

**ASSEMBLEUR  
ET  
PÉRIPHÉRIQUES  
DES  
MO5 ET TO7/70**

---

### **Autres ouvrages relatifs aux M05 et T07/70**

---

- M05 et T07/70 pour tout petit — Daniel Nielsen
- M05 et T07/70 à l'école — Daniel Nielsen
- M05 et T07/70 pour tous — Jacques Boisgontier et Sophie Delaur
- Le Dasté des M05 et T07/70 — Gilles Blanchard
- La découverte du M05 — Dominique Schraen et Maurice Charbit
- Exercices pour M05 — Dominique Schraen et Maurice Charbit
- La découverte du T07/70 — Dominique Schraen et Maurice Charbit
- Exercices pour T07/70 — Maurice Charbit et Dominique Schraen
- Jeux, trucs et comptes pour T07/70 — Michel Benelfoul
- M05 et T07/70 en famille — Jean-François Séhan
- 102 programmes pour M05 et T07/70 — Jacques Decorniat
- M05 et T07/70 : méthodes pratiques — Jacques Boisgontier
- 102 programmes pour T07 — Jacques Decorniat
- M05 et T07/70 à l'affiche — Jean-François Séhan

### **A paraître :**

- Clefs pour M05 — Gilles Blanchard

La loi du 1 mars 1957 (s'appliquant, aux termes des articles 2 et 3, de l'article 41, d'une part, que les œuvres de reproduction sont réservées à l'usage personnel, 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, « ou la représentation ou reproduction intégrale ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 2<sup>o</sup> 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.

# **ASSEMBLEUR ET PÉRIPHÉRIQUES DES MO5 ET TO7/70**

**Frédéric Blanc et François Normand**

**Guide  
Pratique**



**Editions du P.S.I.**

**1985**

## Introduction

Cet ouvrage comporte deux parties axées différemment mais construites dans la même optique.

D'abord il vous permet d'apprendre à programmer vos MO5 et TO7/70 directement dans le langage que comprend le microprocesseur, le langage machine. Le langage machine vous offre de nombreux avantages par rapport au BASIC, notamment un gain de rapidité considérable (un programme en langage machine tourne entre dix et mille fois plus vite qu'un programme en BASIC) et également une économie de mémoire. Malgré son côté rébarbatif au premier abord, vous serez rapidement conquis par ses capacités.

Ensuite il vous offre la possibilité d'utiliser les routines se trouvant en ROM, dans la partie intitulée « *Routines et adresses utiles* ». Cet ensemble de trucs et d'astuces permet au programmeur de gagner du temps et de la place. Cette partie ne se veut pas exhaustive car nos recherches ne nous ont pas permis de tout découvrir.

Le seul impératif avant d'aborder la lecture de ce livre est d'avoir parfaitement assimilé le BASIC et les concepts de base de la programmation sur lesquels nous ne reviendrons pas. S'il en est ainsi, nous espérons que ce livre vous permettra de vous familiariser avec votre MO5 de manière à en extraire la quintessence.

Sans plus attendre, passons aux « choses sérieuses » sans oublier de remercier tout particulièrement la société L'ORIGINE S pour sa coopération lors de l'élaboration de ce livre.

## Sommaire

<b>Première partie. — Langage machine sur MOS</b>	<b>9</b>
Table des mnémoniques	10
Basic ou langage machine	12
Présentation générale	12
Inconvénients du langage machine	13
Avantages du langage machine	14
Différences fondamentales	14
Langage d'assemblage	15
Conclusion	15
Représentation de l'information	16
Représentation en binaire simple	16
Représentation hexadécimale	17
Représentation en binaire signé	19
Représentation en complément à deux	19
Représentation en DCB	20
Représentation en virgule flottante	20
Représentation des caractères alphanumériques	22
Représentation des variables	22
Organisation d'une ligne Basic	23
Conclusion	28
Le système MOS	29
Les liaisons	30
L'horloge	30
Les mémoires ROM et RAM. le 6821	30
Le microprocesseur	32
Les modes d'adressage	34
Le concept de pile	38
Le concept d'organigramme	39

<b>Les interruptions</b>	<b>40</b>
Introduction	40
Caractéristiques	40
Interruptions du 6809	40
Détournement d'une interruption	43
<b>Le jeu d'instructions du 6809</b>	<b>44</b>
Notations	44
Modes d'indexation	46
Chargements et stockages	47
Echanges et transferts	49
Opérations sur la pile	50
Incrémentations — Décrémentations	51
Comparaisons et tests	52
Opérateurs booléens	54
Opérateurs arithmétiques	56
Décalages et rotations	60
Branchements inconditionnels et retour	63
Branchements conditionnels	65
Instructions de traitement des interruptions	71
<b>Deuxième partie. — Routines et adresses utiles</b>	<b>73</b>
<b>L'écran et le crayon optique</b>	<b>74</b>
Le système écran	74
Les couleurs de fond et d'encre	75
Le curseur	76
Le caractère d'échappement (ESC)	76
Les routines du moniteur	79
Le générateur de caractères	82
Les routines d'affichage	83
<b>Le joystick</b>	<b>84</b>
<b>Le clavier</b>	<b>87</b>
Scrutation clavier rapide	87
Scrutation clavier avec retour du code ASCII	88
Utilisation de l'entrée ligne Basic	89
Modification de ce qui se répète	90
Modification des tables du clavier	90
<b>Le magnétophone</b>	<b>91</b>
Stockage du nom du programme	91
Sauvegarde d'un programme	92
Chargement d'un programme	93
Utilisation des routines moniteur	95

<b>Le lecteur de disquettes</b>	<b>98</b>
Adresses des fonctions DOS MO5	99
Sauvegarde et chargement au format Basic	100
Format disque Microsoft	100
Utilisation des routines moniteur	102
 <b>L'interface de communication</b>	 <b>106</b>
 <b>Son et musique</b>	 <b>108</b>
Beep clavier	108
Synthétiseur musical	109
 <b>Les variables systèmes</b>	 <b>111</b>
 <b>Annexe 1. — Tableau de conversion</b>	 <b>113</b>
<b>Annexe 2. — Code des couleurs</b>	<b>118</b>
<b>Annexe 3. — Caractères de contrôle</b>	<b>118</b>
<b>Annexe 4. — Particularités du T07/70</b>	<b>119</b>





LANGAGE MACHINE | 1  
SUR M05 |

## Table des mnémoniques

Mnémonique	Page	Mnémonique	Page
ABX .....	57	BVC .....	70
ADCA .....	57	BVS .....	70
ADCB .....	57		
ADDA .....	57	CLR .....	49
ADDB .....	57	CLRA .....	49
ADDD .....	57	CLRB .....	49
ANDA .....	54	CMPA .....	53
ANDB .....	54	CMPB .....	53
ANDCC .....	54	CMPC .....	53
ASI .....	60	CMPS .....	53
ASLA .....	61	CMPJ .....	53
ASLB .....	61	CMPX .....	53
ASH .....	61	CMPI .....	53
ASRA .....	61	COM .....	56
ASRB .....	61	COMA .....	56
		COMB .....	56
		GWAI .....	71
BCC .....	65		
BOS .....	66	DAA .....	58
BEQ .....	66	DEC .....	52
BGE .....	67	DECA .....	52
BGT .....	68	DECB .....	52
BHI .....	69		
BHS .....	67	EORA .....	55
BITA .....	53	EOHB .....	55
BITB .....	53	EXG .....	49
BLE .....	67		
BLO .....	69	INC .....	52
BLS .....	60	INCA .....	52
BLT .....	68	INCB .....	52
BMI .....	69		
BNE .....	66	JMP .....	63
BPL .....	70	JSR .....	64
BRA .....	63		
BRN .....	64	LBCC .....	65
BSR .....	64		

Mnémonique	Page	Mnémonique	Page
LBCS . . . . .	56	ORA . . . . .	55
LBEO . . . . .	56	ORB . . . . .	55
LBGE . . . . .	57	ORCC . . . . .	55
LBGT . . . . .	58	PSHS . . . . .	51
LBHI . . . . .	59	PSHU . . . . .	51
LBHS . . . . .	67	PULS . . . . .	51
LBLE . . . . .	57	PULU . . . . .	51
LBLO . . . . .	59		
LBI S . . . . .	58	ROL . . . . .	62
LBLT . . . . .	58	ROI A . . . . .	62
LBMI . . . . .	59	ROLB . . . . .	62
LBNE . . . . .	56	ROR . . . . .	62
LBPL . . . . .	70	RORA . . . . .	63
LBRA . . . . .	53	RORE . . . . .	63
LBRN . . . . .	54	RTI . . . . .	72
LBSR . . . . .	54	RIS . . . . .	65
LBVC . . . . .	70		
LBVS . . . . .	70	SBCA . . . . .	59
LGA . . . . .	47	SBCB . . . . .	59
LGB . . . . .	47	SEX . . . . .	60
LCD . . . . .	47	STA . . . . .	48
LDS . . . . .	47	STB . . . . .	48
LDU . . . . .	47	STD . . . . .	48
LDX . . . . .	47	STS . . . . .	48
LDY . . . . .	47	STU . . . . .	48
LEAS . . . . .	48	STX . . . . .	48
LEAU . . . . .	48	STY . . . . .	48
LEAX . . . . .	48	SUBA . . . . .	59
LFAY . . . . .	40	SUHB . . . . .	59
LSR . . . . .	61	SUBO . . . . .	59
LSRA . . . . .	62	SWI . . . . .	71
LSRB . . . . .	62	SWI2 . . . . .	72
		SWI3 . . . . .	72
MUL . . . . .	58	SYNC . . . . .	71
NEG . . . . .	56	TFR . . . . .	50
NEGA . . . . .	56	TST . . . . .	54
NEGD . . . . .	56	TSTA . . . . .	54
NOP . . . . .	64	TSTB . . . . .	54

