents délais programmés.

Chaque instruction du microprocesseur 65Ø2A nécessite un certain nombre de périodes T d'horloge. Comme l'horloge fonctionne à une fréquence de 1 MHz, une période T dure une microseconde. Nous ne donnons les temps d'exécution que de quelques instructions qui nous serviront à programmer des retards.

Certaines instructions indexées durent 1 T de plus quand on change de page, d'où la programmation par page que nous avons adoptée.

Toutes les instructions de branchement durent :

- **3 T** quand il y a branchement dans une même page ;
- **4 T** quand il y a branchement dans une autre page ;
- **2 T** quand il n'y a pas de branchement.

Les retards programmés en C.M. sont équivalents au Basic **WAIT.**

Exemple 5 : **Retard programmé** au moyen d'une boucle.

```
A0,01      DELAI   LDY 1          ;2T
88                 ►DEY           ;2T
D0,FD              └BNE 2-3       ;2T
```

Ce retard dure 2+2+2=6 T soit 6 microsecondes.

```
A0,00      DELAI   LDY 0            ;2T
88                 DEY              ;2T
D0,FD              BNE 2-3          ;3T avec branchement
                                    ;2T sans branchement
```

Ce retard dure 2+255 (2+3) +2+2 = 1281 microsecondes. Selon le chargement de Y, ce délai programmé dure de 6 à 1281 T par tranche de 5 T.

Pour augmenter le retard, on introduit dans la boucle une instruction postiche. Comme presque toutes les instructions influencent l'indicateur Z, il ne reste que les instructions de rangement **STA**, **STX** et **STY** avec emploi d'une adresse de la ROM (MEM), puisque celle-ci ne peut s'altérer.

```
A0,00      DELAI   LDY 0            ;2T
88                 DEY              ;2T
8C,FF,FF           STY (FFFF)       ;4T
D0,FA              BNE 2-6          ;3T ou 2T
```

Ce délai dure 256 fois 4 soit 1Ø24 T de plus que l'exemple précédent donc 23Ø5 microsecondes.

Notez que l'instruction **NOP** dure 2 T.

Exemple 6 : **Retard programmé** au moyen de plusieurs boucles.

```
A2,00      DELAI   LDX 0            ;2T
A0,00              LDY 0            ;2T
88                 DEY              ;2T
D0,FD              BNE 2-3          ;3T ou 2T
CA                 DEX              ;2T
D0,F8              BNE 2-8          ;3T ou 2T
```

Ce délai dure 2 + 255 (2 + 255 (2 + 3) + 2 + 2)
+ 255 (2 + 3) + 2 + 2 = 327936 T
soit 1/3 de seconde.

En chargeant **X** et **Y** avec d'autres valeurs, vous obtenez les délais désirés.

Si vous avez besoin de **X** pour autre chose, vous le remplacez par un octet mémoire inemployé, comme, par exemple, l'octet d'adresse 4ØØh.

```
A0,00      DELAI   LDY 0            ;2T
8C,00,04           STY (400h)       ;4T
```

```
A0,00          LDY 0          ;2T
88             DEY            ;2T
D0,FD          BNE 2-3        ;3T ou 2T
CE,00,04       DEC (400)      ;6T
D0,F6          BNE 2-10       ;3T ou 2T
```

Dans ce sens, vous employez autant d'octets mémoire comme compteurs que vous le désirez. De toute façon vous avez besoin d'au moins un des trois registres principaux A, X ou Y. pour effectuer un **STA**, **STX** ou **STY** en mémoire.

Pendant que l'ordinateur tourne pour générer son délai, il ne fait rien d'autre. Il est donc intéressant de confier cette tâche à un autre circuit spécialisé. L'ORIC possède un tel circuit dans son boîtier, le C.I. 6522. La programmation de ce circuit sort du cadre de cet ouvrage ; les lecteurs intéressés consulteront les notices des constructeurs et, ou, des livres spécialisés. Il vous est loisible de faire effectuer des routines par l'ordinateur pour remplacer ces délais et de stocker les résultats. Ceci, bien entendu, si ces routines calculent des éléments qui vous seront utiles par la suite.

GENERATION DE NOMBRES PSEUDO-ALEATOIRES

Des nombres dûs au hasard sont parfois nécessaires pour programmer certains jeux inter-actifs pour amener de la diversité. Il s'agit de saisir un nombre imprévisible.

La ROM (MEM) contient 16 K d'octets. En extrayant l'un de ces octets, nous obtenons un nombre imprévisible. Il y a des octets qui reviennent cycliquement dans la ROM à certains endroits, limitons-nous à ceux d'adresses qui commencent par D (DØØØh, etc.).

Ainsi, en prenant l'octet d'adresse 4ØØh pour stocker notre pointeur, on ajoute ce pointeur à la page DØ ou Dl ou D2, etc... pour en extraire le contenu.

Exemple 7 : **Nombres pseudo-aléatoires** extraits de la ROM.

```
37888   00                DEF nombre extrait de la ROM
37889   AD,00,04  ALEA     LDA (400h)
```

```
37892   8D,08,94          STA (37896)
37895   AD,00,D0          LDA (ROM)        ;extraction
37898   8D,00,94          STA (37888)      ;stockage
37901   EE,00,04          INC (400h)       ;pour le suivant
37904   60                RTS
```

Ce programme extrait les octets successifs de la page DØ. Si vous voulez la page D1, chargez D1 à l'adresse 37897. Ce programme n'est pas relogeable ; pour l'employer dans vos programmes, changez les adresses absolues 37896 et 37888 selon l'endroit où votre programme est situé.

Pour visualiser, faites :

```
5 CLS
10 REPEAT
20 CALL 37889
30 PRINT PEEK(37888),
40 LET A=A+1
50 UNTIL A=50
5000 STOP
```

RUN vous donne 5Ø nombres ; RUN 1Ø vous affiche les 5Ø suivants.

Essayez d'autres pages que la page DØ.

Timer

Aux adresses 276/277h -63Ø/631d- se trouve un timer de 16 bits décrémenté toutes les centièmes de seconde. Si on a besoin d'un nombre de temps en temps, on saisit son octet bas : à 63Ød. Celui-ci se reproduit évidemment toutes les 2,56 secondes. L'ORIC emploie ce timer, notamment pour l'instruction **WAIT**. Avec cette instruction le timer redémarre à Ø. Faites :

```
5 CLS
10 REPEAT
20 PRINT PEEK(630),DEEK(630)
30 A=A+1
40 UNTIL A=20
```

Vous voyez à la fois l'octet bas et le nombre de 16 bits. Dès qu'on a besoin d'un nombre régulièrement, ce procédé ne convient pas.

Génération de nombres pseudo-aléatoires

Exemple 8 : Ecrivons un **programme pour générer des nombres pseudo-aléatoires** en employant la formule :

$$A_1 = 9. A_\emptyset + 1$$

L'ancien nombre $A_\emptyset$ engendre le nouveau A_1 que l'on stocke.

```
37888   AD,00,04  ALEA    LDA (400h)      ;saisie
37891   0A                ASL             ;fois 2
37892   0A                ASL             ;fois 4
37893   0A                ASL             ;fois 8
37894   18                CLC
37895   6D,00,04          ADC (400)       ;fois 9
37898   8D,00,04          STA (400)       ;stockage
37901   EE,00,04          INC (400)       ; +1
37904   60                RTS
```

Ce programme génère 256 nombres, de Ø à 255, tous différents, dans un ordre pseudo-aléatoire ; après quoi la séquence recommence.

Ce programme est relogeable. Sauvez-le sur cassette.

Notez que cette suite génère des nombres alternativement pair, impair et que son emploi pour certains jeux, comme la simulation d'une roulette par exemple, nécessite un aménagement.

Le nombre pseudo-aléatoire se trouve à l'adresse 4ØØh-1Ø24d. Pour s'introduire dans cette séquence n'importe où, poussez un nombre à l'adresse 1Ø24 au préalable. Faites :

```
5 CLS
10 FOR A=0 TO 255
20 CALL 37888
30 PRINT PEEK(1024);
40 NEXT
50 END
```

L'ANIMATION

Fonction KEY$ et instruction GET A$

Dans des programmes où le déroulement s'oriente suivant la touche pressée, on emploie la fonction **KEY$.**

En C.M., nous appelons une routine de la ROM située à EB78 pour l'ATMOS et à E9Ø5 pour l'ORIC 1.

```
20,78,EB  KEY$    JSR EB78        ;pour l'ATMOS
20,05,E9          JSR E905        ;pour l'ORIC 1
```

Cette routine, appelée **GTORKB** par Tangerine, retourne dans le registre A, le code de la touche pressée, et met à 1 l'indicateur N si la touche pressée est valable. Ceci nous donne le moyen de réaliser l'instruction **GET**, donc d'attendre une touche avant de poursuivre.

```
20,78,EB  GET   ┌►JSR EB78        ;E905 pour l'ORIC 1
10,FB           └─ BPL 2-5
```

Exemple 9 : **Codes retournés par le clavier.**

Entrez le programme suivant :

```
1026    20,78,EB        ┌►JSR EB78        ;E905 pour l'ORIC 1
1029    10,FB           └─ BPL 2-5
1031    8D,00,04          STA (1024d)
1034    60                RTS
```

Entrez ensuite le Basic :

```
5 CLS
10 PRINT "CODE",,"INDICATEUR"
20 CALL 1026
30 PRINT PEEK(1024),,PEEK(#209)
40 WAIT 100
50 GOTO 20
```

Faites **RUN** puis, en poussant sur une touche, vous avez le code ASCII retourné par le clavier dans le registre A qui a été stocké en 1Ø24. Nous affichons également le contenu de 2Ø9h sous la rubrique indicateur.

Quatre touches ne donnent aucun retour de code : **CTRL**, les deux **SHIFT** et **FUNCT**, cette dernière pour l'ATMOS uniquement. Ces quatre touches doivent être enfoncées et, en même temps, une autre touche doit l'être également. A ce moment, la colonne INDICATEUR qui donne toujours 56 pour l'enfoncement d'une seule touche, donne à présent :

162 pour la touche CTRL
164 pour la touche SHIFT de gauche
167 pour la touche SHIFT de droite
165 pour la touche FUNCT

Enfoncez	Au retour, vous avez	
	CODE	INDICATEUR
A	65	56
B	66	56
=	61	56
SHIFT gauche et =	43	164
SHIFT droite et =	43	167
CTRL et A	1	162
FUNCT et A	65	165
etc.		

