Université Abdelhamid Ibn Badis de Mostaganem
Faculté des Sciences et de la Technologie
Département de Génie Électrique

Cours pour Licence Électronique L3

Systèmes à Microprocesseurs (SAM)

Auteur :
M. Mohamed Bentoumi

Version 1.1 du
11 janvier 2022

Table des matières

Sommaire **iv**

I Les systèmes de numération **1**

I.1 Introduction 1

I.2 Le système binaire 1

I.2.1 Le système Hexadécimal 2

I.3 Codage de l'information 3

I.3.1 Code DCB (Décimal codé binaire, BCD en anglais) 3

I.3.2 Représentation des caractères alphanumériques 3

I.3.3 L'arithmétique binaire 4

I.3.4 Représentation des nombres négatifs 5

II Historique et évolution des ordinateurs **9**

II.1 Histoire de l'ordinateur 9

II.1.1 Génération zéro : les calculateurs mécaniques (1642 – 1945) 9

II.1.2 La première génération : les tubes à vide (1945 – 1955) 10

II.1.3 La deuxième génération : les transistors (1955 – 1965) 10

II.1.4 La troisième génération : les circuits intégrés (1965 – 1980) 11

II.1.5 La quatrième génération : VLSI (Machines actuelles) 12

II.2 Architecture d'un système à microprocesseur 12

II.2.1 Notion de HARDWARE (Matériel) 12

II.2.2 Notion d'Entrées/Sorties 15

II.2.3 Notion de SOFTWARE (Logiciel) 16

III Les Mémoires à semi-conducteurs **19**

III.1 Introduction 19

III.2 Mémoires mortes 20

III.3 Mémoires vives : RAM 20

III.3.1 RAM Statique (SRAM) 20

III.3.2 RAM dynamique (DRAM) 21

III.4 Critères de choix d'une mémoire 21

III.5 Fonctionnement d'une mémoire 22

III.5.1 Schéma fonctionnel d'une mémoire 22

III.5.2 Organisation de la mémoire 22

III.6 Interfaçage microprocesseur/mémoire 23

III.6.1 Chronogrammes de lecture-écriture 23

III.7 Extension de la capacité mémoire 24

III.7.1 Connexion de plusieurs boîtiers mémoire sur le bus d'un μP 24

III.7.2 Le mapping de la mémoire 25

III.7.3 Décodage d'adresses 25

IV Architecture et fonctionnement d'un microprocesseur **27**

IV.1 Introduction 27

IV.2 Architecture interne d'un microprocesseur 28

IV.2.1 Représentation binaire 28

IV.2.2 Architecture interne d'un microprocesseur type 29

IV.2.3 L'unité de calcul 30

IV.2.4 L'unité de contrôle 31

IV.3 Fonctionnement d'un Microprocesseur 32

IV.3.1 Type d'architecture d'un microprocesseur 34

IV.4 Les processeurs Spécialisés 35

IV.4.1 Micro-contrôleurs (μC) 35

IV.4.2 Digital Signal Processor (DSP) 35

IV.4.3 Microprocesseurs sur mesure 36

V Etude d'un microprocesseur 8 bits : MC6809 de Motorola **37**

V.1 Introduction 37

V.2 Boitier et brochage 37

V.2.1 Minimum pour démarrer 40

V.2.2 Adressage de la mémoire 40

V.3 Architecture interne du 6809 41

V.3.1 Les registres internes du 6809 41

V.4 Langage assembleur 44

V.4.1 Langage machine 44

V.4.2 Langage assembleur 44

V.5 Le jeu d'instruction du 6809 45

V.5.1 Instructions de chargement de données 45

V.5.2 Les modes d'adressage du 6809 46

V.5.3 Affectation des indicateur 49

V.5.4 Calcul Arithmétique et Logique 49

V.6 Les directives en assembleur 53

V.7 Les instructions de comparaison et de branchement 54

V.7.1 Les instructions de comparaison 56

V.7.2 les Instructions de branchement (saut) 57

V.8 Mode d'adressage indexé poste-incrémenté et pré-décrémenté 57

V.8.1 Indexé poste-incrémenté 58

V.8.2 Indexé pré-décrémenté 59

V.9 Les Sous Programmes (S.P.) 60

V.9.1 Définition 61

V.9.2 Appel et retour de S.P. 61

V.9.3 Notion d'imbrication 62

V.9.4 Passation de paramètres 62

V.9.5 Mode d'adressage indexé avec déplacement fixe 64

VI Les interfaces d'entrées /sorties **65**

VI.1 Introduction 65

VI.2 Interface parallèle programmable : PIA6821 65
VI.2.1 Boîtier et brochage du PIA6821 65
VI.2.2 Registres internes du PIA6821 66
VI.2.3 Configuration du PIA 68
VI.2.4 Premier Exemple 68
VI.2.5 Deuxième Exemple 70
VI.2.6 Troisième Exemple 71