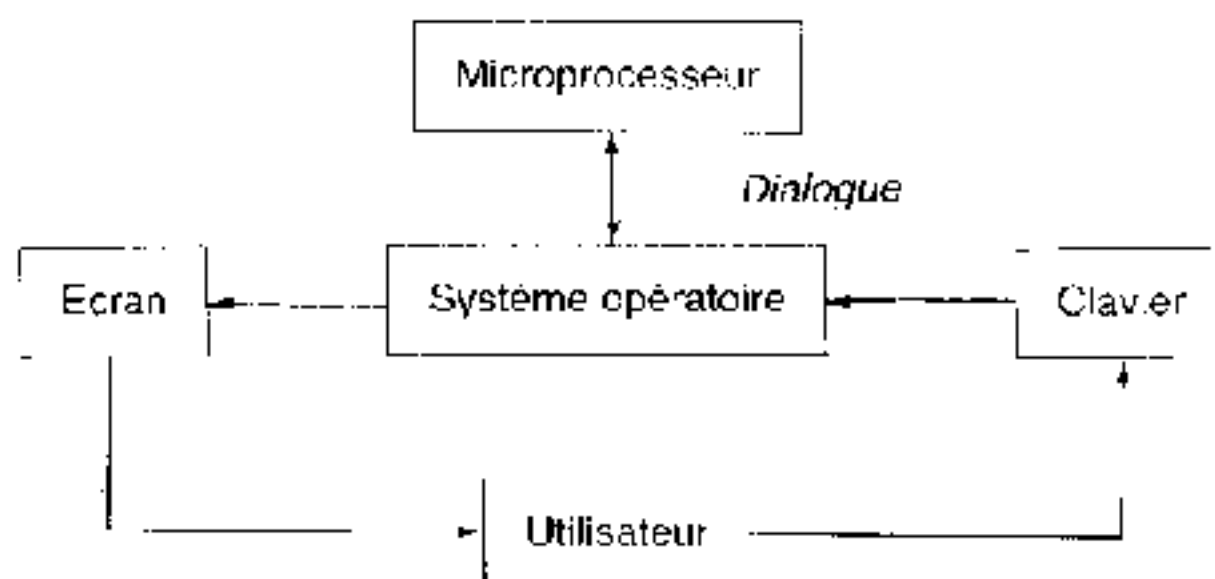
## 1

# Basic ou langage machine ?

## PRÉSENTATION GÉNÉRALE

Ce livre se voulant une introduction au langage machine et à la programmation en assembleur sur MO5, il est tout à fait possible que vous n'ayez aucune idée sur ce qu'est ce mode de programmation.

A vrai dire, la différence entre langage machine et langage assembleur, ou leurs spécificités par rapport au basic, peuvent vous être totalement étrangères. Alors, observons ensemble le fonctionnement simplifié de l'ordinateur.



Ce diagramme montre bien la barrière existant entre l'utilisateur et le microprocesseur appelé souvent unité centrale et noté CPU (de l'anglais, *Central Processing Unit*).

Dans le MO5, ce CPU est un 6809E et nous allons voir que ce circuit ne peut comprendre qu'un langage bien particulier et difficilement déchiffrable par un humain.

Quand vous communiquez avec votre MO5, vous lui transmettez les informations par l'intermédiaire du clavier. Ces informations sont alors codées par le système opératoire et traduites sous forme d'impulsions électriques. C'est seulement à partir de ce stade que l'information parvient au CPU et qu'il peut la comprendre. En fait, vos informations sont traduites par le système opératoire pour être compréhensibles par la machine. Ce traducteur est appelé « interpréteur BASIC ».

Voyons maintenant comment les impulsions électriques parviennent au CPU.

Sur la totalité des broches du circuit, huit servent à cet effet. Il ne peut donc accepter que huit signaux simultanément. Lorsqu'on combine ces signaux, on obtient un code machine. Par convention, on note la présence de 5 volts sur une broche par un 1, et celle de 0 volt par un 0. On obtient ainsi soit des 0, soit des 1 ; nous sommes par conséquent en base 2 ou encore en binaire. Ces chiffres sont appelés des bits (de l'anglais, *Binary Digits*) et sont l'unité fondamentale d'information. Associés par huit, on nomme le groupe octet.

Comme on le voit donc, une information parvenant au CPU est de la forme 1000 0110. 1000 0110 est un octet, association de huit bits. Si nous comparons par exemple l'opération langage machine, 10000110 11111111, et celle du BASIC, LET A = 255, on observe une légère différence et un côté plus rébarbatif du groupe d'octets car comme l'eût dit ce bon seigneur de LA PALICE, le langage machine est un langage pour... es machines !

Mais trêve de plaisanterie, car vous devez vous poser la question suivante : « Pourquoi se tourmenter avec tous ces chiffres alors que l'on peut utiliser des langages évolués comme le BASIC, le FORTH... ? »

Essayons de vous répondre par une comparaison entre un langage évolué, en l'occurrence le BASIC, et un langage de bas niveau, dans le cas présent, le langage machine.

## INCONVÉNIENTS DU LANGAGE MACHINE

- Impossibilité d'adaptation à d'autres ordinateurs : la portabilité du BASIC pose déjà certains problèmes suivant son type. Celle du langage machine est pratiquement insolvable car un programme est spécifique à un système (il ne suffit pas d'avoir le même microprocesseur).
- Programmes difficiles à lire et à corriger : une suite de nombres n'est pas très expressive, même s'il existe une équivalence avec un langage dit d'assemblage (voir ci-après). De plus, le repérage de l'erreur est assez critique.
- Calculs arithmétiques difficiles : la gestion de nombres en virgule flottante est très compliquée. Si l'on en a vraiment besoin, il vaut mieux faire appel à un langage évolué et spécialisé comme le PASCAL, par exemple.

- **Nécessité d'un apprentissage plus important** : l'apprentissage du BASIC se fait en quelques semaines, celui du langage machine est beaucoup plus long car il demande beaucoup de rigueur et d'attention.
- **Structuration difficile d'un programme** : il n'est pas question par exemple de rajouter des instructions à l'intérieur, si l'on n'a pas prévu de place à cet effet (sauf si l'on possède un assembleur, voir ci-après). Tout doit être pensé, sinon le travail de modification est très important.

## AVANTAGES DU LANGAGE MACHINE

- **Vitesse d'exécution maximale** : beaucoup de temps est perdu par l'interprétation d'un ordre BASIC. Les différences sont flagrantes entre les deux langages et l'on peut gagner jusqu'à cent fois en temps d'exécution.
- **Utilisation plus rationnelle de la taille mémoire** : un programme écrit en langage machine occupe moins de place en mémoire et permet ainsi de faire des programmes très puissants avec 32 K.
- **Possibilité de création de fonctions irréalisables en BASIC** : de nouvelles commandes peuvent être inventées à la guise du programmeur. Les seules limites sont pratiquement celles de votre imagination.
- **Utilisation de toutes les ressources de l'ordinateur** : le MO5 a ces possibilités pratiquement inexploitées en BASIC.
- **Possibilité de mêler BASIC et langage machine** : un programme ne nécessite pas toujours une rapidité absolue dans toutes ses parties. Seules quelques-unes peuvent être écrites en langage machine et utilisées sous forme de sous-programmes, les autres seront en BASIC.

## DIFFÉRENCES FONDAMENTALES

- **Pas de variables au sens du BASIC** : Il n'existe dans le microprocesseur que des registres internes (voir *Chapitre 3*). Ils ne correspondent pas vraiment aux variables BASIC proprement dites.
- **Pas de lignes** : en langage machine les instructions se suivent selon leurs adresses mémoires, alors que les lignes se repèrent selon leurs numéros. Ainsi, on peut par exemple allonger ou raccourcir une ligne, ou même en ajouter une : le réajustement est automatique alors qu'il n'en est pas question en langage machine (à moins de posséder un assembleur, voir plus loin).
- **Pas de correction de syntaxe** : si une erreur est faite en langage machine, l'ordinateur a toute chance de se « planter » et il ne reste plus que la solution de débrancher l'alimentation.

## LANGAGE D'ASSEMBLAGE

La seule différence avec le langage machine est la plus grande facilité de compréhension pour un humain. De ce fait, le langage assembleur n'est pas réellement compréhensible par la machine. En fait, il est une adaptation en un langage plus lisible qu'une suite d'octets.

Mais attention, ce n'est qu'une traduction, au même titre que celle d'un langage évolué, de chaque code sous forme d'abréviations appelées MNEMONIQUES. Par abus de langage, on dit que c'est équivalent.

Par ce procédé, on traduit par exemple 10000110 11111111 par LDA #\$FF. Ce qui signifie « mettre \$FF dans le registre A » (LD étant l'abréviation de **LOAD**, charger en anglais).

## CONCLUSION

Quand vous concevrez un programme, vous le ferez sous forme de MNEMONIQUES que vous traduirez ensuite en code machine, soit à l'aide de tables, soit à l'aide d'un programme appelé **ASSEMBLER**, tel le logiciel **Moniteur Assembleur Désassembleur ODIN**, commercialisé par **LORICIELS**.

Tout ceci est compliqué, long, décourageant, rébarbatif, mais vous serez très satisfaits de vos réalisations. soyez-en sûrs !