Pour sortir du programme, laissez enfoncées CTRL et C. C'est à cela que sert la ligne **WAIT.**

Exemple 10 : **Caractères définis** par l'usager.
Créons 4 caractères graphiques nouveaux et plaçons-les dans le jeu de caractères normaux à la place des minuscules a, b, c, et d.

```
37905   0E,11,22,24,22,11,0E,00          ;DEF carac. graphique
37913   0C,1E,3F,1E,0C,1E,3F,00          ;DEF   idem
37921   00,21,1E,12,12,1E,21,00          ;DEF   idem
37929   20,38,3E,3F,3E,38,20,00          ;DEF   idem

37937   A2,20      GRAPH   LDX 32d
37939   BD,10,94           ┌►LDA (37904+X)
37942   9D,07,B7           │ STA (46855+X)
37945   CA                 │ DEX
37946   D0,F7              └─ BNE 2-9
37948   60                   RTS
```

Les quatre caractères de 8 octets chacun rangés de 37905 à 37936, sont chargés aux adresses 46856 à 46887. L'adresse 46856 est le départ de la matrice du a minuscule.

Sauvez ce programme, en même temps que celui de *l'exemple 8*, avec la commande :

```
CSAVE "JEU",A37888,E37948
```

Pour les afficher au hasard sur l'écran, nous reprenons le programme de *l'exemple 3* du chapitre III, en changeant les adresses car ce programme n'est pas relogeable tel quel.

Nous reprenons ce programme pour rester compatible avec l'ORIC 1. Celui-ci, contrairement à l'ATMOS, ne possède pas la facilité d'afficher n'importe où sur l'écran à l'aide de l'instruction **PRINT.**

Entrez le programme suivant :

```
37950   00,00                DEF pos. d'affichage
37952   00                   DEF colonne: x de 2 a 39
37953   00                   DEF ligne : y de 1 a 27
37954   A9,BB      POSAFF    LDA BB
37956   8D,3F,94             STA (37951)
37959   18                   CLC
37960   A9,80                LDA 80h
37962   6D,40,94             ADC (37952)
37965   AE,41,94             LDX (37953)
37968   F0,0B                BEQ 2+11
37970   69,28                ADC 40
37972   90,04                BCC 2+4
37974   EE,3F,94             INC (37951)
37977   18                   CLC
37978   CA                   DEX
37979   D0,F5                BNE 2-11
37981   8D,3E,94             STA (37950)
37984   60                   RTS
```

Ce programme a été commenté au chapitre III.

Sauvez ce programme avec les deux précédents

```
CSAVE "JEU",A37888,E37984
```

Pour ne pas faire apparaître les caractères graphiques que nous avons définis, d'une manière monotone, nous écrivons un 2e programme, **ALEA2**, de génération de nombres aléatoires. En effet, comme nous le disions dans *l'exemple 8* ci-dessus, ce programme ne convient pas pour certains jeux et notamment quand nous avons besoin d'un couple (x,y) comme coordonnées d'affichage. Ce 2e programme, ALEA2, est construit selon la formule :

$$A_1 = 13 \cdot A_0 + 1$$

De cette façon, vous avez deux générateurs de nombres aléatoires qui vous permettent de créer des nombres aléatoires modulo 65536, par exemple, en utilisant l'un pour créer l'octet bas et l'autre pour l'octet haut.

Entrez le programme suivant :

```
37990   AD,01,04   ALEA2   LDA (401h)
37993   0A                 ASL              ;fois 2
37994   18                 CLC
37995   6D,01,04           ADC (401)        ;fois 3
37998   0A                 ASL              ;fois 6
37999   0A                 ASL              ;fois 12
38000   18                 CLC
38001   6D,01,04           ADC (401)        ;fois 13
```

→

```
38004   8D,01,04            STA (401)
38007   EE,01,04            INC (401)       ; +1
38010   60                  RTS
```

Sauvez ce programme avec les précédents :

```
CSAVE "JEU",A37888,E38Ø1Ø
```

Pour terminer, écrivons un programme pour créer un nombre x, compris entre 2 et 38, et un nombre y, compris entre 1 et 25. Nous entrons ce couple (x,y) dans le programme qui calcule la position d'affichage.

Ce programme illustre également la façon de traiter un nombre aléatoire de Ø à 255 pour le réduire dans une fourchette.

```
38015   20,00,94  XY        JSR 37888       ;ALEA
38018   AD,00,04            LDA (400)       ;saisie d'un nombre
38021   C9,25               CMP 37          ;le reduit
38023   90,05               BCC 2+5         ;   entre
38025   E9,25               SBC 37d         ;   0 et 36
38027   4C,85,94            JMP 38021       ;
38030   AA                  TAX
38031   E8                  INX             ;entre 1 et 37
38032   E8                  INX             ;entre 2 et 38
38033   8E,40,94            STX (37952)     ;le stocke
38036   20,66,94            JSR 37990       ;ALEA2
38039   AD.01.04            LDA (401)       :saisie d'un nombre
38042   C9,19               CMP 25          ;le reduit
38044   90,05               BCC 2+5         ;   de
38046   E9,19               SBC 25          ;   0 a 24
38048   4C,9A,94            JMP 38042       ;
38051   A8                  TAY
38052   C8                  INY             ;   de 1 a 25
38053   8C,41,94            STY (37953)     ;le stocke
38056   60                  RTS
```

Sauvez le tout CSAVE "JEU",A37888,E38Ø56

Entrez le Basic suivant :

```
5 CLS
10 CALL 37937
20 PAPER 0 :INK 1
30 D=48041
40 FOR C=1 TO 7
50 FOR E=1 TO 3
60 D=D+40
70 POKE D,C :NEXT
80 NEXT
90 REPEAT
100 FOR B=97 TO 100
110 CALL 38015 :CALL 37954
```

```
120 POKE DEEK(37950),B
130 NEXT
140 A=A+1
150 UNTIL A=25
160 GET A$
5000 STOP
```

Ce programme affecte sept couleurs encre aux lignes de l'écran puis, dans la boucle **REPEAT**, affiche au hasard les quatre caractères graphiques et cela, 25 fois de suite. Au préalable, la ligne 1Ø transfère les quatre caractères graphiques qui seront affichés avec les codes 97, 98, 99 et 1ØØ.

Exemple 11 : **Jeu d'évitement.**

Nous plaçons au centre de la grille obtenue dans l'exemple précédent, un curseur, représenté par le signe + inversé. A ce moment, observez bien l'écran pour déterminer le chemin que va emprunter votre curseur pour faire le plus grand nombre de pas possibles tout en évitant les obstacles. Vous faites progresser votre curseur à l'aide des quatre touches fléchées. Dès que le jeu est démarré, vous ne pouvez plus l'arrêter sauf si votre curseur bute sur un caractère graphique, sur un des bords de l'écran ou sur la trace de son passage qu'il ne peut franchir.

Pour pouvoir afficher n'importe où, n'importe quoi, écrivons un programme d'affichage inspiré de celui du chapitre III. Rechargez les routines de l'exemple précédent et ajoutez :

```
38060   20,42,94  AFF     JSR 37954       ;pos. affichage
38063   AD,3E,94          LDA (37950)
38066   8D,C5,94          STA (38085)
38069   AD,3F,94          LDA (37951)
38072   8D,C6,94          STA (38086)
38075   A0,00             LDY 0
38077   B9,00,00          LDA (adr+Y)     ;adr. texte
38080   F0,09             BEQ 2+9
38082   09,80             ORA 128         ;pour inverser
38084   99,00,00          STA (adr+Y)     ;adr. affichage
38087   C8                INY
38088   4C,BD,94          JMP 38077
38091   60                RTS
38092   7E,00                             DEF carac.graph.,fin
38094   07,50,4F,49,4E,54,53,             DEF encre, POINTS
        3A,30,30,30,00                    DEF :000, fin
38106   2B,00                             DEF +, fin
```

Sauvez le tout CSAVE "JEU",A37888,E381Ø7

Cette routine appelle celle qui calcule la position d'affichage qui utilise celle-ci pour afficher un texte.

Nous disposons ainsi des routines suivantes :

37888 **ALEA** , au retour 4ØØh contient un nombre
37937 **GRAPH** , à l'entrée il faut d'abord charger les octets 379Ø5 à 37936 pour définir 4 objets à votre goût, ou laisser ceux qui y sont.
37954 **POS.AFF**, à l'entrée chargez 37952 avec une valeur de x entre 2 et 39 et 37953 avec une valeur de y entre 1 et 27. Au retour 3795Ø/51 contiennent la position d'affichage.
3799Ø **ALEA2** , au retour 4Ø1h contient un nombre
38Ø15 **XY** , place un couple (x,y) au hasard, comme entrées de la routine POS.AFF
38Ø6Ø **AFF** , à l'entrée chargez x et y comme pour POS.AFF et 38Ø78/79 avec l'adresse du texte à afficher. Nous en utilisons trois pour notre jeu, leurs adresses sont 38Ø02, 38Ø94 et 381Ø6.

Plan du programme

- Dessins des contours. Nous entourons l'écran de caractères graphiques de code 126 pour que ce soit un obstacle au curseur et aussi pour enjoliver la présentation. Nous aurions pu faire des tests sur les bords de l'écran. De plus, ceci montre une façon de faire un contour.
- Affichage d'un texte. Initialise le compteur de pas à ØØØ.
- Affichage du curseur au centre de l'écran.
- Boucle d'attente qui attend votre 1re intervention.
- Délai et saisie. Nous avons besoin d'un délai pour limiter la vitesse du jeu. Nous incorporons dans ce délai la saisie de l'une des quatre touches fléchées que vous avez choisi. Cette saisie est réalisée en consultant la variable système située à 2DF qui contient le code de la dernière touche frappée, augmentée de 128. Cette variable système est utilisée également pour la boucle d'attente.
- Déplacement du curseur suivant la touche enfoncée.
- Test de collision entre le curseur et un obstacle, qui boucle à la rubrique délai ou qui arrête le jeu si un obstacle est rencontré.
- Manipulation du nombre de pas réussis sans rencontrer d'obstacles.