VII Les Interruptions **75**
VII.1 Introduction 75
VII.1.1 Mode programmé 75
VII.1.2 Mode Interruption 75
VII.1.3 Mode DMA 76
VII.2 Les Interruptions matérielles du μP 6809 77
VII.2.1 Définition 77
VII.2.2 Mécanisme d'une interruption matérièle 77
VII.2.3 Contrôle de la prise en compte d'interruption 78

Annexe A : Les Registres **83**

Annexe B : Les fiches TD **89**

Annexe C : Le polycopié TP **91**

References bibliographiques **93**

Table des figures

II.1 Le premier ordinateur 11
II.2 Le transistor : la révolution dans l'histoire de l'ordinateur ! 11
II.3 Les premiers PC 11
II.4 Matériel pour système à μP 12
II.5 Architecture simple d'un système à μP 13
II.6 L'architecture Von Neuman 14
II.7 L'architecture Harvard 15
II.8 Exemple de périphériques d'E/S 15
II.9 Interfaces d'entrée/sortie 16
II.10 L'architecture Harvard 16

III.1 Schema d'une bascule RS 19
III.2 Hiérarchisation de la mémoire 21
III.3 Boitier Mémoire 22
III.4 Organisation interne d'une mémoire 22
III.5 Branchement d'une mémoire 23
III.6 Chronogrammes de lecture/écriture pour une mémoire 23
III.7 Demande d'un temps d'attente 24
III.8 Selection de boitiers 24
III.9 Espace Mémoire 25
III.10 Décodage d'adresses 25
III.11 Espace Mémoire avec décodage 26

IV.1 réalisation d'un bit logique 28

IV.2 Portes logiques 28
IV.3 Exemple de circuits logiques combinatoires complexes 29
IV.4 Des circuits séquentiels de base 29
IV.5 Architecture interne d'un μP type simple 30
IV.6 Séquenceur câblé 32
IV.7 Sequence à base de mémoire 32
IV.8 Cycles du μP 33
IV.9 fonctionnement du μP 33
IV.10Fonctionnement du μP 34
IV.11Registre d'état du 6809S 36

V.1 Le MC6809 37
V.2 Brochage du MC6809 38
V.3 Dialogue avec mémoire lente 38
V.4 Horloge externe du MC6809 39
V.5 Le circuit RESET 40
V.6 L'adresse du P.P. 40
V.7 la configuration minimale matériel pour le 6809 41
V.8 Architecture interne du 6809 42
V.9 Les registres Internes du 6809 42
V.10 Le principe de la pile 43
V.11 Registre d'état 44
V.12 Exemple d'adressage absolu étendu 48
V.13 Exemple d'adressage absolu direct 48
V.14 exemple d'adressage indexé simple 49
V.15 L'organigramme de la solution 1 55
V.16 Mécanisme d'appel et de retour d'un S.P. 62
V.17 Mécanisme d'appel et de retour d'un S.P. 62

VI.1 Le circuit PIA6821 66
VI.2 composition interne simplifiée du PIA6821 67
VI.3 les bits du registre CRA (même chose pour CRB) 67

VI.4 montage et organigramme de l'exemple 1 69
VI.5 Montage de l'exemple 2 70
VI.6 Organigramme de l'exemple 2 70
VI.7 Montage de l'exemple 3 et le chronogramme poignée de main du périphérique 72
VI.8 les bits du registre CRA (même chose pour CRB) 73
VI.9 Organigramme pour l'exemple 3 74

VII.1 Mode programmé par scrutation 76
VII.2 exemple : mode interruption 76
VII.3 chronogramme d'une interruption autorisée 78
VII.4 les différents bits du registre d'état CCR 78
VII.5 Montage de l'exemple d'interruption 80
VII.6 Schema d'une bascule RS 83
VII.7 Schema d'une bascule RSH 84
VII.8 Schema d'une bascule RSH 84
VII.9 Schema d'une bascule RSH 84
VII.10 Schema d'un registre 85
VII.11 Schema d'un registre mémoire 4 bits 85
VII.12 Registre EP-LS 86
VII.13 Registre ES-LP 87
VII.14 Registre EP-LS 87
VII.15 Schema d'un registre universel 87