# 2

## Représentation de l'information

Nous avons vu au chapitre précédent, que l'information est codée sous forme de nombres appelés octets et que chaque octet comprend huit bits. Lorsque l'on parle d'information, il faut différencier d'une part la représentation des instructions, et d'autre part la représentation des données : les premières comportent un ou plusieurs octets spécifiques à chaque instruction, c'est ce qui représente le jeu d'instructions que nous verrons plus loin ; les secondes ont une représentation beaucoup plus compliquée pour laquelle nous devons distinguer plusieurs cas.

### REPRÉSENTATION EN BINAIRE SIMPLE

Un nombre est simplement représenté par sa valeur en base deux. Nous allons d'abord voir la logique de la représentation d'un nombre en décimal. Dans ce système, chaque chiffre représente, en allant de la droite vers la gauche, le chiffre des unités, des dizaines, des centaines, des milliers...

Chaque chiffre d'un nombre exprimé en base dix vaut entre zéro et neuf fois une puissance de dix correspondant à son rang.

Ainsi 6809 représente :

$$\begin{array}{rclclclcl}
 6 \times 10^3 & + & 8 \times 10^2 & + & 0 \times 10^1 & + & 9 \times 10^0 \\
 = 6000 & + & 800 & + & 00 & + & 9 \\
 = 6809
 \end{array}$$

Chaque chiffre représente en base dix une puissance de dix, en binaire (base deux) il représente une puissance de deux



Ainsi 10101110 représente :

$$\begin{aligned}
 & 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 = & 128 + 0 + 32 + 0 + 8 + 4 + 2 + 0 \\
 = & 174 \text{ en décimal}
 \end{aligned}$$

valeur correspondant à chaque bit.

N° bit	7	6	5	4	3	2	1	0
Valeur	128	64	32	16	8	4	2	1

On appelle bit de plus fort poids (plus significatif) le bit n° 7 et bit de plus faible poids (moins significatif) le bit n° 0, le poids étant croissant de la droite vers la gauche. On peut ainsi représenter sur huit bits un nombre décimal compris entre 0 et 255. Cependant, si on veut coder un nombre plus grand, on utilise alors deux octets. Il y a un octet de poids fort et un octet de poids faible, le premier est le prolongement (à gauche) du second.

N° bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Valeur	$2^{15}$	$2^{14}$	$2^{13}$	$2^{12}$	$2^{11}$	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

La valeur d'un nombre codé sur deux octets est aussi égale à la valeur de l'octet de poids faible ajouté à la valeur de l'octet de poids fort préalablement multiplié par 256.

$$\begin{aligned}
 \text{Exemple :} \quad & 11110100 \quad \quad \quad 00101101 \\
 = & 244 \times 256 + 45 \\
 = & 62509
 \end{aligned}$$

## REPRÉSENTATION HEXADÉCIMALE

Le binaire, comme on a pu s'en rendre compte, n'est pas des plus commode à utiliser ; on a donc choisi une notation équivalente mais mieux adaptée à la programmation.

Le décimal a été mis de côté car il n'est pas pratique de l'utiliser conjointement avec le binaire. On utilise donc l'hexadécimal, système de numérotation de base seize. Les chiffres hexadécimaux sont les nombres de 0 à 9 et les lettres de A à F. De plus, chaque chiffre hexadécimal représente un quartet (ensemble de quatre bits) binaire, d'où une très grande facilité de conversion.

L'hexadécimal a donc l'avantage de se convertir très facilement et de représenter un octet sur deux chiffres. C'est d'ailleurs pour cela que ce système de numérotation est devenu une convention en micro-informatique.

Binaire	Hexadécimal	Décimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

A titre d'exemple, essayons quelques conversions.

— conversion 10010010 en hexadécimal

1001	0010
—	—
9	2

soit 92 en hexadécimal.

— conversion 73 hexadécimal en binaire

7	3
—	—
0111	0011

soit 01110011 en binaire.

Un nombre sur deux octets sera codé de la même façon mais sur quatre chiffres hexadécimaux

— conversion 11110100 00101101 en hexadécimal

1111	0100	0010	1101
—	—	—	—
F	4	2	D

soit F42D en hexadécimal.

— conversion CB94 en binaire

C	B	9	4
—	—	—	—
1100	1011	1001	0100

soit 1100101110010100 en binaire.

## REPRÉSENTATION EN BINAIRE SIGNÉ

Nous avons vu que par les représentations précédentes, on ne peut coder que les entiers naturels. La représentation que voici permet de coder les entiers relatifs. Dans ce mode, le signe est donné par le bit de plus fort poids (MSB). On utilise un 0 pour un nombre positif et un 1 pour un nombre négatif.

Ainsi :

01001001 représente + 73;

11001001 représente - 73.

L'utilisation du bit n° 7 pour signe ne permet plus de coder que les nombres de valeur absolue inférieure à 128.

Cette représentation n'est cependant pas idéale car si l'on cherche à additionner un nombre négatif et un nombre positif, le négatif étant plus grand en valeur absolue (à cause de son bit n° 7 qui vaut 1) le résultat va être erroné. C'est pourquoi l'on a introduit une autre représentation.

## REPRÉSENTATION EN COMPLÉMENT À DEUX

Avec ce système, les nombres positifs ont la même représentation qu'en binaire signé, la différence est visible pour les nombres négatifs. La représentation d'un nombre négatif se fait en complétant ce nombre (c'est-à-dire en inversant chaque bit 1 en 0 et 0 en 1) et en ajoutant 1.

Exemple : représentation de - 17 en complément à deux

17 = 00010001

On complémente → 11101110

On ajoute 1 → 11101111

Nous admettons ici sans démonstration mathématique que la représentation en complément à deux permet de faire des opérations arithmétiques sur les octets et ce en utilisant les règles habituelles que nous allons rappeler :

$$\begin{array}{rclcl}
 0 & + & 0 & = & 0 \\
 1 & + & 0 & = & 1 \\
 0 & + & 1 & = & 1 \\
 1 & + & 1 & = & (1) 0
 \end{array}$$

(1) étant la retenue.

Exemples de calcul en complément à deux :

$$\begin{array}{r}
 \text{ajoutons } + 8 \text{ à } - 10 \\
 \begin{array}{r}
 00001000 \\
 + 11101110 \\
 \hline
 11111110
 \end{array}
 \end{array}$$

soit - 2

ajoutons + 17 à - 6

$$\begin{array}{r} 00010001 \\ + 1111010 \\ \hline (1) \quad 00001011 \quad \text{soit } + 11 \end{array}$$

On peut noter ici que l'on n'a pas à tenir compte de la retenue, ce qui peut être très pratique.

## REPRÉSENTATION EN DCB

Le principe réside sur le codage de chaque chiffre décimal séparément et ce sur quatre bits. On a donc besoin de  $N \times$  quatre bits pour coder en DCB un nombre de  $N$  chiffres.

Un quartet (quatre bits) permet le codage de seize combinaisons ; n'en ayant besoin que de dix, six sont inutilisées. On peut mettre de ce fait deux chiffres dans un octet, ce qui permet un codage des nombres de 0 à 99 sur un octet.

Code	DCB
0000 .....	0
0001 .....	1
0010 .....	2
0011 .....	3
0100 .....	4
0101 .....	5
0110 .....	6
0111 .....	7
1000 .....	8
1001 .....	9
1010	} Inutilisés
1011	
1100	
1101	
1110	
1111	

Exemple : le nombre 01110011 se lit

$$\begin{array}{cc} \underbrace{0111} & \underbrace{0011} \\ 7 & 3 \end{array}$$

soit 73 en décimal.

## REPRÉSENTATION EN VIRGULE FLOTTANTE

Avec les systèmes de codage précédents, il n'est pas possible de coder les décimaux relatifs. Nous allons à présent aborder un procédé qui le permet.

Voyons tout d'abord la logique de codage d'un nombre décimal à virgule : par exemple 12,31. Nous avons vu que le chiffre des unités correspond au rang 0 :  $10^0 = 1$ , celui des dizaines au rang 1 :  $10^1 = 10$  et ainsi de suite. Mais il faut savoir aussi que le chiffre des dixièmes correspond au rang  $-1$  :  $10^{-1} = 0,1$ , que celui des centièmes correspond au rang  $-2$  :  $10^{-2} = 0,01$  et ainsi de suite.

Ainsi 12,31 est égal à :

$$\begin{array}{rccccccc}
 1 \times 10^1 & + & 2 \times 10^0 & - & 3 \times 10^{-1} & + & 1 \times 10^{-2} \\
 = & 10 & + & 2 & - & 0,3 & + & 0,01 \\
 = & 12,31
 \end{array}$$

Voyons maintenant le principe de codage avec mantisse et exposant.

*Exemple : 101325,12.*

On peut le normaliser par  $0,10132512 \times 10^6$ .

On appelle alors 10132512 la mantisse, et 6 l'exposant.

En binaire, on utilise exactement les mêmes principes. Ainsi 101011,101 à convertir en décimal donne :

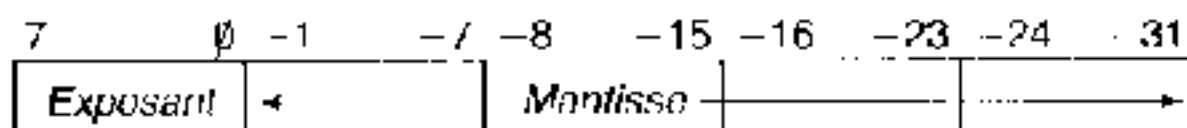
$$\begin{array}{l}
 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 - 1 \times 2^1 - 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} - 1 \times 2^{-3} \\
 = 32 + 0 + 8 + 0 - 2 - 1 + 0,5 + 0 - 0,125 \\
 = 43,625
 \end{array}$$

De la même façon, on peut mettre ce nombre sous forme mantisse — exposant :

$$101011,101 = 0,101011101 \times 2^6$$

101011101 est la mantisse, 6 l'exposant.