Programme du jeu

```
38160   A9,FE      CONT.H  LDA 254          ;code 126 inverse
38162   A2,26              LDX 38           ;38 colonnes
38164   9D,A9,BB       ->  STA (48041+X)    ;dessus
38167   9D,B9,BF           STA (49081+X)    ;dessous
38170   CA                 DEX
38171   D0,F7              BNE 2-9
38173   A9,CC      CONT.V  LDA CC           ;adresse texte
38175   A2,94              LDX 94           ;   38092
38177   8D,BE,94           STA (38078
38180   8E,BF,94           STX (38079
38183   A0,1B              LDY 27           ;compteur et ligne
38185   8C,41,94       ->  STY (37953)
38188   A2,02              LDX 2            ;gauche
38190   8E,40,94           STX (37952)
38193   98                 TYA
38194   48                 PHA              ;sauve Y
38195   20,AC,94           JSR 38060        ;affichage
38198   A2,27              LDX 39           ;droite
38200   8E,40,94           STX (37952)
38203   20,AC,94           JSR 38060        ;aff.
38206   68                 PLA
38207   A8                 TAY
38208   88                 DEY              ;dec. le Y recupere
38209   D0,E6              BNE 2-26
38211   A2,1D      SCORE   LDX 29           ;colonne
38213   8E,40,94           STX (37952)
38216   A0,1B              LDY 27           ;ligne
38218   8C,41,94           STY (37953)
38221   A9,CE              LDA CE           ;adr.texte
38223   8D,BE,94           STA (38078)      ;   38094
38226   20,AC,94           JSR 38060        ;affichage
38229   A2,12      CUR.    LDX 18           ;colonne
38231   A0,0E              LDY 14           ;ligne
38233   8E,40,94           STX (37952)
38236   8C,41,94           STY (37953)
38239   A9,DA              LDA DA           ;adr. texte
38241   8D,BE,94           STA (38078)      ;   38106
38244   20,AC,94           JSR 38060)       ;affichage curseur
38247   AD,DF,02   ATT.  ->LDA (2DF)        ;boucle
38250   F0,FB              BEQ 2-5          ;   d'attente
38252   A2,80      DELAI ->LDX 128          ;vitesse du jeu
38254   A0,00            ->LDY 0
38256   AD,DF,02         ->LDA (2DF)        ;saisie touche
38259   88                 DEY
38260   D0,FA              BNE 2-6
38262   CA                 DEX
38263   D0,F5              BNE 2-11
38265   C9,88      TOUCHE  CMP 136          ;   8+128
38267   D0,03              BNE 2+3
38269   CE,40,94           DEC (37952)
38272   C9,89            ->CMP 137          ;   9+128
38274   D0,03              BNE 2+3
38276   EE,40,94           INC (37952)
38279   C9,8A            ->CMP 138          ;   10+128
```

→

```
38281   D0,03            BNE 2+3
38283   EE,41,94         INC (37953)
38286   C9,8B            CMP 139          ;   11+128
38288   D0,03            BNE 2+3
38290   CE,41,94         DEC (37953)
38293   20,42,94  TEST   JSR 37954        ;pos. affichage
38296   AD,3E,94         LDA (37950)
38299   8D,A5,95         STA (38309)
38302   AD,3F,94         LDA (37951)
38305   8D,A6,95         STA (38310)
38308   AD,00,00         LDA (POS.AFF.)   ;test de
38311   C9,20            CMP 32           ;   collision
38313   D0,29            BNE 2+41
38315   20,AF,94         JSR 38063        ;aff. le curseur
38318   AE,DF,BF  MANIP  LDX (49119)      ;unites
38321   E0,B9            CPX 185          ;57 + 128
38323   F0,06            BEQ 2+6
38325   EE,DF,BF         INC (49119)
38328   4C,6C,95         JMP 38252        ;saut a DELAI
38331   A2,B0            LDX 176          ;48 + 128
38333   8E,DF,BF         STX (49119)
38336   EE,DE,BF         INC (49118)      ;dizaines
38339   AE,DE,BF         LDX (49118)
38342   E0,BA            CPX 186          ;58 + 128
38344   D0,EE            BNE 2-18
38346   A2,B0            LDX 176          ;48 + 128
38348   8E,DE,BF         STX (49118)
38351   EE,DD,BF         INC (49117)      ;centaines
38354   D0,E4            BNE 2-28         ;saut oblige
38356   60               RTS
```

Si vous avez bien suivi ce chapitre, vous n'aurez aucune difficulté de compréhension.

Le Basic d'exploitation est le même que celui de *l'exemple 1Ø* auquel vous ajoutez ces lignes :

```
5 CLS :HIMEM 37887
15 PRINT CHR$(17)
155 CALL 38160
165 PRINT CHR$(17)
```

Si vous désirez moins d'obstacles sur l'écran, modifiez la ligne 15Ø en conséquence.

Si vous voulez que le curseur se déplace plus vite, introduisez une valeur plus petite dans X à la ligne 38252. Faites, par exemple POKE 38253,5Ø. Si vous voulez un déplacement plus lent, augmentez cette valeur. Pour X=Ø c'est très lent ; pour X=1 c'est très rapide, au point de ne pas pouvoir manoeuvrer.

En agissant sur ces deux paramètres : vitesse et

nombre d'obstacles, vous augmenterez votre plaisir de jouer à ce jeu d'évitement.

Faites des compétitions dans votre entourage.

Bon amusement.

CHAPITRE 6
LA MÉMOIRE MORTE

La MEM (ROM) contient le moniteur et l'interpréteur Basic qui fait appel à une multitude de routines d'exploitation. C'est le coeur de votre ORIC. Son écriture est le fait de programmeurs de haut niveau et son étude est passionnante. Nous ne démontons que quelques routines, car ici, la dualité règne. En effet, d'un ORIC ATMOS à un ORIC 1, les points d'entrée sont différents, le nombre de routines également et leurs contenus varient fréquemment.

Nous donnons ensuite les points d'entrée des routines d'exploitation des instructions et des fonctions.

Pour terminer, nous expliquons les différentes manières d'exploiter les codes machine.

Ce livre est une initiation à la programmation en codes machine. Les programmes présentés sont exemplatifs, de difficultés croissantes et ne sont pas optimisés. La teneur des chapitres précédents aurait pu être réduite pour donner plus de place à ce chapitre. D'aucun aurait préféré plus de programmes de jeux ou de traitement de texte, etc. Le lecteur curieux pourra continuer l'exploration de la MEM grâce aux explications données ici.

QUELQUES ROUTINES

La routine GTORKB

La routine **GTORKB**, ainsi appelée par Tangerine, est simple. Au retour A contient le code ASCII de la touche frappée et l'indicateur de signe, le bit 7 du registre P, est mis à 1 si la touche frappée est validée.

C'est la même pour les deux ORIC, seul le point d'entrée change. Le point d'entrée est EB78 pour l'ATMOS et E9Ø5 pour l'ORIC 1.

```
60280   AD,DF,02        LDA (2DF)       ;saisie
60283   10,0B           BPL 2+11
60285   08              PHP             ;sauve P (bit 7)
60286   29,7F           AND 7F          ;elimine le bit 7
60288   48              PHA             ;sauve A
60289   A9,00           LDA Ø
60291   8D,DF,02        STA (2DF)       ;RAZ de (2DF)
60294   68              PLA             ;restaure A
60295   28              PLP             ;restaure P
60296   60              RTS
```

Le clavier est scruté régulièrement et le code du caractère frappé +128 est mis dans la var. sys. 2DF. Si le caractère n'est pas valide, 2DF contient Ø. Ceci explique la boucle d'attente de *l'exemple 11* du chapitre V.

Supposons que 2DF contienne le code de la lettre A, soit 65+128 ou 193. LDA met N à 1 et le registre P est sauvé puis rétabli avant le retour. And élimine le bit 7 de A, d'où A contient 65, également sauvé puis rétabli avant le retour. Enfin on remet 2DF à Ø.

La routine E2

Examinons une petite routine, très fréquemment employée par l'ordinateur pour pointer sur une adresse qui est soit celle du tampon d'entrée, soit celle de la zone du programme Basic, etc. pour en extraire le caractère à interpréter.

Elle va de E2 à F2 pour les deux ORIC. Le contenu est légèrement différent. Son transfert de la ROM en page Ø se fait au cours de l'initialisation, à la mise sous tension.

Pour l'ATMOS, le transfert des 17 octets se fait à partir de EC9C - 60572 et pour l'ORIC 1 à partir de EA24 - 59940.

Voyons la routine de l'ATMOS :

E2

```
226     E6,E9          INC (E9)          ;l'adresse
228     D0,02          BNE 2+2           ;   est
230     E6,EA          INC (EA)          ;   incrementee
232     AD,60,EA       LDA (adresse)     ;extrait le code
235     C9,20          CMP 32            ;code 'espace'
237     F0,F3          BEQ 2-13
239     20,B9,EC       JSR ECB9
242     60             RTS
```

Pour l'ORIC 1, la ligne 239 change :

```
239     20,41,EA       JSR EA41
```

L'adresse dont on extrait le code pour le mettre dans A, ligne 232, est chargée avant l'appel de cette routine suivant l'endroit à interpréter. L'adresse renseignée ici, EA60, est celle qui est transférée de la ROM. Nous reviendrons sur cette routine en expliquant la boucle d'attente de l'interpréteur.

La routine E2 incrémente l'adresse d'où on extrait un code ; si ce code est 32d, on l'ignore et on recommence. La routine ECB9 pour l'ATMOS, EA41 pour l'ORIC 1, est ensuite appelée. Ces 2 routines sont différentes. Elles mettent l'indicateur Z à 1 si A contient le code 39, apostrophe, ou le code 200, code de la fonction **ELSE** ; dans les autres cas Z=0. Elles mettent également l'indicateur C à 0 si A contient le code d'un chiffre et à 1 dans les autres cas. Cette dernière partie est commune aux deux ORIC. Examinons-la, car elle est astucieuse et instructive :

ATMOS

```
60609   C9,3A          CMP 58            ;code de :
60611   B0,06          BCS 2+6
60613   38             SEC
60614   E9,30          SBC 48            ;code de 0
60616   38             SEC
60617   E9,D0          SBC 208
60619   60             RTS
```

Si le code est 58 ou plus, CMP met C à 1, puis RTS. Si le code est plus petit que 58, remarquons tout d'abord que retrancher 48 puis 208 soit 256 de A **ne change pas** A.