Chapitre I

Les systèmes de numération

I.1 Introduction

On considère un nombre entier positif A de n chiffres $\{p_i\}$ exprimé dans un système de numération de base B. La valeur décimale de A est obtenue par la relation suivante :

$$A = \sum_{i=0}^{n-1} p_i B^i = p_{n-1}B^{n-1} + \cdots + p_2 B^2 + p_1 B^1 + p_0 B^0 \tag{I.1}$$

On a toujours : $0 \leqslant p_i < B$ où p_i présente un poids. B^0 correspond au poids le plus faible et B^{n-1} correspond au poids le plus fort.

Remarque : Dans un système de base B, avec n chiffres, on peut former B^n nombres et compter de 0 à (B^{n-1}).

On veut déterminer N, nombre de chiffres dans une base B, nécessaire pour exprimer un même nombre de nombres en base décimale (10^n), c'est-à-dire une même quantité d'information. On écrit :
$10^n = B^N \Rightarrow \log_{10}(10^n) = \log_{10}(B^N) \Rightarrow n = N \log_{10}(B)$

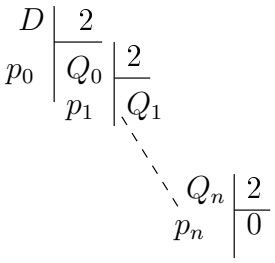
I.2 Le système binaire

Dans le système de numération à la base $B = 2$, les seuls chiffres existants sont donc **0** et **1**. C'est le seul système qu'on peut réaliser au niveau matériel de façon simple $\rightarrow$ 2 états 0 ou 1 :

- information ou pas d'information
- relais ou interrupteur ouvert ou fermé
- transistor passant ou bloqué

Tout nombre décimal D peut être convertie dans la base 2 par la division successive par 2 :

$$D = p_n 2^n + \cdots + p_2 2^2 + p_1 2^1 + p_0 2^0$$



Exemple :

$$\begin{aligned}(10101)_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 16 + 4 + 1 = 21\end{aligned}$$

Toute information est stockée, dans un système numérique, sous forme de **BIT** (***BI****nary digi****T***). Ces informations sont regroupées par blocs de :

- ✓ **8 bits** nommés **OCTETs** (***BYTE*** en anglais)
- ✓ **16 bits** nommés **MOTs** (***WORD*** en anglais)
- ✓ **32 bits nommés MOTs LONGs** (***LONG WORD***).

Notons en particulier :

1k (kilo) $=2^{10}=1024$;	1M (méga) $=2^{20}=1024$k
1G (giga) $=2^{30}=1024$M ;	1T (térra) $=2^{40}=1024$G

I.2.1 Le système Hexadécimal

Dans la base Hexadécimal ($B = 16$), les chiffres existants sont donc : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Conversion binaire– hexadécimal : on peut passer d'une représentation binaire vers une représentation hexadécimal, et vice versa, en effectuant des groupements de 4 bits $\rightarrow 16 = 2^4$ du nombre à convertir. Et chaque groupement, on lui donne son équivalent hexadécimal (tableau : I.1).

Exemple :

- ► $(0110\,1100)_2 \leftrightarrow (6C)_H$
- ► $(11\,1001)_2 \leftrightarrow (39)_H$
- ► $1k = 1024 = (100\,0000\,0000)_2 = (400)_h$
- ► $(3FC8)_h = (0011\,1111\,1100\,1000)_2$

TABLE I.1 – Représentation binaire-hexadécimal

Nombres décimaux	Représentation binaire				En Hexa
	2^3=8	2^2=4	2^1=2	2^0=1	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	A
11	1	0	1	1	B
12	1	1	0	0	C
13	1	1	0	1	D
14	1	1	1	0	E
15	1	1	1	1	F

I.3 Codage de l'information

I.3.1 Code DCB (Décimal codé binaire, BCD en anglais)

Chaque chiffre décimal $(0, \ldots, 9)$ est représenté par un groupe de quatre bits en binaire.
Exemple :

- $(45)_{10} = (0100\,0101)_{BCD}$
- $(5609)_{10} = (0101\,0110\,0000\,1001)_{BCD}$

I.3.2 Représentation des caractères alphanumériques

Le code **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) de base utilise 7 bits et permet de représenter 128 (2^7) caractère différents :

- les lettres majuscules et minuscules ;

- les chiffres (de 0 à 9) ;
- les caractères spéciaux et symboles numérique (&, ∗, $ ...) ;
- les caractères de control (retour chariot, fin ligne, ...)