Regardons comment le MO5 code un nombre en virgule flottante. Ce codage se fait sur cinq octets : quatre sont réservés à la mantisse, un à l'exposant.



Le codage est cependant assez complexe.

L'exposant est représenté complémenté à deux avec le bit 7 à 0 s'il est positif et le bit 7 à 1 s'il est négatif.

Pour la mantisse, le même phénomène est observé. Lorsque le nombre est positif, le bit  $n-1$  est mis à 0 (alors qu'il est interprété pour les calculs comme s'il était à 1). Lorsqu'il est négatif, il est mis à 1. C'est une sorte de bit de signe comme en binaire signé mais à la différence qu'il est toujours pris égal à 1 pour la conversion.

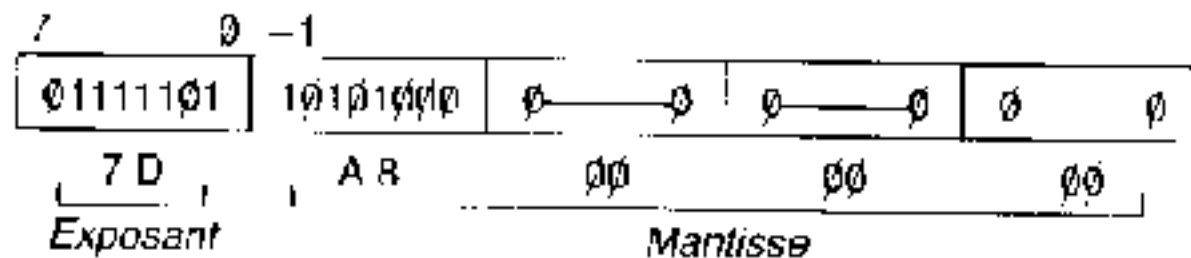
Codons par exemple  $-0,00010101$  :

On le normalise en  $0,10101 \times 2^{-2}$ .

Mantisse : 10101.

Exposant : 1111101  $\rightarrow$  0111101 car l'exposant est négatif.

d'où le codage :



On note le maintien du bit n° 31 à 1 car la mantisse est négative et l'inversion du bit n° 7 de l'exposant, lui aussi négatif.

Le codage en virgule flottante apparaît donc comme très compliqué et les routines qui permettent de traiter les nombres en virgule flottante le sont encore plus. C'est pour cela qu'en général on préfère choisir un langage évolué pour toute application numérique, tout au moins au niveau de l'amateur.

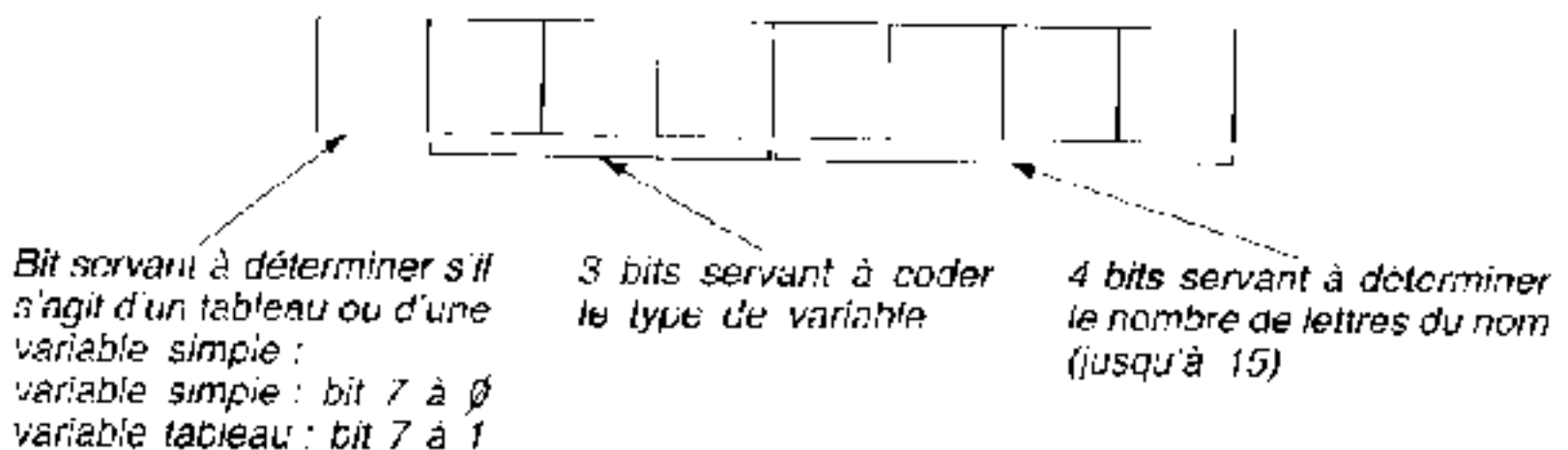
## REPRÉSENTATION DES CARACTÈRES ALPHANUMÉRIQUES

Le MO5 fonctionne sur le standard ASCII, standard le plus répandu dans le domaine informatique. Disons seulement que ce système permet de coder les vingt-six lettres de l'alphabet (en majuscule d'une part et en minuscule d'autre part), les dix chiffres et vingt caractères spéciaux, en tout sur sept bits. Nous donnons en annexe ce standard ASCII.

## REPRÉSENTATION DES VARIABLES

### ■ Codage du nom de la variable

Ce codage s'effectue sur  $n + 1$  octets :  $n$  pour le nombre de la variable et 1, octet de codage. En premier se trouve l'octet de codage.



Codage du type de variable :

Variable	Exemple	Type
Entière	A %	2
Réelle	A	4
Chaîne	A \$	3

Ensuite, vient le nom de la variable : celui-ci est codé en ASCII standard.

## ■ Codage de la variable

### i Variable numérique entière

La valeur est codée en complément à deux sur deux octets soit une plage de  $-32768$  à  $+32767$ . Le premier octet est l'octet de poids fort, le second celui de poids faible.

Cas des tableaux : on a successivement  $n + 1$  octets (pour le codage et le nom), deux octets pour le nombre d'octets du tableau, un octet pour le nombre de dimensions,  $p$  octets pour le nombre d'indices de chaque dimension.

### ii Variable numérique en virgule flottante

La valeur est codée en virgule flottante. Une variable réelle occupe donc  $1 + n + 5$  octets : 1 pour le codage,  $n$  pour le nom et 5 pour la valeur (1 pour l'exposant et 4 pour la mantisse).

Cas des tableaux : identique au procédé décrit pour les variables entières sauf que la variable est codée sur cinq octets au lieu de deux.

### i Variable alphanumérique

À la suite du nom de la variable, nous trouvons un octet pour la longueur de la variable, puis deux octets pour son adresse en mémoire. Le cas des tableaux est identique à celui des variables entières sauf pour le codage de la variable elle-même qui se fait alors en ASCII standard.

## ORGANISATION D'UNE LIGNE BASIC

Les lignes BASIC sont codées d'une certaine manière. Nous expliquons ici leur structure. Ceci va vous permettre, à l'aide de POKE et PEEK de faire ce que vous voulez dans une ligne BASIC. Par exemple, vous pouvez bâtir une ligne de plus de 256 octets (valeur maximale à l'origine fixée par la taille du buffer d'entrée) ou bien attacher des REM en couleur !

Dans le MO5, il existe deux variables systèmes qui définissent le début et la fin du programme BASIC présent en mémoire. Ces variables se trouvent respectivement aux adresses 2113 et 2115 (dans la page zéro du BASIC).

Initialisez l'ordinateur, puis tapez :

```
10 REM COUCOU
20 REM BONJOUR
30 FOR I = 9636 TO 9688 : ?PEEK (I) ; : NEXT
```

(9636 est l'adresse que l'on obtient en tapant ?PEEK (&H2113)\*256 + PEEK(&H2114))

Faites RUN : vous voyez alors apparaître une suite de chiffres qui correspondent aux codes des instructions que vous venez de rentrer.

- Les deux premiers octets donnent l'adresse en mémoire de la ligne suivante : ici, cette adresse est donc  $37 * 256 + 176 = 9648$ .
- Les deux octets suivants donnent le numéro de la ligne : ainsi, le numéro de notre première ligne est :  $0 * 256 + 10 = 10$ .
- Les octets suivants codent les instructions jusqu'à ce que l'on trouve un 0 qui indique la fin de cette ligne, ici le 140 est le code du REM, les six octets suivants sont les codes ASCII des lettres du mot « COUCOU ».

Maintenant que vous connaissez ce principe de codage (excepté celui des mots du BASIC que nous vous connaissons d'ici quelques lignes), vous pouvez travailler à votre guise sur les lignes BASIC ; ainsi, vous pouvez « cacher » une ligne en déplaçant l'adresse des deux premiers octets de la ligne précédente. Vous pouvez également faire des lignes aussi longues que vous voulez (ceci empêchant la modification directe de ces lignes puisqu'elles ne peuvent être rééditées). Ceci peut aussi vous permettre de réaliser des REM en couleur par le système des attributs décrit dans le chapitre « Écran ».