Deux cas se présentent :

1er cas : A contient le code d'un chiffre, code 48 à 57, et SBC 48 laisse un nombre positif dans A, Ø à 9, et SBC 2Ø8 met C à Ø.

2e cas : A contient un code de Ø à 47 et SBC 48 laisse un nombre négatif dans A, 2Ø8 à 255 et SBC 2Ø8 met C à 1. Ceci est important pour l'interpréteur, car à la sortie de la routine E2, l'indicateur C oriente vers l'interprétation d'une commande ou d'une ligne Basic.

Boucle d'attente

Cette boucle est située à C4B7 - 5Ø359 pour l'ATMOS et à C4C7 - 5Ø375 pour l'ORIC 1, avec des différences.

```
50359   20,2F,C8      ┌► JSR C82F
50362   20,92,C5      │  JSR C592
50365   86,E9         │  STX (E9)        ;met l'adresse pour
50367   84,EA         │  STY (EA)        ;   la routine E2
50369   20,E2,00      │  JSR E2
50372   AA            │  TAX
50373   F0,F0         └─ BEQ 2-16        ;si A=0
```

Les deux premières lignes sont différentes pour l'ORIC 1 :

```
4E,F1,Ø2        LSR (2F1)
20,A2,C5        JSR C5A2
```

Dans ce cas-ci, on n'a pas besoin de l'indicateur Z retourné par la routine E2 ; pour l'éviter on fait **TAX**, de cette façon A=Ø provoque le bouclage. Quand l'interprèteur a entré une ligne Basic, ou exécuté une commande, il revient dans cette boucle. Voyons les deux premières lignes.

La routine C82F constate si on veut afficher, (2F1) =Ø, ou imprimer, (2F1)=128 (bit 7). Nous n'entrerons pas dans les détails.

La routine C592, C5A2 pour l'ORIC 1, est très importante. Nous ne la démontons pas ici. Voici ce qu'elle fait :

D'emblée elle appelle une routine qui est la véritable boucle d'attente, que nous avons déjà vue

```
JSR EB78                    ;E9Ø5 pour l'ORIC 1
BPL 2-5
```

ensuite, dès qu'un caractère est saisi, il est affiché et introduit dans le tampon d'entrée qui peut contenir 8Ø caractères. Ce tampon est situé en page Ø, de 35 à 84h - 53 à 132d. Quand il y a 76, 77 ou 78 caractères introduits, un PING retentit pour attirer l'attention. Au 79e caractère, la routine se replace au début du tampon avec affichage d'un \, la ligne est annulée.

Dès qu'un code **RETURN** est frappé, la routine pousse un Ø sur le tampon puis retourne à la routine qui a appelé avec Y=Ø et X=34h. Donc dans la boucle d'attente, on pousse l'adresse moins un du tampon d'entrée avant d'exécuter E2.

En sortant de la boucle d'attente pour exécuter ce qu'il y a dans le tampon d'entrée, l'interprèteur exécute la routine C5FA - 5Ø682 (C6ØA pour l'ORIC 1).

La routine C5FA

Cette routine extrait les caractères successifs du tampon d'entrée. Elle compare une suite de lettres avec une table de mots clé. Dès qu'il y a coïncidence, elle remplace les lettres du mot clé par son code et replace celui-ci dans le tampon d'entrée.

Nous avons ainsi connaissance d'une table qui commence à CØEA - 49386 et finit à C2A6 - 4983Ø. La dernière lettre de chaque mot clé est représentée par son code augmenté de 128. Par exemple :

```
49386 contient 45 code de E
49387 contient 4E code de N
49388 contient C4 code de D + 128
```

et ainsi de suite jusqu'à 4983Ø.

Pour l'ORIC 1 la table commence à CØEA - 49386 et se termine à C2AA - 49384.

Code	*Mot clé*	*Code*	*Mot clé*
128	END	132	TRON
129	EDIT	133	TROFF
13Ø	STORE (pas pour ORIC 1)	134	POP
131	RECALL (pas pour ORIC 1)	135	PLOT

Code	*Mot clé*
136	PULL
137	LORES
138	DOKE
139	REPEAT
14Ø	UNTIL
141	FOR
142	LLIST
143	LPRINT
144	NEXT
145	DATA
146	INPUT
147	DIM
148	CLS
149	READ
15Ø	LET
151	GOTO
152	RUN
153	IF
154	RESTORE
155	GOSUB
156	RETURN
157	REM
158	HIMEM
159	GRAB
16Ø	RELEASE
161	TEXT
162	HIRES
163	SHOOT
164	EXPLODE
165	ZAP
166	PING
167	SOUND
168	MUSIC
169	PLAY
17Ø	CURSET
171	CURMOV
172	DRAW
173	CIRCLE
174	PATTERN
175	FILL
176	CHAR
177	PAPER
178	INK
179	STOP
18Ø	ON
181	WAIT
182	CLOAD
183	CSAVE
184	DEF
185	POKE
186	PRINT
187	CONT
188	LIST
189	CLEAR
19Ø	GET
191	CALL
192	!
193	NEW
194	TAB(
195	TO
196	FN
197	SPC(
198	@ pas ORIC 1
199	AUTO
2ØØ	ELSE
2Ø1	THEN
2Ø2	NOT
2Ø3	STEP
2Ø4	+
2Ø5	-
2Ø6	*
2Ø7	/
2Ø8	↑
2Ø9	AND
21Ø	OR
211	>
212	=
213	<
214	SGN
215	INT
216	ABS
217	USR
218	FRE
219	POS
22Ø	HEX$
221	&
222	SQR
223	RND
224	LN
225	EXP
226	COS

Code	*Mot clé*	*Code*	*Mot clé*
227	SIN	237	CHR$
228	TAN	238	PI
229	ATN	239	TRUE
23Ø	PEEK	24Ø	FALSE
231	DEEK	241	KEY$
232	LOG	242	SCRN
233	LEN	243	POINT
234	STR$	244	LEFT$
235	VAL	245	RIGHT$
236	ASC	246	MID$

Table des codes des mots clé du Basic

Nous avons placé une ligne après le code 193 NEW, pour séparer les instructions des fonctions.

Dans l'ORIC 1, les codes 13Ø et 131 n'existent pas. Ils correspondent aux mots INVERSE et NORMAL. De plus, le code 247 correspond au mot GO. La fonction @ est indisponible également.

La routine C915

L'interprèteur consulte à nouveau le tampon d'entrée remanié par la routine précédente en appelant la routine E2. A contient maintenant le code d'un caractère ou d'un mot clé.

L'interprèteur appelle ensuite la routine C915 pour l'ATMOS, C8FE pour l'ORIC 1.

Démontons cette importante routine qui va nous permettre de trouver les adresses des 66 instructions : de **END** à **NEW**, code 128 à 193.

C915

```
51477   F0,49        BEQ 2+73          ;51552
51479   E9,80        SBC 128           ;A=A-128
51481   90,11        BCC 2+17          ;si A < 128
51483   C9,42        CMP 66
51485   B0,30        BCS 2+48          ;51535 si A > 65
51487   CA           ASL               ;A=2 fois A
51488   A8           TAY               ;Y=A
51489   B9,07,C0     LDA (C007+Y)      ;pour
51492   48           PHA               ;   le RTS
51493   B9,06,C0     LDA (C006+Y)      ;   de E2
51496   48           PHA               ;   qui suit
51497   4C,E2,00     JMP E2
51500   4C,1C,CB     JMP CB1C          ;51996
```

Pour l'ORIC 1, les sauts sont différents :

```
51454   F0,2D           BEQ 2+45        ;51501
  ''
51462   B0,14           BCS 2+20        ;51484
  ''
51477   4C,D2,CA        JMP CAD2        ;51922
```

Revenons à l'ATMOS.

A l'entrée A contient un code.

Si le code est Ø, aller à 51552 (515Ø1 pour ORIC 1).

Si le code est plus petit que 128, donc une lettre, un chiffre, un signe, aller à 51996 (51922 pour ORIC 1).

Si le code est une fonction, de TAB(à MID$, aller à 51535 (51484 pour ORIC 1).

Si le code est une instruction, Y a pour valeur Ø à 13Ø et uniquement une valeur paire, pour les instructions de code 128 à 193. En effet, après avoir retranché 128 de A, A contient un nombre de Ø à 65, que l'on multiplie par deux puis qu'on transfère dans Y. Les adresses des routines des instructions sont situées de CØØ6h+Ø à CØØ6h+13Ød.

Remarquons un procédé important. On pousse l'octet haut dans la pile, puis on y pousse l'octet bas. Ensuite, on fait JMP E2 et **non** JSR E2. L'adresse de retour a été modifiée ; quand apparaît le RTS de la routine E2, on saute à la routine de l'instruction. Revoyez ce que nous disions de RTS au début du chapitre IV. Après avoir récupéré l'adresse, RTS ajoute 1 à cette adresse. Donc l'adresse qui se trouve à CØØ6 - 49158 et seq. est l'adresse de la routine **moins** 1.

POINTS D'ENTREE DE QUELQUES ROUTINES

Le **tableau 1** ci-contre donne l'adresse réelle des instructions, donc celle que l'on trouve dans la ROM de CØØ6h-49158d à CØ88h - 49288d augmentée de 1. La table de la ROM pour l'ORIC 1 est située au même endroit mais les adresses qu'elle donne sont évidemment différentes.