TABLE I.2 – Code ASCII

HEX	**Contrôle**		**Symbole num.**		**Majuscules**		**Minuscules**	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DLE	SP	0	@	P	’	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	♯	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	∗	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS()	,	<	L		l	
D	CR	GS()	-	=	M	]	m	}
E	SO	RS()	.	>	N	∧	n	∽
F	SI	US()	/	?	O		o	DEL

I.3.3 L’arithmétique binaire

Les circuits logiques opèrent sur des nombres qui ont toujours le même nombre de bits (**format**). Par exemple, si le format est de 8 bits : $(23)_{10} = (10111)_2 \rightarrow (00010111)_2$

Principe :

		A + B		A - B	
A	B	Résultat	Retenue	Résultat	Retenue
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

- ✓ En effectuant une addition, une **retenue** (**carry**) peut apparaître.
- ✓ En effectuant une soustraction, une **retenue** (**borrow**) peut apparaître.
- ✓ En manipulant des nombres trop grands pour le format, on obtient un **dépassement de capacité** (**overflow**).

	Addition		Soustraction	
Exemple :	59 $+42$ 101_{10}	$0^{1}0^{1}1^{1}11^{1}011$ $+\ 0\ 0\ 1\ 01\ 010$ $0\ 1\ 1\ 00\ 101$	$^{7}8^{1}5$ $-\ 5\ \ 7$ $2\ \ 8$	$\overset{1\ \ 10}{0\ 10\ 0\ 10}$ $0\,1\,0\,1\,0\,101$ $-\ 0\ 0\ \ 1\ \ 1\ \ 1001$ $0\ 0\ \ 0\ \ 1\ \ 1100$

I.3.4 Représentation des nombres négatifs

C'est le bit de poids fort (**MSB** : Most Signficant Bit) qui représente le signe d'un nombre signé. On a alors : $0 \rightarrow$ positif ; $1 \rightarrow$ négatif. Il existe trois modes de représentation.

I.3.4.1 Signe et valeur absolue

Le MSB est le bit de signe, les autres bits indiquent la valeur absolue des nombres.
Exemple :

- ► $(23)_{10} = \mathbf{0}0010111$
- ► $(-23)_{10} = \mathbf{1}0010111 \Rightarrow$ et non $(128+23)_{10} = (151)_{10}$ exprimé en binaire pure.

Remarque : Cette représentation ne convient pas pour les calculs arithmétiques.

I.3.4.2 Complément à 1

Si n est le nombre de bits constituant le nombre D, il est appelé aussi complément à 2^n-1. C'est le complément logique ($-D = \overline{D}$) de tout les bits, bit à bit : $-D+D = 2^n-1$.

Exemple :

- ► $(23)_{10} = \mathbf{0}0010111$
- ► $(-23)_{10} = \mathbf{1}1101000$

  $\Rightarrow$ et non $(128+64+32+8)_{10} = (232)_{10}$ exprimé en binaire pure.

I.3.4.3 Complément à 2

Si n est le nombre de bits constituant le nombre D, il est appelé aussi complément à 2^n. C'est le complément à 1 auquel on ajout 1 ($-D = \overline{D}+1$) : $-D+D = 2^n$.

Exemple :

$(23)_{10} = 0\,0010111$

$(-23)_{10} = 1\,1101000 \Rightarrow$ complément à 1

$+ \qquad\qquad 1$

$(-23)_{10} = 1\,1101001 \Rightarrow$ complément à 2 et non $(233)_{10}$ en base 2

Autres exemples :

$17 + 8 = 25$

Signe	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
0	1	0	0	0	1
0	0	1	0	0	0
0	1	1	0	0	1

$17 + 17 = 34$

Signe	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	1	0

Dans ce dernier exemple, le résultat est faux, car le signe des deux opérandes est positif (bit de signe à 0) et celui du résultat est négatif : il y a dépassement de capacité.

Le tableau (I.3) donne l'interval de nombres signés représentés en complément à 2 sur 8 bits.