Réinitialisez le MO5. Tapez :

```
10 REM***COUCOU***
20 REM*COUCOU*
```

Puis tapez :

```
POKE 9641,27 : POKE 9642,&H41 : POKE 9651,27
POKE 9652,&H44
```

Faites LIST : le COUCOU de la REM 10 apparaît en rouge alors que celui de la REM 20 reste en bleu. (Le codage du rouge par le caractère d'échappement est expliqué au chapitre « Écran »)

Nous vous donnons maintenant les codes de toutes les fonctions du BASIC avec aussi leur adresse en ROM. Si vous vous intéressez à l'intérieur de la machine, il vous suffit de désassembler la ROM à partir de l'adresse que nous vous fournissons à l'aide du logiciel Moniteur Assembleur-Désassembleur, ODIN.



Code (décimal)	Code (hexa- décimal)	Fonction	Adresse
128	80	END	C49E
129	81	FOR	CF97
130	82	NEXT	D05C
131	83	DATA	C5E2
132	84	DIM	CA61
133	85	READ	DD3B
134	86	—	—
135	87	GO	C55F
136	88	RUN	C54B
137	89	IF	C5EA
138	8A	RESTORE	C48A
139	8B	RETURN	C5C1
140	8C	REM	C5E5
141	8D		C5E5
142	8E	STOP	C4A7
143	8F	ELSE	C5E5
144	90	TRON	CF1F
145	91	TROFF	CF20
146	92	DEFSTR	CEE C
147	93	DEFINT	CEE F
148	94	DEFSNG	CEE 9
149	95	—	—
150	96	ON	C61 F
151	97	TUNF	F900
152	98	ERROR	CF24
153	99	RESUME	CF2D
154	9A	AUTO	2236
155	9B	DELETE	CEC5
156	9C	LOCATE	E5FE
157	9D	CLS	E8FC
158	9E	CONSOLE	E640
159	9F	PSFT	E793
160	A0	MOTOR	EC6C
161	A1	SKIPF	EBCF
162	A2	EXEC	F8F9
163	A3	BEEP	E8FA
164	A4	COLOR	E6B3
165	A5	LINE	E81B
166	A6	BOX	E82C
167	A7	—	—
168	A8	ATTRB	E61C
169	A9	DEF	E86B
170	AA	POKE	CCDD

Code (décimal)	Code (hexa- décimal)	Fonction	Adresse
171	AB	PRINT	CD7A
172	AC	CONT	C4CF
173	AD	LIST	E06A
174	AE	CLEAR	C1DE
175	AF	DOS	F000
176	D0	—	—
177	B1	NEW	C422
178	B2	SAVF	F0A3
179	B3	LOAD	E28C
180	B4	MERGE	E258
181	B5	OPEN	E1E9
182	B6	CLOSE	E1B8
183	B7	INPEN	E8C3
184	B8	PEN	21F9
185	B9	PLAY	LF2D
186	BA	TAB <	CD4A
187	BB	TO	GFC9
188	BC	SUB	C5FA
189	BD	FN	3761
190	BE	SPC <	CE12
191	BF	USING	D0E2
192	C0	USR	37C8
193	C1	ERL	C8DA
194	C2	ERR	C6D6
195	C3	OFF	FC1F
196	C4	THEN	C600
197	C5	NOT	C71F
198	C6	STFP	GFF9
199	C7	+	D2AC
200	C8	—	D29F
201	C9	*	D2EE
202	CA	/	D51C
203	CB	↑	D6B4
204	CC	AND	C886
205	CD	OR	C88C
206	CE	XOR	C892
207	CF	EQV	C89D
208	D0	IMP	C898
209	D1	MOD	D377
210	D2	EQ	D34A
211	D3	>	C87F
212	D4	—	C865
213	D5	<	C86D

Les codes suivants (de 214 à 255) ne correspondent pas à des fonctions du BASIC. En revanche, d'autres fonctions du BASIC (les manquantes) sont codées sur deux octets. Nous donnons le détail de ces codes ci-après.

Précode (décimal)	Code (décimal)	Précode (hexa- décimal)	Code (hexa- décimal)	Fonction	Adresse
255	128	FF	80	SGN	D5D9
255	129	FF	81	INT	D6D8
255	130	FF	82	ABS	D59F
255	131	FF	83	FRE	CA6A
255	132	FF	84	SQR	D6AB
255	133	FF	85	LOG	D6F1
255	134	FF	86	EXP	D734
255	135	FF	87	COS	D7AE
255	136	FF	88	SIN	D7B5
255	137	FF	89	TAN	D80B
255	138	FF	8A	PFFK	CCD4
255	139	FF	8B	LEN	CC11
255	140	FF	8C	STRS	CB64
255	141	FF	8D	VAL	CB9F
255	142	FF	8E	ASC	CC0F
255	143	FF	8F	CHR\$	CC5B
255	144	FF	90	EOR	E3BE
255	145	FF	91	CINT	D627
255	148	FF	94	FIX	D5E0
255	149	FF	95	HFX\$	223C
255	151	FF	97	STICK	E89F
255	152	FF	98	STRIG	F6B1
255	153	FF	99	GR\$	CC52
255	154	FF	9A	LEFT\$	CC25
255	155	FF	9B	RIGHT\$	CC1E
255	156	FF	9C	MID\$	CC67
255	157	FF	9D	INSTR	CCE7
255	158	FF	9E	VARPTR	D0D9
255	159	FF	9F	RND	DB36
255	160	FF	A0	INKEY\$	E46E
255	161	FF	A1	INPUT	DD0A
255	162	FF	A2	CSRLIN	E469
255	163	FF	A3	POINT	F714
255	164	FF	A4	SCREEN	E6AA
255	165	FF	A5	POS	E436
255	166	FF	A6	PTRIG	E8AD

Il existe aussi d'autres fonctions du BASIC absentes dans ces tableaux telles que GOTO, GOSUB, INPUTEN. Leur codage est aisé à partir des tableaux précédents ; ainsi la fonction GOSUB se code sur deux octets, celui de GO : 135 suivi de celui de SUB : 188

## CONCLUSION

Ce chapitre est indispensable à assimiler car c'est la base de la programmation en langage machine. Nous vous conseillons même de le relire plusieurs fois et de vous attarder sur chaque détail qui ne vous paraît pas clair; en effet, il faut que ces notations vous deviennent tout à fait familières pour pouvoir programmer efficacement.

Seule la représentation des variables du MO5 n'est pas fondamentale à connaître car elle ne sert qu'à l'élaboration d'utilitaires. Cependant, nous pensons qu'elle n'est pas à négliger, car il est toujours intéressant d'approfondir la gestion interne d'un système.

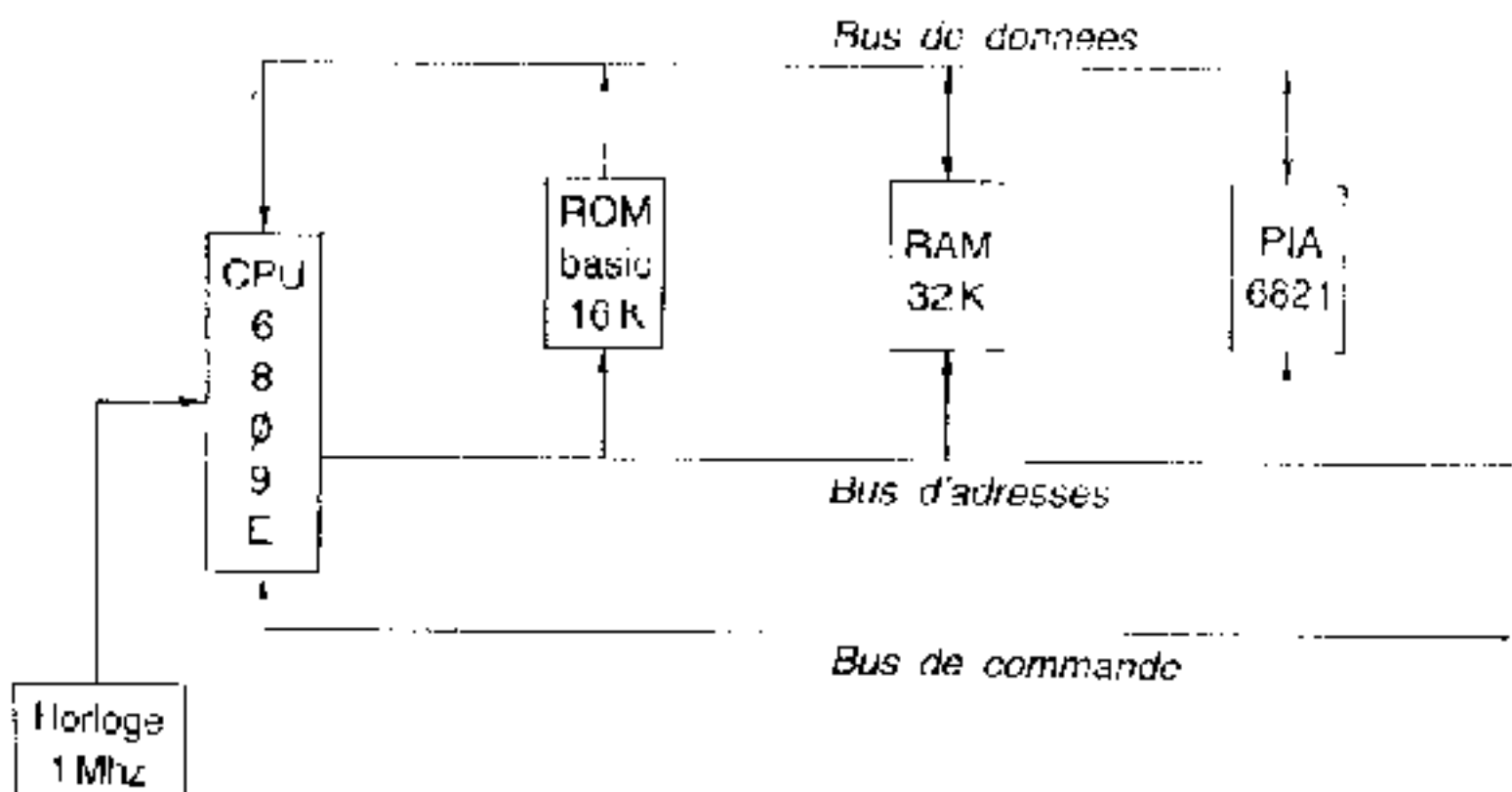
Si vous nous avez lu jusqu'à là et que vous pensez être « au point », alors passons sans plus attendre à une lecture moins rébarbative !