Mot clé	*ATMOS*	*ORIC 1*	*Mot clé*	*ATMOS*	*ORIC 1*
END	C973	C941	TEXT	EC21	E9A9
EDIT	C692	C6A5	HIRES	EC33	E9BB
STORE	E987		SHOOT	FAB5	F415
RECALL	E9D1		EXPLODE	FACB	F418
TRON	CD16	CC8C	ZAP	FAE1	F41B
TROFF	CD19	CC8F	PING	FA9F	F412
POP	CA12	C9EØ	SOUND	EAFC	E889
PLOT	DA51	D9C6	MUSIC	EAFC	E889
PULL	DAA1	DA16	PLAY	EAFC	E889
LORES	D9DE	D937	CURSET	EAFØ	E87D
DOKE	D967	D8AC	CURMOV	EAFØ	E87D
REPEAT	DA85	D9FA	DRAW	EAFØ	E87D
UNTIL	DAA1	DA16	CIRCLE	EAFØ	E87D
FOR	C855	C841	PATTERN	EAFØ	E87D
LLIST	C7FD	C824	FILL	EAFØ	E87D
LPRINT	C8Ø9	C832	CHAR	EAFØ	E87D
NEXT	CE98	CEØC	PAPER	EAFC	E889
DATA	CA3C	CAØA	INK	EAFC	E889
INPUT	CD55	CCC9	STOP	C971	C93F
DIM	D17E	DØF2	ON	CAC2	CA78
CLS	CCCE	CCØA	WAIT	D958	D89D
READ	CD89	CCFD	CLOAD	E85B	E7AA
LET	CB1C	CAD2	CSAVE	E9Ø9	E7DB
GOTO	C9E5	C9B3	DEF	D4BA	D4Ø1
RUN	C9BD	C98B	POKE	D94F	D894
IF	CA7Ø	CA3E	PRINT	CBAB	CB61
RESTORE	C952	C91F	CONT	C9AØ	C96E
GOSUB	C9C8	C996	LIST	C748	C773
RETURN	CA12	C9EØ	CLEAR	C7ØD	C738
REM	CA99	CA61	GET	CD46	CCBA
HIMEM	EBCE	E95B	CALL	E946	E8ØD
GRAB	EBE7	E974	!	CD13	CC89
RELEASE	ECØC	E994	NEW	C6EE	C719

Tableau 1- Adresses des instructions

Les douze instructions **SOUND** à **INK** (**tableau** 2) entrent toutes dans la même routine, avec un test supplémentaire pour sept d'entre elles. Cette routine détermine le nombre de paramètres nécessaires, pousse ces paramètres dans la zone PARAMS de la page 2 puis donne l'adresse des routines de ces instructions en y sautant de la même manière que celle employée par la routine C915 vue plus haut, en changeant l'adresse de retour.

Instruction	*ATMOS*	*ORIC 1*	*Paramètres*
SOUND	FB4Ø	F41E	3
MUSIC	FC18	F424	4
PLAY	FBDØ	F421	4
CURSET	FØC8	EBDF	3
CURMOV	FØFD	EBE2	3
DRAW	F11Ø	EBE5	3
CIRCLE	F37F	EBE8	2
PATTERN	F11D	EBEB	1
FILL	F268	EBFA	3
CHAR	F12D	EBEE	3
PAPER	F2Ø4	EBF4	1
INK	F21Ø	EBF7	1

Tableau 2 - Adresses des instructions avec paramètres obligatoires

Ces routines Basic, gérées par l'interpréteur, vérifient le nombre et la validité des paramètres. En C.M. nous ne le faisons pas, nous devrons donc introduire des paramètres exacts. De plus ces paramètres doivent être des entiers et codés sur 16 bits. La zone PARAMS commence à 2EØ. Cet octet doit être à Ø, au retour il sera à 1 si une erreur a été reconnue.

Exemple 1 : Traçons un **cercle en HIRES.**
Entrez le programme suivant pour ATMOS :

```
37888   00,78,00,64,00,01,00                DEF CURSET 120,100,1
37895   00,40,00,01,00                      DEF CIRCLE 64,1
37900   20,9F,FA  ENT       JSR FA9F        ;PING
37903   20,33,EC            JSR EC33        ;HIRES
37906   A0,06               LDY 6
37908   B9,00,94          ┌►LDA (37888+Y)
37911   99,E0,02          │ STA (736+Y)
37914   88                │ DEY
37915   10,F7             └─ BPL 2-9
37917   20,C8,F0            JSR F0C8        ;CURSET
37920   A0,04               LDY 4
37922   B9,07,94          ┌►LDA (37895+Y)
37925   99,E0,02          │ STA (736+Y)
37928   88                │ DEY
37929   10,F7             └─ BPL 2-9
37931   20,7F,F3            JSR F37F        ;CIRCLE
37934   60                  RTS
```

Pour l'ORIC 1, changez les quatre adresses des lignes où il y a des commentaires.

Faites CALL 379ØØ

Remarque sur les sons

L'ORIC possède un circuit merveilleux pour produire toutes sortes de sons. Ceux-ci, étant essentiellement temporels, leurs programmations en C.M. sont superflues. Il n'y a aucun gain de temps d'exécution. Si vous désirez rester dans un programme en C.M. sans être obligé de revenir au Basic pour programmer un son, vous avez tous les éléments pour le faire. Il y a beaucoup de paramètres à entrer.

Les fonctions

Il y a différentes catégories de fonction et les routines que l'interpréteur consulte pour extraire leurs adresses, sont nombreuses et ne seront pas démontées ici, car il faudrait y consacrer trop de pages. La catégorie qui va de **SGN** à **MID$** (**tableau** 3) se traite différemment selon que la fonction est comprise entre **SGN** et **CHR$** ou entre **PI** et **POINT** ou entre **LEFT$** et **MID$**.

On obtient les adresses de ces fonctions dans la ROM, de 49290 à 49354 pour les deux ORIC.

Notons qu'il ne faut pas ajouter 1 à ces adresses car elles sont manipulées de la façon suivante à la fin de la routine qui les appelle :

```
LDA (49118+Y)
STA (C4)
LDA (49119+Y)
STA (C5)
JSR C3
```

Y tient un nombre pair qui va de 172 à 236 selon la fonction ; C3 contient le nombre 4Ch et la routine C3 correspond à 4C,adresse soit JMP à l'adresse poussée en C4 et C5. On ne peut ici employer PHA, parce que le retour doit se faire dans cette routine, donc **JSR** C3 est obligatoire. Nous avons aussi un moyen de contrôle : l'adresse tenue en C4 et C5 est l'adresse de la dernière fonction utilisée.

Fonction	*ATMOS*	*ORIC 1*	*Fonction*	*ATMOS*	*ORIC 1*
SGN	DF21	DF12	FRE	D47E	D3D6
INT	DFBD	DFA5	POS	D4A6	D3FA
ABS	DF49	DF31	HEX$	D9B5	D917
USR	21	21	&	2FB	2FB

Fonction	*ATMOS*	*ORIC 1*	*Fonction*	*ATMOS*	*ORIC 1*
SQR	E22E	E22A	VAL	D8D7	D81C
RND	E34F	E34B	ASC	D8B5	D7FA
LN	DCAF	DC79	CHR$	D816	D75B
EXP	E2AA	E2A6	PI	DE77	D8EE
COS	E38B	E387	TRUE	DFØF	DFØØ
SIN	E392	E38E	FALSE	DFØB	DEFC
TAN	E3DB	E3D7	KEY$	DADA	DA4F
ATN	E43F	E43B	SCRN	DA3F	D9B4
PEEK	D938	D87D	POINT	EC45	E9CD
DEEK	D983	D8C8	LEFT$	D82A	D76F
LOG	DDD4	DDDØ	RIGHT$	D856	D79B
LEN	D8A6	D7EB	MID$	D861	D7A6
STR$	D593	D4D8			

Tableau 3- Adresses des fonctions

Une autre catégorie de fonction, de **TAB(** à STEP, code 194 à 2Ø3, est traitée au fur et à mesure de leurs apparitions. Ce sont surtout les instructions **PRINT**, **IF** et **FOR** qui les utilisent. L'instruction **PRINT**, par exemple, considère la virgule et le point-virgule comme des fonctions ; d'autres instructions traitent la virgule comme un séparateur de paramètres. Tout est cas d'espèce.

Les routines CF17 (CE8B pour l'ORIC 1) et DØØØ (CF74 pour l'ORIC 1) manipulent et extraient les fonctions. Leurs études sont très instructives, mais quelquefois la voie suivie par l'ATMOS ou l'ORIC 1 n'est pas semblable et nous ne saurions dupliquer les chemins suivis pour chaque fonction. Cependant ces routines nous donnent les adresses, dans la ROM de CØCC à CØE9 - 49356 à 49385 (idem pour ORIC 1), de la catégorie des fonctions, code 2Ø4 à 213, qui nécessitent deux opérandes. L'index Y est ici un multiple de 3 : le premier nombre sert de comparaison pour différents usages et les deux nombres suivants donnent l'adresse moins 1.

Fonction	*ATMOS*	*ORIC 1*	*Fonction*	*ATMOS*	*ORIC 1*
+	DB25	DA9A	AND	DØE6	DØ5A
-	DBØE	DA83	OR	DØE3	DØ57
*	DCFØ	DCBA	>	E271	E26D
/	DDE7	DDE3	=	DØ3C	CFBØ
↑	E238	E234	<	D113	DØ87

DIFFERENTS MOYENS D'APPELER DES CODES MACHINE : ! & ET USR

L'ORIC met à notre disposition d'autres commandes que CALL pour exploiter nos programmes écrits en C.M. Cela permet des raccourcis ou la création de nouvelles instructions et fonctions.

L'instruction !

Le signe point d'exclamation !, employé comme instruction est codé 192. Faisant partie des instructions, ce signe s'emploie sans argument ou avec des arguments, sans parenthèses, selon le programme en C.M. auquel il fait appel. La routine appelée se situe *(voir tableau plus haut)* à CD13 pour l'ATMOS et à CC89 pour l'ORIC 1. Cette routine n'a qu'une instruction :

```
6C,F5,02        JMP 2F5,IND
```

Ce saut indirect nous oblige à entrer dans 2F5 et 2F6 - 757 et 758d - l'adresse où commence notre routine.

Supposons que nous voulions remplacer **EXPLODE** par **!** , intéressant si on emploie souvent EXPLODE. Il suffit de faire DOKE#2F5,#FACB (F418 pour l'ORIC 1) ; par la suite on emploie ! comme instruction Basic. Essayez, faites **! RUN.** C'est plus court et cela prend moins de place.