I.3.4.4 La soustraction

Elle est ramenée à une addition effectuée sur les nombres représentés en complément à deux. Avant d'effectuer les opérations il faut toujours définir le format (la taille) de représentation des nombres (dans les exemple ça sera sur 8 bits ; le bit MSB est toujours le bit de signe).

Exemple 1 :

$17 + (-8) = +9$

Retenue	Signe							
	0	0	0	1	0	0	0	1
	1	1	1	1	1	0	0	0
1	0	0	0	0	1	0	0	1

Le complèment à 2 de (-8)

	0	0	0	0	1	0	0	0	**(+8)**
	1	1	1	1	0	1	1	1	
+								1	
	1	1	1	1	1	0	0	0	**C/2(-8)**

Exemple 2 :

TABLE I.3 – Représentation des nombres signés sur 8 Bits

Décimal	Représentation des nombres signés sur 8 Bits
+127	0111 1111
+7	0000 0111
+6	0000 0110
+5	0000 0101
+4	0000 0100
+3	0000 0011
+2	0000 0010
+1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
-3	1111 1101
-4	1111 1100
-5	1111 1011
-6	1111 1010
-128	1000 0000

$(-17) + (-8) = X = -25$

Retenue	Signe							
	1	1	1	0	1	1	1	1
	1	1	1	1	1	0	0	0
1	1	1	1	0	0	1	1	1 (X)

On ne tient pas compte de la retenue.

	1	1	1	0	0	1	1	1	**(X)**
	0	0	0	1	1	0	0	0	
+								1	
	0	0	0	1	1	0	0	1	**C/2(+25)**

	0	0	0	0	1	0	0	0	**(+8)**
	1	1	1	1	0	1	1	1	
+								1	
	1	1	1	1	1	0	0	0	**C/2(-8)**

	0	0	0	1	0	0	0	1	**(+17)**
	1	1	1	0	1	1	1	0	
+								1	
	1	1	1	0	1	1	1	1	**C/2(-17)**

Exemple 3 :

$96 + 48 = +154$

Retenue	Signe							
	0	1	1	0	0	0	0	0
	0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0	0

$(-96) + (-48) = -154$

Retenue	Signe							
	1	0	1	0	0	0	0	0
	1	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	0

Dans les deux derniers cas (exemple 3), les résultats sont faux, car les signes des résultats ne correspondent pas aux vrais résultats → il y a **dépassement de capacité**.
Si "-" est l'opérateur de complémentation, "+" et "· " sont la somme et le produit logique respectivement. On peut donc exprimer le débordement (dépassement de capacité) par :

$$\text{si } A + B = R \quad \Rightarrow \quad DEB = \overline{S_A} \cdot \overline{S_B} \cdot S_R + S_A \cdot S_B \cdot \overline{S_R} \tag{I.2}$$

où : S_A est le signe de A
S_B est le signe de B
S_R est le signe du résultat R.

Chapitre II

Historique et évolution des ordinateurs

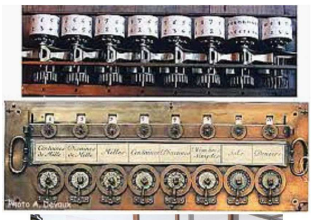
II.1 Histoire de l'ordinateur

La construction de l'ordinateur s'est inspirée des idées et des développements de tous les temps, mais n'a réellement connu des racines qu'au 17^{me} siècle, siècle du début de l'ère de l'industrie.

II.1.1 Génération zéro : les calculateurs mécaniques (1642 – 1945)

Les premières machines à calculer étaient purement mécaniques : bouliers, abaques, ... et on trouve :

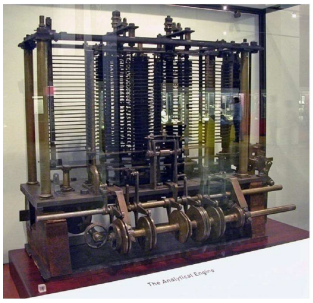
La Pascaline : Blaise Pascal (1623 – 1662), machine à additionner ;



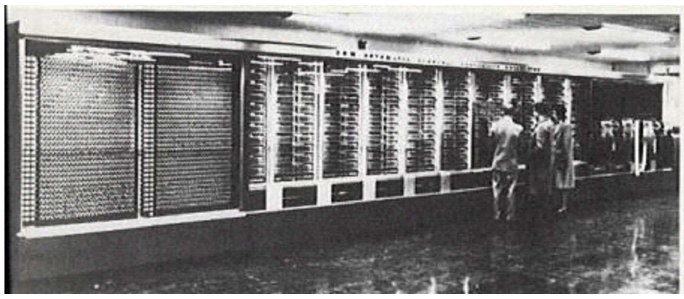
Machine à multiplier : Leibniz, 1694, perfectionne la machine de Pascal ;

Première machine programmable : métier à tisser, Jacquard,18ème siècle, machine à cartes perforées.