## 3

## Le système M05

Certains concepts de base vont être présentés dans ce chapitre, notamment les modes d'adressage et les registres internes. Nous allons aussi visualiser rapidement l'architecture interne du système et plus particulièrement celle du CPU.

M05



Comme on peut le voir sur ce synoptique, le microprocesseur 6809 est relié avec les autres organes de l'ordinateur et aussi à l'extérieur grâce à son PIA.

## LES LIAISONS

Le 6809 communique avec son entourage au moyen de trois bus.

### ■ Bus de données

Il est sur huit bits et achemine donc un octet à la fois. Il va permettre de transférer les données du CPU vers les autres boîtiers (mémoires et entrées-sorties) et inversement.

### ■ Bus d'adresses

Il est sur seize bits et va servir à transporter une adresse générée par le microprocesseur, adresse qui va sélectionner la source ou la destination des données qui transitent sur le bus destiné à cet effet.

Le nombre maximum d'octets adressables est 65536, soit 64K.

Comme on le voit sur le diagramme, le bus étant relié à tous les autres boîtiers, il permet de tous les sélectionner et donc de les relier physiquement au CPU.

### ■ Bus de commande

Il s'agit d'une ligne qui transporte les différents signaux de synchronisation nécessaires au système.

## L'HORLOGE

Autre partie du schéma, l'horloge fournit une base de temps au microprocesseur. Ce le fait au rythme de son quartz qui a une fréquence de 1Mhz, soit une période ou encore un cycle d'horloge de une micro-seconde ( $\mu$ s). Une instruction est caractérisée par son nombre de cycles duquel on déduit son temps de déroulement (temps que nous indiquerons lors de l'analyse de chaque instruction).

## LES MÉMOIRES ROM ET RAM, LE 6821

### ■ La ROM

C'est ce qu'on appelle la mémoire morte, c'est-à-dire que l'on peut juste lire les données qu'elle contient, et non pas en inscrire de nouvelles (ce que l'on voit sur le synoptique grâce à la flèche à un seul sens).

Son autre particularité est de rester figée même quand le système est hors-tension.

La ROM du MO5 fait 16K et contient le BASIC, le moniteur (gestion du système : écran, mémoire, périphériques...). Beaucoup de routines de la ROM sont utilisables en langage machine ; elles sont présentées dans la seconde partie de cet ouvrage.

## ■ La RAM

Elle fait 32K utilisateur. C'est ce qu'on appelle la mémoire vive ; on peut y lire ou y inscrire des données, mais elles sont effacées à la mise hors-tension.

Elle s'étend de \$2000 à \$9FFF, la place restante étant prise par les extensions et la ROM.

En ce qui concerne le stockage de programmes en langage machine, il faut savoir que le code machine peut être mis dans n'importe quel espace de la RAM libre pour l'utilisateur (voir carte mémoire).

Comme nous l'avons dit, il est possible de mêler BASIC et langage machine, cependant on doit prendre certaines précautions : le BASIC et le langage machine doivent être légèrement distants pour éviter tout chevauchement. La meilleure façon est de protéger le programme en langage machine par un CLEAR, &HXXX, \$XXX, \$XXX étant l'adresse du début du programme en langage machine.

L'appel à un programme en langage machine se fait par un EXEC &HXXX, \$XXX, \$XXX, \$XXX étant l'adresse d'exécution du programme.

## CARTE-MÉMOIRE MOS

Adresses (en hexadécimal)	Occupées par
0000 — 1F3F	Ecran (points ou couleur selon la valeur de \$ A7C0)
1F40 — 1FFF	Libre pour l'utilisateur
2000 — 20FF	Page zéro du moniteur
2100 — 21FF	Page zéro du BASIC
2200 — 9FFF	Mémoire disponible pour l'utilisateur
A000 — A7BF	Réservé pour le floppy
A7C0 — A7C3	Système 6821 (PIA)
A7C4 — A7CB	Libre
A7CC — A7CF	Réservé pour le 6821 Joystick
A7D0 — A7DF	Réservé pour contrôler le floppy
A7E0 — A7E7	6821 : interface parallèle
A7E8 — A7FF	Libre, réservé pour des extensions de PIA ou ACIA
A800 — AFFF	Libre (place — 2K)
B000 — BFFF	Réservé pour les cartouches enfichables
C000 — EFFF	BASIC (12K)
F000 — FFFF	Moniteur (4K)

La sauvegarde d'un programme en langage machine se fait par : SAVEM « Nom du programme », &HXXX, &HYYY, &HZZZ, \$XXX, \$XXX étant l'adresse de début du

programme, \$YYYY celle de fin du programme et \$7777 celle d'exécution du programme.

### ■ Le 6821

Il permet au MO5 de communiquer avec des organes périphériques au microprocesseur : clavier, magnétophone, écran, crayon optique...

Il est situé de \$A/C0 à \$A/C3

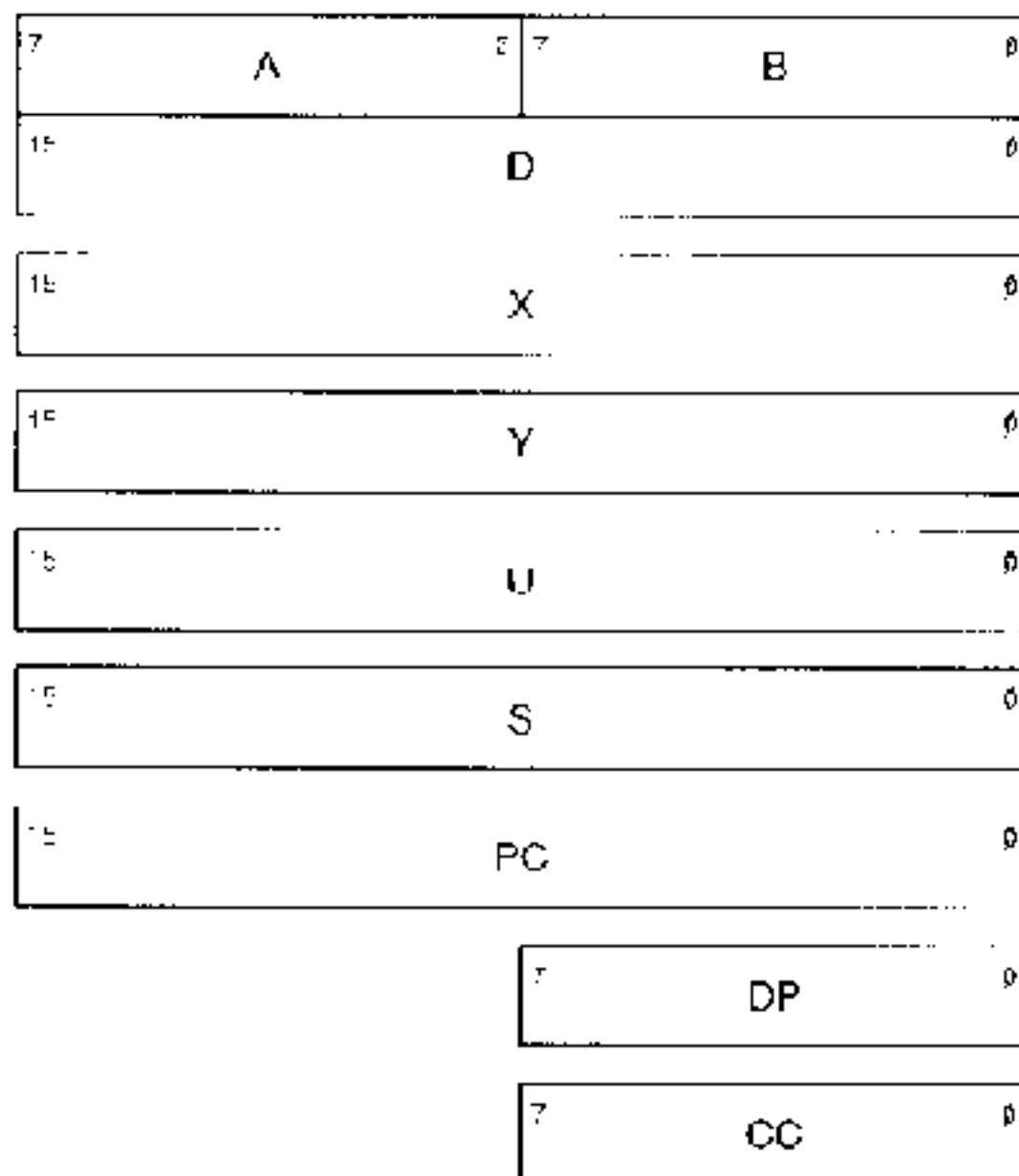
## LE MICROPROCESSEUR

On peut distinguer deux parties principales : l'unité de commande que nous n'aborderons de côté et l'unité arithmétique et logique comprenant les registres internes (UAL).