Créons une nouvelle instruction avec entrée d'argument.

Exemple 2 : Création de l'instruction **RESTORE** l où l est un numéro de ligne.

Cette instruction nouvelle est intéressante puisque nous ne disposons que de **RESTORE** qui renvoie toujours le pointeur de lecture à la lère ligne de **DATA** ; et avec **! l**, on peut repointer sur n'importe quelle ligne de **DATA.**

Entrez le programme suivant :

```
37888   A5,E9          LDA (E9)        ;pour decrementer
37890   D0,02          BNE 2+2         ;   l'adresse
37892   C6,EA          DEC (EA)        ;   utilisee
37894   C6,E9          DEC (E9)        ;   dans E2
```

```
37896   20,E2,00      JSR E2          ;charge A
37899   B0,FB         BCS 2-5         ;attend 1 chiffre
37901   20,E2,CA      JSR CAE2        ;CA98 pour ORIC 1
37904   A5,9A         LDA (9A)        ;transfert
37906   85,B0         STA (B0)        ;   du depart
37908   A5,9B         LDA (9B)        ;   de la zone
37910   85,B1         STA (B1)        ;   Basic
37912   A0,02         LDY 2
37914   B1,B0         LDA (B0),INDY   ;no de ligne bas
37916   C5,33         CMP (33)        ;compare octet bas
37918   D0,10         BNE 2+16
37920   C8            INY
37921   B1,B0         LDA (B0),INDY   ;no de ligne haut
37923   C5,34         CMP (34)        ;compare octet haut
37925   D0,09         BNE 2+9
37927   A5,B0         LDA (B0)
37929   D0,02         BNE 2+2
37931   C6,B1         DEC (B1)
37933   C6,B0         DEC (B0)
37935   60            RTS
37936   A0,00         LDY 0
37938   B1,B0         LDA (B0),INDY   ;adresse ligne
37940   AA            TAX             ;   Basic
37941   C8            INY             ;   suivante
37942   B1,B0         LDA (B0),INDY   ;   bas et haut
37944   A8            TAY             ;teste la fin
37945   D0,03         BNE 2+3         du programme
37947   8A            TXA             ;   Basic
37948   F0,F1         BEQ 2-15        ;
37950   86,B0         STX (B0)        ;actualise le
37952   84,B1         STY (B1)        ;   pointeur DATA
37954   D0,D4         BNE 2-44        ;saut oblige
```

Nous avons placé cette routine à 37888 ; vous pouvez la mettre à un autre endroit car elle est entièrement relogeable mais n'oubliez pas de changer la ligne 2 du Basic ci-dessous.

Explication

L'appel de la routine E2 charge A avec les caractères qui suivent le signe ! jusqu'à rencontre d'un chiffre.

On appelle ensuite la routine CAE2, qui demande le chargement de A avant l'appel. Cette routine transforme les chiffres qu'elle rencontre en un nombre binaire, codé sur 2 octets, qu'elle pousse dans les octets 33 et 34h.

L'adresse du pointeur de **DATA** est tenue dans les octets BØ et B1. Nous les chargeons avec le départ de la

zone Basic, tenu par les octets 9A et 9B. Nous comparons ensuite (octet bas puis haut) le numéro de ligne Basic avec le numéro choisi dans notre ligne ! . Si la comparaison tient, on décrémente l'adresse tenue par BØ et Bl puis retour.

Si la comparaison ne tient pas, on extrait l'adresse de la ligne Basic suivante. Si celle-ci contient Ø, c'est la fin de la zone Basic et on retourne. Si ce n'est pas la fin, on actualise BØ et Bl et on boucle pour explorer la ligne Basic suivante. Illustration :

```
1 HIMEM 37887
2 DOKE 757,37888
5 CLS
10 DATA 1,2,3
20 DATA 4,5,6
30 ! 100
40 READ A :PRINT A
100 DATA 33,60
500 END
```

RUN et l'écran affiche 33.

Faites 3Ø ! 2Ø puis RUN et vous aurez 4.

La fonction &

Le signe 'et contracté' s'emploie comme fonction et possède alors le code 221. A cet endroit, elle fait partie d'une catégorie de fonction qui oblige l'emploi d'un argument entre parenthèses. La routine est située à 2FB pour les deux ORIC. L'octet 2FB contient 4C soit JMP, nous devrons donc, avant l'emploi, faire un DOKE à 2FC avec l'adresse où nous aurons mis notre routine. Comme toute fonction, elle doit être précédée d'une instruction.

A l'appel, le nombre qui est entre parenthèses est converti en virgule flottante et placé dans l'accumulateur, adresse DØ à D5. DØ tient l'exposant +128 ; Dl à D4 tiennent le nombre avec le bit 7 de Dl toujours à 1, et D5=Ø pour un nombre positif.

Au retour (à RTS), le nombre situé dans l'accumulateur est retourné dans le Basic. Suivant l'instruction, il s'affiche (PRINT) ou est affecté à une variable (LET).

Ecrivons une routine qui manipule l'accumulateur afin de retourner un autre nombre que celui entré.

Exemple 3 : **Conversion d'un nombre** de Ø à 255 en sa forme binaire.

```
37888   A5,D0          LDA (D0)        ;exposant
37890   29,7F          AND 7F          ;moins 128
37892   AA             TAX
37893   06,D1       ┌─►ASL (D1)       ;transfert du
37895   26,D4       │  ROL (D4)       ;   nombre dans D4
37897   CA          │  DEX
37898   D0,F9       └─ BNE 2-7
37900   A0,08          LDY 8           pour examiner les
                                       ;   8 bits de D4
37902   06,D4    ┌────►ASL (D4)       ;bit 7 dans C
37904   A2,03    │     LDX 3          ;pour D1 D2 et D3
37906   B5,D0    │  ┌─►LDA (D0+X)
37908   69,00    │  │  ADC 0          ;ajoute C
37910   95,D0    │  │  STA (D0+X)
37912   95,D8    │  │  STA (D8+X)     ;stockage intermediaire
37914   CA       │  │  DEX
37915   D0,F5    │  └─ BNE 2-11
37917   88       │     DEY
37918   F0,17    │  ┌─ BEQ 2+23
37920   20,46,94 │  │  JSR 37958      ;fois 2
37923   20,46,94 │  │  JSR 37958      ;fois 4
37926   A2,03    │  │  LDX 3
37928   B5,D0    │  │┌►LDA (D0+X)
37930   75,D8    │  ││ ADC (D8+X)     ;fois 5
37932   95,D0    │  ││ STA (D0+X)
37934   CA       │  ││ DEX
37935   D0,F7    │  │└─ BNE 2-9
37937   20,46,94 │  │  JSR 37958      ;fois 10
37940   4C,0E,94 └──┼─ JMP 37902
37943   A2,18       └►LDX 24
37945   24,D1       ┌►BIT (D1)        ;jusqu'a ce que
37947   30,10       │ BMI 2+16        ;   le bit 7
37949   20,46,94    │ JSR 37958       ;   de D1
37952   CA          │ DEX             ;   soit a 1
37953   D0,F6       └─ BNE 2-10       ;
37955   86,D0          STX (D0)
37957   60             RTS             ;representation de 0
37958   26,D3     SR   ROL (D3)        ;multiplie D1
37960   26,D2          ROL (D2)        ;   D2 et D3
37962   26,D1          ROL (D1)        ;   par 2
37964   60             RTS             ;retour de SR
37965   8A          ┌─►TXA             ;exposant
37966   69,80       │  ADC 128        ;plus 128
37968   85,D0       │  STA (D0)       ;dans D0
37970   60          │  RTS
```

On transfère D1, qui tient le nombre entre parenthèses, dans D4 en rétablissant le poids des bits.

A l'aide de huit glissements de D4, on traite chaque bit de D4. Le traitement consiste à multiplier par 1Ø chaque bit pour obtenir la forme binaire. Par exemple, le nombre 5 est représenté par 1Ø1 dans D4. Les cinq premiers glissements laissent D1, D2 et D3 à Ø puisqu'on multiplie Ø par 1Ø. Le 6e glissement amène 1 dans D3, que l'on multiplie par 1Ø. Le 7e glissement amène Ø dans D3, et on multiplie 1Ø par 1Ø soit 1ØØ. Le 8e glissement amène 1 qu'on additionne pour faire 1Ø1. Y vient à Ø et on ne multiplie plus.

On manipule ensuite l'ensemble D1, D2 et D3, en le faisant glisser à gauche jusqu'à ce que le bit 7 de D1 soit un 1. Ceci est nécessaire, car le premier chiffre binaire significatif d'un nombre en virgule flottante est toujours 1. En même temps on manipule l'exposant, tenu en X, que l'on charge à 24 puisqu'il y a 3 octets qui peuvent tenir un nombre de maximum 11111111.

Pour représenter Ø on met DØ à Ø ; ce que fait la ligne 37955 si on avait entré Ø.

Cette routine n'est pas relogeable, sauf si vous changez les cinq adresses absolues qu'elle contient.

Pour illustrer, entrez le Basic :

```
1 HIMEM 37887
2 DOKE 764,37888
10 A=255
20 PRINT &(A)
100 STOP
```

RUN vous affiche 11111111

Changez la ligne 1Ø pour avoir d'autres représentations binaires. Il est évident que vous pouvez vous passer de la variable A et entrer le nombre directement dans la ligne 2Ø.

La fonction USR

La fonction **USR** pɒssède les mêmes caractéristiques que la fonction précédente. Vous devez faire un DOKE à 22h - 34d avec l'adresse où se trouve la routine. Si vous avez gardé la routine précédente, le Basic suivant donne le même résultat :

```
2 DOKE 34,37888
10 A=255
20 PRINT USR(A)
```

Plutôt que d'employer DOKE, vous avez la faculté de faire :

```
2 DEF USR=37888
```

c'est la même chose.