Machine analytique programmable : Charles Babbage (1792 – 1871), non réalisable avec les technologies de l'époque.



Mark I : Un descendant direct de la machine analytique de Babbage. Ces machines sont à base de relais électromécaniques (*Aiken* et *Stibitz*, 1936-1944).



II.1.2 La première génération : les tubes à vide (1945 – 1955)

Première machine à calculer électronique : ENIAC, 1946, 30t, $72m^2$, 170kW, 19000 tubes, 350 multiplications/s, 5000 additions/s. machine à programme câblé. La première erreur informatique est due à un insecte qui, attiré par la chaleur, était venu se loger dans les lampes et avait créé un court-circuit.

→ « insecte » en anglais : « Bug ».

Machine à programme enregistré (figure II.1) : John Von Neumann, 1952, les instructions sont enregistrées dans la mémoire du calculateur : ordinateur.

II.1.3 La deuxième génération : les transistors (1955 – 1965)

Ordinateur à transistors : 1963, PDP5 de Digital Equipment Corporation (DEC), introduction des mémoires à ferrites : mini-ordinateurs. L'invention du transistor à semi-conducteur a permit une réduction de taille phénoménale des ordinateur et des systèmes électroniques de façon générale (figure II.2).

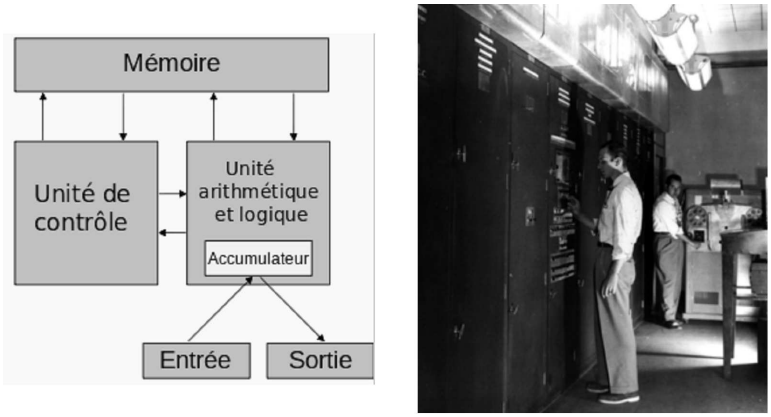


FIGURE II.1 – Le premier ordinateur

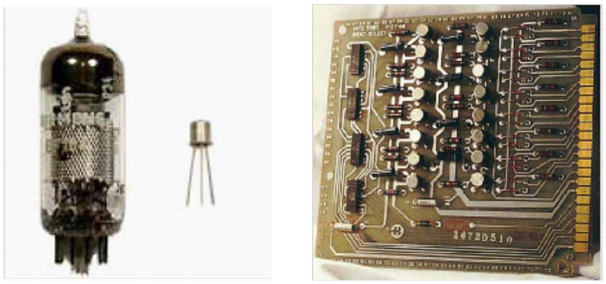


FIGURE II.2 – Le transistor : la révolution dans l'histoire de l'ordinateur !

II.1.4 La troisième génération : les circuits intégrés (1965 – 1980)

Micro-ordinateurs : 1969-70, utilisation des circuits intégrés LSI. Premier microprocesseur : Intel, 1971, microprocesseur 4004, puis 8008, Autres microprocesseurs : 8080 et 8085 d'Intel, 6800 et 6809 de Motorola, ... Microprocesseurs 16 bits : 8086/8088 d'Intel, 68000 de Motorola. Microprocesseurs 32 bits en 1986 : 80386 d'Intel et 68020 de Motorola. Fabrication en grandes séries des micro-ordinateurs : 1977, Apple, Commodore, Tandy. IBM PC + MS-DOS (Microsoft) en 1981.