UAL comme son nom l'indique permet les opérations arithmétiques (addition et soustraction) et les opérations logiques (masques, rotations, décalages...). Pour effectuer ces calculs, le microprocesseur dispose de registres internes que nous allons étudier en détail.

### ■ Les registres internes

Ils sont au nombre de neuf et habituellement représentés de la façon suivante :





### 1) Les accumulateurs : A, B et D

A et B sont sur huit bits. Ils servent à accumuler des données comme leur nom l'indique. Ils servent donc lors de n'importe quelle opération arithmétique, lors de transferts ou d'analyses de données.

A sert plus spécialement lors de calculs en DCB (DAA), tandis que B est plutôt utilisé pour les indexations (ABX).

D est un registre seize bits, résultat de la concaténation de A et B, A est l'octet de poids fort, B celui de poids faible. D est utilisé au même titre que A et B.

### 2) Les registres d'index : X et Y

Sur seize bits tous les deux, ils servent le plus souvent comme registres d'indexation ou comme compteurs.

Ils peuvent aussi servir pour des stockages temporaires de données ou d'adresses. On utilise de préférence X, car les instructions concernant Y nécessitent un pré octet.

### 3) Les pointeurs de pile : S et U

S est le pointeur de la pile système, il sert à indexer la pile gérée par le 6809 en RAM.

U est le pointeur de la pile utilisateur qui permet au programmeur de sauver ou recharger des valeurs, sans pour autant altérer la pile système.

S et U pointent toujours sur le dernier octet empilé (voir plus loin « Le concept de pile »).

### 4) Le compteur ordinal : PC

Il sert au 6809 pour stocker l'adresse de la prochaine instruction à exécuter. Il est donc modifié à chaque instruction exécutée. PC est par conséquent sur seize bits.

### 5) Le registre de page directe : DP

Sur huit bits, ce registre permet de spécifier une page directe dont l'accès est facilité (voir « Les modes d'adressage »).

Attention à remettre toujours DP à \$21 lors d'un retour au BASIC.

### 6) Le registre d'état : CC

Ce registre permet à l'utilisateur de connaître l'état du microprocesseur. Chaque bit a une signification bien précise. Il est habituellement représenté comme suit :

E	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

- Le bit 0 (symbole C) est le bit de retenue (*carry* en anglais). Il indique s'il y a une retenue sur le bit de plus fort poids.

*Exemple : additions 197 et 153*

$$\begin{array}{r}
 197 = \% 11000101 \\
 153 = \% 10011001 \\
 \hline
 \begin{array}{r}
 11000101 \\
 + 10011001 \\
 \hline
 10\ 011110
 \end{array}
 \end{array}$$

Le résultat, étant sur huit bits, ne peut être que  $\% 01011110$  soit 94 et la retenue C est mise à 1.

Vérifions :  $197 + 153 = 94 + 256 = 350$

*Résultat / \ Retenue*

- Le bit 1 (symbole V) indique un résultat supérieur à ce qu'un octet donné peut représenter en binaire signé.
- Le bit 2 (symbole Z) est mis à 1 lorsque le résultat d'une opération est nul autrement il est mis à 0.

Le bit 3 (symbole N) a la valeur du bit de plus fort poids de l'octet considéré. En arithmétique binaire signé,  $N = 1$  indique un résultat négatif, tandis que  $N = 0$  indique un résultat positif. Si on ne travaille pas en arithmétique binaire signé, on peut se servir de N pour déterminer la valeur du bit n° 7.

- Le bit 4 (symbole I) est l'indication d'interruption IRQ. S'il est à 1 ces interruptions sont interdites, dans le cas contraire, elles sont autorisées.
- Le bit 5 (symbole H) est le bit de demi-retenue. Il indique une retenue du bit 3 sur le bit 4.
- Le bit 6 (symbole F) est l'indication d'interruption FIQR (identique à I).
- Le bit 7 (symbole E) indique l'état de la sauvegarde des registres lors d'une interruption :  $E = 0$  indique que seuls CC et PC sont empilés,  $E = 1$  indique que tous les registres sont sauvegardés (voir « Les interruptions »).

## LES MODES D'ADRESSAGE

Après avoir vu ensemble les registres du 6809, il nous faut maintenant aborder les modes d'adressage. On peut compter six modes d'adressage principaux.

### ■ Mode inherent

Toutes les instructions sont sur un octet. Le code opération spécifie implicitement les informations permettant la réalisation de l'instruction.

*Exemple : CLRA : mise de A à zéro  
ABX : addition de B et X*

### ■ Mode immédiat

Il faut distinguer d'une part les fonctions d'échange, de transfert et d'empilement et toutes les autres utilisant le même mode.

Les premières sont sur deux octets, le second octet spécifiant les registres.

Les secondes sont sur deux octets ou plus. Dans ce cas la donnée nécessaire à l'exécution de la fonction est présente dans l'octet qui suit la fonction.

*Exemple :* LDA #\$FF charge la valeur \$FF dans l'accumulateur.

Le # indique que le mode est immédiat.

### ■ Mode étendu

Ce mode permet d'effectuer des opérations sur des données en mémoire. En effet, les octets qui suivent le code opérateur représentent l'adresse mémoire de l'opérande. La syntaxe assembleur 6809 est soit rien, soit un « > » suivi de l'adresse.

*Exemple :* LDA \$C000 ou LDA >\$C000 charge dans A la valeur qui se trouve en \$C000.

### ■ Mode page directe

Ce mode est identique au mode étendu, sauf que l'octet de poids fort de l'adresse n'est pas spécifié et sera égal à DP.

On utilise ce mode pour gagner de la place mémoire (l'instruction comporte un octet de moins).

La syntaxe assembleur est soit « / », soit « < » suivi de l'octet de poids faible de l'adresse.

*Exemple :* LDA /\$F0 ou LDA <\$F0 avec DP = \$20 charge dans A la valeur qui se trouve en \$20F0.

**Note :** à la mise sous tension DP est à \$21 et doit être remis à jour lors d'un retour au n431C.

### ■ Mode relatif

Ce mode est utilisé par tous les branchements conditionnels et certains branchements inconditionnels. Il y a d'une part les branchements courts dont l'opérande est sur un octet, et d'autre part les branchements longs pour lesquels l'opérande est sur deux octets.

Dans ce mode l'opérande sera ajoutée ou retranchée (en complément à deux, suivant que l'opérande est positive ou négative) au compteur ordinal PC (rappel : PC pointe sur la prochaine instruction à exécuter).

Un branchement court permet un déplacement de - 128 à + 127 octets, tandis qu'un branchement long permet un déplacement de - 32768 à + 32767 par rapport à PC.

## ■ Mode indexé

C'est le mode le plus puissant sur 6809 puisqu'il permet de multiples adressages.

Dans ce mode l'opérande est constituée d'une adresse de base contenue soit dans un registre d'index, soit dans un pointeur de pile, soit dans PC, et d'un déplacement qui est soit absolu, soit contenu dans A, B ou D.

La somme, en binaire signe, de l'adresse de base et du déplacement constitue l'adresse.

### 1. Adressage direct

Dans ce mode l'adresse est chargée directement dans l'accumulateur. Distinguons les divers adressages accessibles.

Déplacement absolu par rapport à X, Y, U, S :

- déplacement nul

L'adresse est contenue dans le registre, elle est en fait égale à l'adresse de base. La syntaxe assembleur est « ,registre ».

*Exemple* : LDB ,U charge B avec la valeur pointée par U.

- déplacement 5 bits

Le déplacement est codé sur cinq bits en complément à deux. Le bit 4 sert de bit de signe.

La syntaxe assembleur est « n,registre ».

*Exemple* : LDB 2,X charge B avec la valeur pointée par  $X + 2$ .

- déplacement 8 bits

Identique au précédent, mais le déplacement peut s'effectuer sur  $-128$  à  $+127$  octets.

*Exemple* : LDB \$60,X charge B avec la valeur pointée par  $X + \$60$ .

- déplacement 16 bits

Identique au précédent, mais le déplacement peut s'effectuer sur  $-32768$  à  $+32767$  octets.

*Exemple* : LDB \$1000,Y charge B avec la valeur pointée par  $Y + \$1000$ .

— Déplacement accumulateur par rapport à X, Y, U, S :

- déplacement par rapport à A

L'adresse est égale à A plus le registre en complément à deux. Le déplacement est donc  $-128$  à  $+127$  octets.

*Exemple* : LDB A,X charge B avec la valeur pointée par  $A + X$ .

- déplacement par rapport à B

Identique au précédent.

*Exemple :* LDA B,U charge A avec la valeur pointée par B + U.

- déplacement par rapport à D

Identique aux précédents, mais le déplacement peut s'effectuer de - 32768 à + 32767 octets.

*Exemple :* LDA D,Y charge A avec la valeur pointée par D + Y.

#### — Auto-incrémentation et auto-décrémentation du registre de base

- incrémentation simple

L'instruction s'effectue en utilisant le registre de base comme adresse, puis celui-ci est incrémenté de 1. La syntaxe est « ,registre+ ».

*Exemple :* LDA ,U+ charge A avec la valeur pointée par U et incrémente U de 1.