Les fonctions **&** et **USR** sont intéressantes pour créer des fonctions nouvelles en C.M.. Pour une combinaison de fonctions existantes, vous disposez de **DEF FN.**

Ces fonctions retournent directement une valeur dans le Basic. L'emploi de l'instruction **CALL** permet d'en faire autant, avec une ligne Basic en plus. Si notre routine x a déposé un résultat dans la cellule 38ØØØ, il vient :

```
POKE adr,1Ø
CALL x
PRINT PEEK(38ØØØ)
```

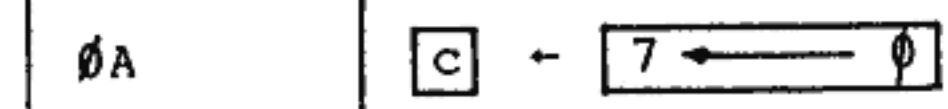
alors qu'avec une fonction programmée, on aurait eu

```
DOKE 34,x
PRINT USR(1Ø)
```

Si on ne doit pas entrer de paramètre dans la routine, le **CALL** est alors équivalent.

ANNEXE I

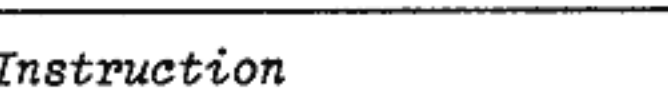
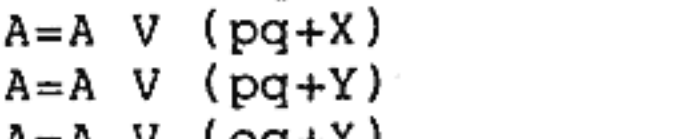
LISTE ALPHABETIQUE DES MNEMONIQUES

Mnémonique	*Codes*	*Instruction*	
ADC n	69,nn	A=A+n+C	immédiat
ADC pq	6D,qq,pp	A=A+(pq)+C	absolu
ADC q	65,qq	A=A+(oq)+C	page Ø
ADC pq,X	7D,qq,pp	A=A+(pq+X)+C	absolu indexé
ADC pq,Y	79,qq,pp	A=A+(pq+Y)+C	absolu indexé
ADC q,X	75,qq	A=A+(oq+X)+C	page Ø indexé
ADC d,X	61,dd	l=(od+X)	indirect pré-indexé
		h=(od+X+1)	pré-indexé
		A=A+(hl)+C	
ADC d,Y	71,dd	l=(od)	indirect
		h=(od+1)	post-indexé
		A=A+(hl+Y)+C	
AND n	29,nn	A=AΛn	
AND pq	2D,qq,pp	A=AΛ(pq)	
AND q	25,qq	A=AΛ(oq)	
AND pq,X	3D,qq,pp	A=AΛ(pq+X)	
AND pq,Y	39,qq,pp	A=AΛ(pq+Y)	
AND q,X	35,qq	A=AΛ(oq+X)	
AND d,X	21,dd	l=(od+X)	
		h=(od+X+1)	
		A=AΛ(hl)	
AND d,Y	31,dd	l=(od)	
		h=(od+1)	
		A=AΛ(hl+Y)	
ASL	ØA		
		glissement arithmétique à gauche sur A	
ASL pq	ØE,qq,pp	glissement arithmétique à gauche sur (pq)	
ASL q	Ø6,qq	glissement arithmétique à gauche sur (oq)	
ASL pq,X	1E,qq,pp	glissement arithmétique à gauche sur (pq+X)	
ASL q,X	16,qq	glissement arithmétique à gauche sur (oq+X)	
BCC e	9Ø,ee	PC=PC+2+e	si C=Ø
		PC=PC+2	si C=1

Mnémonique	*Codes*	*Instruction*
BCS e	Bø,ee	PC=PC+2+e si C=1 PC=PC+2 si C=ø
BEQ e	Fø,ee	PC=PC+2+e si Z=1 PC=PC+2 si Z=ø
BNE e	Dø,ee	PC=PC+2+e si Z=ø PC=PC+2 si Z=1
BMI e	3ø,ee	PC=PC+2+e si N=1 PC=PC+2 si N=ø
BPL e	1ø,ee	PC=PC+2+e si N=ø PC=PC+2 si N=1
BVC e	5ø,ee	PC=PC+2+e si V=ø PC=PC+2 si V=1
BVS e	7ø,ee	PC=PC+2+e si V=1 PC=PC+2 si V=ø
BIT pq	2C,qq,pp	$Z=\overline{A \wedge (pq)}$ N=(pq), $V=(pq)_6$
BIT q	24,qq	$Z=\overline{A \wedge (oq)}$ N=(oq), $V=(oq)_6$
BRK	øø	PC=PC+2 (SP)=PCh SP=SP-1 (SP)=PCb SP=SP-1 (SP)=P SP=SP-1 PCb=(FFFE) PCh=(FFFF) **Attention** : c'est le PC+2 qui est sauvé dans la pile.
CLC	18	C=ø
CLD	D8	D=ø met le mode binaire pour ADC et SBC
CLI	58	I=ø pour autoriser les interruptions
CLV	B8	V=ø

Mnémonique	*Codes*	*Instruction*
CMP n	C9,nn	A-n
CMP pq	CD,qq,pp	A-(pq)
CMP q	C5,qq	A-(oq)
CMP pq,X	DD,qq,pp	A-(pq+X)
CMP pq,Y	D9,qq,pp	A-(pq+Y)
CMP q,X	D5,qq	A-(oq+X)
CMP d,X	C1,dd	l=(od+X)
		h=(od+X+1)
		A-(hl)
CMP d,Y	D1,dd	l=(od)
		h=(od+1)
		A-(hl+Y)
CPX n	EØ,nn	X-n
CPX pq	EC,qq,pp	X-(pq)
CPX q	E4,qq	X-(oq)
CPY n	CØ,nn	Y-n
CPY pq	CC,qq,pp	Y-(pq)
CPY q	C4,qq	Y-(oq)
DEC pq	CE,qq,pp	(pq)=(pq)-1
DEC q	C6,qq	(oq)=(oq)-1
DEC pq,X	DE,qq,pp	(pq+X)=(pq+X)-1
DEC q,X	D6,qq	(oq+X)=(oq+X)-1
DEX	CA	X=X-1
DEY	88	Y=Y-1
EOR n	49,nn	A=A ⊻ n
EOR pq	4D,qq,pp	A=A ⊻ (pq)
EOR q	45,qq	A=A ⊻ (oq)
EOR pq,X	5D,qq,pp	A=A ⊻ (pq+X)
EOR pq,Y	59,qq,pp	A=A ⊻ (pq+Y)
EOR q,X	55,qq	A=A ⊻ (oq+X)
EOR d,X	41,dd	l=(od+X)
		h=(od+X+1)
		A=A ⊻ (hl)
EOR d,Y	51,dd	l=(od)
		h=(od+1)
		A=A ⊻ (hl+Y)
INC pq	EE,qq,pp	(pq)=(pq)+1
INC q	E6,qq	(oq)=(oq)+1
INC pq,X	FE,qq,pp	(pq+X)=(pq+X)+1
INC q,X	F6,qq	(oq+X)=(oq+X)+1
INX	E8	X=X+1
INY	C8	Y=Y+1

Mnémonique	*Codes*	*Instruction*
JMP pq	4C,qq,pp	PC=pq
JMP pq,IND	6C,qq,pp	l=(pq) h=(pq+1) PC=hl
JSR pq	2Ø,qq,pp	PC=PC+2 (SP)=PCh SP=SP-1 (SP)=PCb SP=SP-1 PC=pq **Attention** : c'est le PC+2 qui est sauvé.
LDA n	A9,nn	A=n
LDA pq	AD,qq,pp	A=(pq)
LDA q	A5,qq	A=(oq)
LDA pq,X	BD,qq,pp	A=(pq+X)
LDA pq,Y	B9,qq,pp	A=(pq+Y)
LDA q,X	B5,qq	A=(oq+X)
LDA d,X	A1,dd	l=(od+X) h=(od+X+1) A=(hl)
LDA d,Y	B1,dd	l=(od) h=(od+1) A=(hl+Y)
LDX n	A2,nn	X=n
LDX pq	AE,qq,pp	X=(pq)
LDX q	A6,qq	X=(oq)
LDX pq,Y	BE,qq,pp	X=(pq+Y)
LDX q,Y	B6,qq	X=(oq+Y)
LDY n	AØ,nn	Y=n
LDY pq	AC,qq,pp	Y=(pq)
LDY q	A4,qq	Y=(oq)
LDY pq,X	BC,qq,pp	Y=(pq+X)
LDY q,X	B4,qq	Y=(oq+X)
LSR	4A	Ø → [7 ⟶ Ø] → [C] glissement logique à droite sur A
LSR pq	4E,qq,pp	glissement logique à droite sur (pq)
LSR q	46,qq	glissement logique à droite sur (oq)
LSR pq,X	5E,qq,pp	glissement logique à droite sur (pq+X)
LSR q,X	56,qq	glissement logique à droite sur (oq+X)

Mnémonique	*Codes*	*Instruction*
NOP	EA	aucune
ORA n	Ø9,nn	A=A V n
ORA pq	ØD,qq,pp	A=A V (pq)
ORA q	Ø5,qq	A=A V (oq)
ORA pq,X	1D,qq,pp	A=A V (pq+X)
ORA pq,Y	19,qq,pp	A=A V (pq+Y)
ORA q,X	15,qq	A=A V (oq+X)
ORA d,x	Ø1,dd	l=(od+X) h=(od+X+1) A=A V (hl)
ORA d,Y	11,dd	l=(od) h=(od+1) A=A V (hl+Y)
PHA	48	(SP)=A SP=SP-1
PHP	Ø8	(SP)=P SP=SP-1
PLA	68	A=(SP) SP=SP+1
PLP	28	P=(SP) SP=SP+1
ROL	2A	 rotation 9 bits à gauche sur A
ROL pq	2E,qq,pp	rotation 9 bits à gauche sur (pq)
ROL q	26,qq	rotation 9 bits à gauche sur (oq)
ROL pq,X	3E,qq,pp	rotation 9 bits à gauche sur (pq+X)
ROL q,X	36,qq	rotation 9 bits à gauche sur (oq+X)
ROR	6 A	 rotation 9 bits à droite sur A
ROR pq	6E,qq,pp	rotation 9 bits à droite sur (pq)
ROR q	66,qq	rotation 9 bits à droite sur (oq)
ROR pq,X	7E,qq,pp	rotation 9 bits à droite sur (pq+X)
ROR q,X	76,qq	rotation 9 bits à droite sur (oq+X)

Mnémonique	*Codes*	*Instruction*
RTI	4Ø	P=(SP) SP=SP+1 PCb=(SP) SP=SP+1 PCh=(SP) SP=SP+1
RTS	6Ø	PCb=(SP) SP=SP+1 PCh=(SP) SP=SP+1 PC=PC+1
SBC n	E9,nn	A=A-n-$\overline{C}$
SBC pq	ED,qq,pp	A=A-(pq)-$\overline{C}$
SBC q	E5,qq	A=A-(oq)-$\overline{C}$
SBC pq,X	FD,qq,pp	A=A-(pq+X)-$\overline{C}$
SBC pq,Y	F9,qq,pp	A=A-(pq+Y)-$\overline{C}$
SBC q,X	F5,qq	A=A-(oq+X)-$\overline{C}$
SBC d,X	E1,dd	l=(od+X) h=(od+X+1) A=A-(hl)-$\overline{C}$
SBC d,Y	F1,dd	l=(od) h=(od+1) A=A-(hl+Y)-$\overline{C}$
SEC	38	C=1
SED	F8	D=1 met le mode décimal pour ADC et SBC
SEI	78	I=1 masque les interruptions
STA pq	8D,qq,pp	(pq)=A
STA q	85,qq	(oq)=A
STA pq,X	9D,qq,pp	(pq+X)=A
STA pq,Y	99,qq,pp	(pq+Y)=A
STA q,X	95,qq	(oq+X)=A
STA d,X	81,dd	l=(od+X) h=(od+X+1) (hl)=A
STA d,Y	91,dd	l=(od) h=(od+1) (hl+Y)=A
STX pq	8E,qq,pp	(pq)=X
STX q	86,qq	(oq)=X
STX q,Y	96,qq	(oq+Y)=X

Mnémonique	*Codes*	*Instruction*
STY pq	8C,qq,pp	(pq)=Y
STY q	84,qq	(oq)=Y
STY q,X	94,qq	(oq+X)=Y
TAX	AA	X=A
TAY	A8	Y=A
TSX	BA	X=SP
TXA	8A	A=X
TXS	9A	SP=X
TYA	98	A=Y

ANNEXE II

LISTE NUMERIQUE DES MNEMONIQUES

Poids 16¹								*16°*
Ø	*1*	*2*	*3*	*4*	*5*	*6*	*7*	
BRK	BPLe	JSR pq	BMI e	RTI	BVCe	RTS	BVS e	Ø
ORA d,X	ORA d,Y	AND d,X	AND d,Y	EOR d,X	EOR d,Y	ADC d,X	ADC d,Y	1
								2
								3
		BIT q						4
ORA q	ORA q,X	AND q	AND q,X	EOR q	EOR q,X	ADC q	ADC q,X	5
ASL q	ASL q,X	ROL q	ROL q,X	LSR q	LSR q,X	ROR q	ROR q,X	6
								7
PHP	CLC	PLP	SEC	PHA	CLI	PLA	SEI	8
ORA n	ORA pq,Y	AND n	AND pq,Y	EOR n	EOR pq,Y	ADC n	ADC pq,Y	9
ASL		ROL		LSR		ROR		A
								B
		BIT pq		JMP pq		JMP pq IND		C
ORA pq	ORA pq,X	AND pq	AND pq,X	EOR pq	EOR pq,X	ADC pq	ADC pq,X	D
ASL pq	ASL pq,X	ROL pq	ROL pq,X	LSR pq	LSR pq,X	ROR pq	ROR pq,X	E
								F

8	*9*	*A*	*B*	*C*	*D*	*E*	*F*	
	BCC e	LDY n	BCS e	CPY n	BNE e	CPX n	BEQ e	Ø
STA d,X	STA d,Y	LDA d,X	LDA d,Y	CMP d,X	CMP d,Y	SBC d,X	SBC d,Y	1
		LDX n						2
								3
STY q	STY q,X	LDY q	LDY q,X	CPY q		CPX q		4
STA q	STA q,X	LDA q	LDA q,X	CMP q	CMP q,X	SBC q	SBC q,X	5
STX q	STX q,Y	LDX q	LDX q,Y	DEC q	DEC q,X	INC q	INC q,X	6
								7
DEY	TYA	TAY	CLV	INY	CLD	INX	SED	8
	STA pq,Y	LDA n	LDA pq,Y	CMP n	CMP pq,Y	SBC n	SBC pq,Y	9
TXA	TXS	TAX	TSX	DEX		NOP		A
								B
STY pq		LDY pq	LDY pq,X	CPY pq		CPX pq		C
STA pq	STA pq,X	LDA pq	LDA pq,X	CMP pq	CMP pq,X	SBC pa	SBC pq,X	D
STX pq		LDX pq	LDX pq,Y	DEC pq	DEC pq,X	INC pq	INC pq,X	E
								F

ANNEXE III

INFLUENCE DES MNEMONIQUES SUR LES INDICATEURS

Mnémoniques	*N*	*V*	*Z*	*C*
ADC, SBC, PLP, RTI	x	x	x	x
ASL, CMP, CPX, CPY, ROL, ROR	x		x	x
LSR	ϕ		x	x
AND, DEC, DEX, DEY, EOR, INC, INX, INY,	x		x	
LDA, LDX, LDY, ORA, PLA, TAX, TAY, TXA	x		x	
TYA, TSX	x		x	
BIT	(1)	(2)	x	
CLC				ϕ
SEC				1
CLV		ϕ		
CLD met D à ϕ ; SED met D à 1				
CLI met I à ϕ ; SEI met I à 1				
BCC, BCS, BEQ, BNE, BMI, BPL, BVC, BVS	pas d'effet			
JMP, JSR, NOP, PHA, PHP, RTS, STA, STX	pas d'effet			
STY, TXS	pas d'effet			

(1) le bit 7 du contenu mémoire est mis dans N

(2) le bit 6 du contenu mémoire est mis dans V

PLP et **RTI** influencent aussi les indicateurs **B**, **D** et **I**

BRK met **I** à 1 et également **B** à 1 dans le registre **P** qui est sauvé sur la pile pour différencier **BRK** de **IRQ**

ANNEXE IV

TABLE DES NOMBRES AVEC SIGNE POUR LES VALEURS NEGATIVES

h	*d comp. à 2*	*h*	*d comp. à 2*	*h*	*d comp. à 2*	*h*	*d comp. à 2*
8Ø	-128	AØ	-96	CØ	-64	EØ	-32
81	-127	A1	-95	C1	-63	E1	-31
82	-126	A2	-94	C2	-62	E2	-3Ø
83	-125	A3	-93	C3	-61	E3	-29
84	-124	A4	-92	C4	-6Ø	E4	-28
85	-123	A5	-91	C5	-59	E5	-27
86	-122	A6	-9Ø	C6	-58	E6	-26
87	-121	A7	-89	C7	-57	E7	-25
88	-12Ø	A8	-88	C8	-56	E8	-24
89	-119	A9	-87	C9	-55	E9	-23
8A	-118	AA	-86	CA	-54	EA	-22
8B	-117	AB	-85	CB	-53	EB	-21
8C	-116	AC	-84	CC	-52	EC	-2Ø
8D	-115	AD	-83	CD	-51	ED	-19
8E	-114	AE	-82	CE	-5Ø	EE	-18
8F	-113	AF	-81	CF	-49	EF	-17
9Ø	-112	BØ	-8Ø	DØ	-48	FØ	-16
91	-111	B1	-79	D1	-47	F1	-15
92	-11Ø	B2	-78	D2	-46	F2	-14
93	-1Ø9	B3	-77	D3	-45	F3	-13
94	-1Ø8	B4	-76	D4	-44	F4	-12
95	-1Ø7	B5	-75	D5	-43	F5	-11
96	-1Ø6	B6	-74	D6	-42	F6	-1Ø
97	-1Ø5	B7	-73	D7	-41	F7	-9
98	-1Ø4	B8	-72	D8	-4Ø	F8	-8
99	-1Ø3	B9	-71	D9	-39	F9	-7
9A	-1Ø2	BA	-7Ø	DA	-38	FA	-6
9B	-1Ø1	BB	-69	DB	-37	FB	-5
9C	-1ØØ	BC	-68	DC	-36	FC	-4
9D	-99	BD	-67	DD	-35	FD	-3
9E	-98	BE	-66	DE	-34	FE	-2
9F	-97	BF	-65	DF	-33	FF	-1

ANNEXE V

CONVERSION HEXADECIMAL - DECIMAL

Chiffre hexa.	*Poids* 16^3	*Poids* 16^2	*Poids* 16^1	*Poids* 16^0
Ø	Ø	Ø	Ø	Ø
1	4Ø96	256	16	1
2	8192	512	32	2
3	12288	768	48	3
4	16384	1Ø24	64	4
5	2Ø48Ø	128Ø	8Ø	5
6	24576	1536	96	6
7	28672	1792	112	7
8	32768	2Ø48	128	8
9	36864	23Ø4	144	9
A	4Ø96Ø	256Ø	16Ø	1Ø
B	45Ø56	2816	176	11
C	49152	3Ø72	192	12
D	53248	3328	2Ø8	13
E	57344	3584	224	14
F	6144Ø	384Ø	24Ø	15

Achevé d'imprimer en août 1984
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : août 1984

N° d'impression : 404069
N° d'édition : 86595-168-1
ISBN : 2-86595-168-5

EDITIONS DU P.S.I.

90,00 FF

l'assembleur de l'ORIC 1 et ORIC-ATMOS

Ce livre nous apprend à programmer dans le langage machine du 6502A, orienté vers l'Oric 1 et l'Oric Atmos, dont le jeu d'instructions est expliqué et utilisé. L'assembleur symbolique et ses mnémoniques sont décrits. Les interactions avec le Basic et avec le système sont étudiées. Enfin, l'opportunité de la programmation en langage machine est discutée (place mémoire et temps d'exécution).

EDITIONS DU PSI
BP 86
France - 77402 lagny s/marne cedex
ISBN 2-86595-168-5
imprimé en France

COUVERTURE A GRAPH

Téléchargé sur
Le Vieux Manuel

WWW.MANUELS.ABANDONWARE-FRANCE.ORG