- incrémentation double

Identique au précédent, mais le registre de base est incrémenté de 2.

La syntaxe est « ,registre++ ».

*Exemple :* LDA ,U++ charge A avec la valeur pointée par U et incrémente U de 2.

- décrémentation simple

Le registre de base est d'abord décrémenté de 1, puis l'instruction s'effectue en utilisant la nouvelle valeur du registre de base.

La syntaxe est « ,--registre ».

*Exemple :* LDA ,--X décrémente X de 1 et charge A avec la valeur pointée par la nouvelle valeur de X.

- décrémentation double

Identique au précédent mais la décrémentation est de 2.

La syntaxe est « ,-- --registre ».

*Exemple :* LDA ,-- --X décrémente X de 2 et charge A avec la valeur pointée par la nouvelle valeur de X.

#### — Déplacement absolu par rapport à PC :

- déplacement 8 bits

Le déplacement est codé sur un octet en complément à deux. Ce déplacement est ajouté à PC en binaire signé (rappelons que PC pointe sur la prochaine instruction à exécuter).

La syntaxe assembleur est « n,PC ».

*Exemple :* LDB \$01, PC charge B avec la valeur pointée par PC + 1.

- déplacement 16 bits

Identique au précédent mais le déplacement se code sur deux octets.

*Exemple :* LDB \$1000,PC charge B avec la valeur pointée par B + \$1000.

### Adressage indirect

Dans ce mode, l'adresse définitive est obtenue par un chargement de l'adresse contenue dans un registre.

La syntaxe assembleur est identique à celle du mode direct à la différence que l'opérande est précédée et suivie de crochets.

*Exemple :* LDA [R,X] charge A avec la valeur qui a pour adresse, l'adresse pointée par B + X.

Tous les modes accessibles en direct, le sont en indirect à l'exception du déplacement 5 bits, de l'auto-incrémentation simple et de l'auto-décrémentation simple.

Par contre, il existe le mode indirect étendu.

#### — Indirect étendu

Comme son nom l'indique, cet adressage est un adressage étendu dont l'adresse est donnée en indirect.

La syntaxe assembleur est « [n] ».

*Exemple :* LDA [\$2000] charge A avec la valeur qui a pour adresse, l'adresse située en \$2000.

## LE CONCEPT DE PILE

Dans le paragraphe concernant les modes d'adressage, il a été supposé que l'adresse exacte, ou au moins une relation avec une adresse exacte, était connue pour tout adressage. Cependant, il est vrai que dans beaucoup de programmes, certaines applications nécessitent que le microprocesseur ne travaille pas avec des emplacements mémoires connus, mais plutôt avec un système de pile. Il s'agit d'un ensemble d'emplacements mémoires pour lesquels le CPU possède un générateur d'adresses spécifique qui permet l'accès à ces emplacements. Le générateur d'adresses utilise le concept d'empilement-dépilement.

L'exemple classique est le suivant : on vous distribue trois cartes : un as, un roi et un valet (l'ordre de distribution importe). Vous les empilez sur la table dans l'ordre où l'on vous les a données. Vous posez donc d'abord l'as, ensuite le roi et enfin le valet au sommet. On vous demande ensuite de prendre chaque carte une par une. La première carte dépilée va être le valet alors qu'il a été la dernière empilée. Pour l'as, cela va être l'inverse : il va être dépilé le dernier alors qu'il avait été empilé le premier.

La pile du microprocesseur a exactement le même principe que dans cet exemple. Les commandes disponibles sont nombreuses du fait de la possibilité d'indexation avec B ou X. Il est très important de se souvenir du concept premier-entré-dernier sorti.

La présence d'une pile système va faciliter :

- les retours après interruption,
- les retours de sous-programmes,
- le stockage de données temporaires.

La présence d'une pile utilisateur va faciliter le stockage de données temporaires ou la lecture de zones mémoires.

Nous allons maintenant nous attarder sur la gestion des adresses de chaque pile.

Le pointeur de pile (S ou U) repère l'adresse du sommet de la pile (exemple : LDA ,S charge le dernier octet empilé)

La pile commence toujours à l'adresse la plus haute et elle régresse, le pointeur étant décrémenté à chaque empilement

Cependant, tout ceci n'est pas indispensable à savoir car la gestion est automatique (à moins que l'on désire utiliser les indexations avec les pointeurs programmes). On a juste à se soucier de l'ordre dans lequel sont rentrés les éléments.

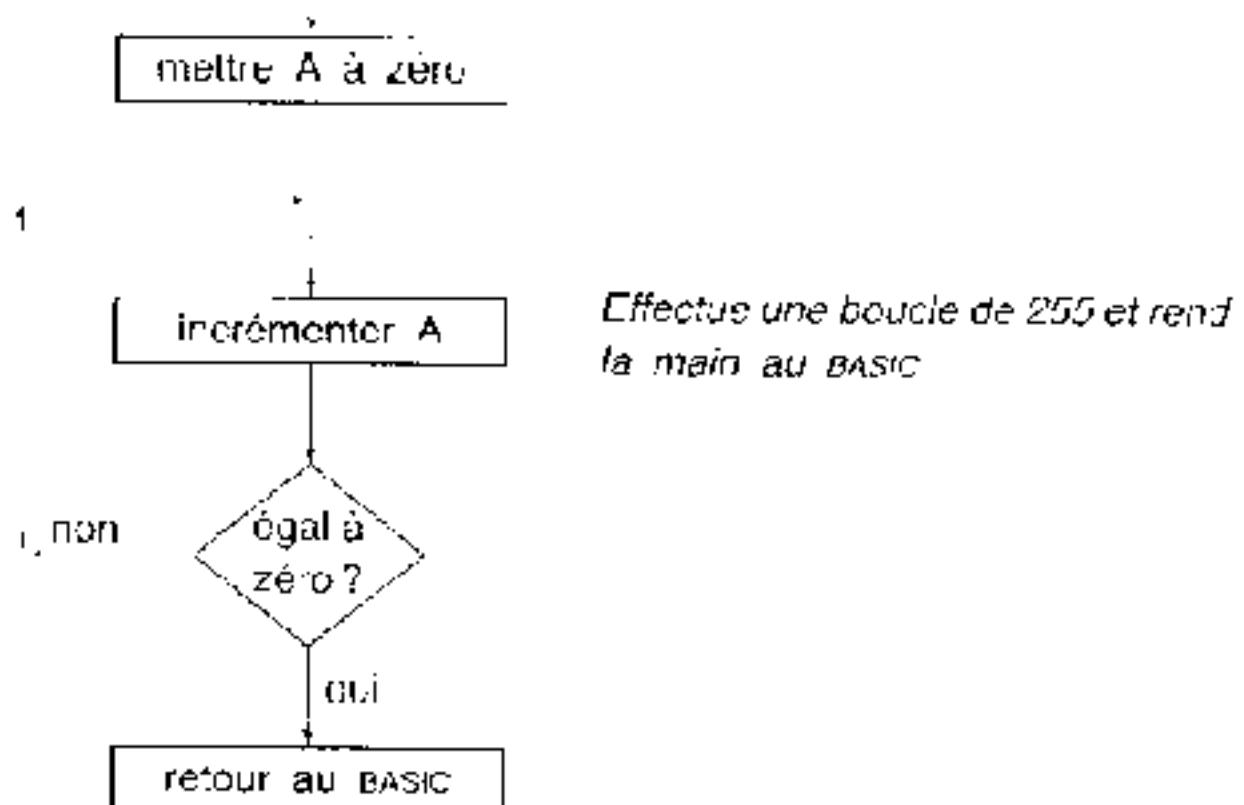
## LE CONCEPT D'ORGANIGRAMME

Pour pouvoir réussir un programme, il faut avoir conçu un algorithme, c'est-à-dire avoir analysé toutes les différentes phases qui permettent d'aboutir au résultat. Pour pouvoir concrétiser clairement cet algorithme, on utilise une représentation schématique appelée organigramme (ou encore ordiogramme).

Celui-ci comporte des rectangles pour représenter les actions (lecture, chargement...) et des losanges pour représenter les tests (si l'information est vraie, alors faire telle action, sinon faire tel autre).

L'organigramme permet la réalisation d'un programme bien construit et doit permettre à un autre utilisateur de le comprendre et de l'utiliser.

*Exemple :*



# 4

## Les interruptions

### INTRODUCTION

Les interruptions sont des signaux envoyés par un organe d'entrée-sortie du microprocesseur pour lui signaler qu'il a besoin de ses services. Le microprocesseur scrute à chaque cycle, c'est-à-dire toutes les micro-secondes, une éventuelle interruption, ce qui permet une réponse instantanée.

C'est un système très pratique, car le microprocesseur n'a pas besoin de scruter les périphériques. Il se contente d'attendre qu'on l'avertisse et poursuit son exécution.

### CARACTÉRISTIQUES

En réponse à une interruption, lorsque le processeur en détecte une et que celle-ci est autorisée, il termine avant tout l'instruction qu'il était en train d'exécuter. Il interdit ensuite les autres interruptions moins prioritaires et empile PC, CC et éventuellement les autres registres. Il se branche alors à l'adresse de la routine d'interruption. L'interruption étant terminée, il copie les registres empilés et poursuit le programme qui a été interrompu.

### INTERRUPTIONS DU 6809

#### ■ *Interruptions rapides FIRO*

Ce sont les interruptions les moins prioritaires, elles sont donc masquables, c'est-à-dire que suivant la valeur de l'indicateur F, elles auront lieu ou non.