FIGURE II.3 – Les premiers PC

II.1.5 La quatrième génération : VLSI (Machines actuelles)

Ordinateurs de plus en plus puissants, basés sur des microprocesseurs performants : Pentium, Power PC,Dual CORE, I3, I7 ... Il est très difficile de nos jours de suivre l'évolution de l'ordinateur ...

II.2 Architecture d'un système à microprocesseur

Les applications des systèmes à microprocesseurs sont multiples et variées :

- Ordinateur, Automate Programmable Industriel ;
- console de jeux ;
- calculatrice ;
- télévision ;
- téléphone portable ;
- distributeur automatique d'argent ;
- robotique ;
- lecteur carte à puce, code barre ;
- automobile ;
- etc ...

II.2.1 Notion de HARDWARE (Matériel)

C'est tous les composants physiques qui constituent un système, à savoir circuits imprimés, composants électroniques, alimentation, châssier, ... Ca représente la partie palpable ou concrète du système (figure II.4). L'architecture d'un système à microprocesseur (μP) représente l'organisation de ses différentes unités et de leurs interconnexions.

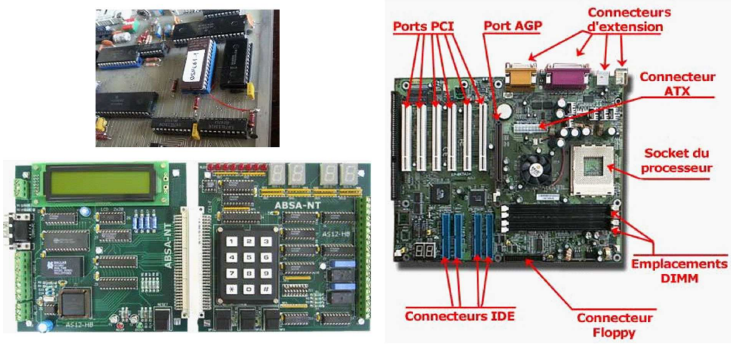


FIGURE II.4 – Matériel pour système à μP

II.2.1.1 Structure générale

C'est un système logique programmable, conçu à base d'un μP, et comprend principalement :

- ✓ un μP ;
- ✓ mémoires ;
- ✓ Circuits d'interface entrées/sorties
- ✓ horloge.

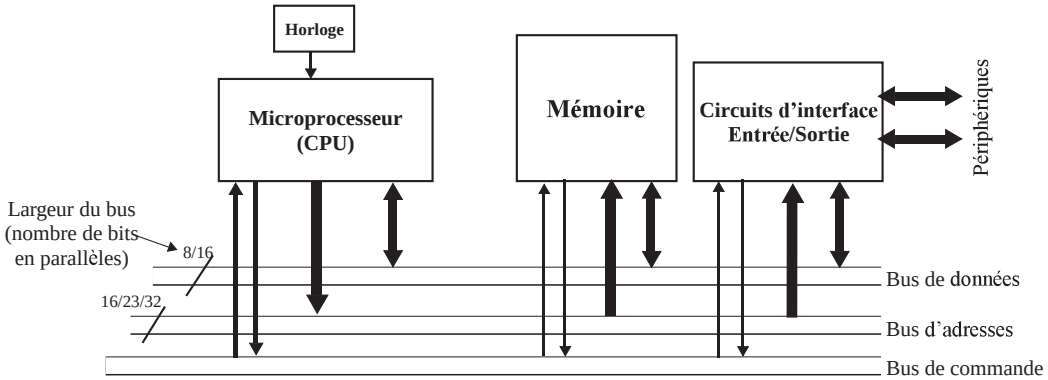


FIGURE II.5 – Architecture simple d'un système à μP

Microprocesseur : C'est l'élément exécuteur (**CPU**, **C**entral **P**rocessing **U**nit), exécute les ***instructions*** qui se trouvent en mémoire (vive ou morte).

Mémoire morte : elle contient des programmes et des données *ineffaçables*.

Mémoire vive : elle contient des programmes et des données qui peuvent être effacées par le μ**P**.

Interfaces : circuits servants d'intermédiaire entre le μ**P** et d'autres éléments qui communiquent avec ce circuit.

Horloge : c'est la base de temps qui cadence le fonctionnement séquentiel du μ**P**.

Les différentes unités sont réunies par des canaux d'échanges : ***les bus***.
Les microprocesseurs peuvent être classés selon la longueur maximale des mots binaires qu'ils peuvent échanger avec la mémoire et les E/S : microprocesseurs 8 bits, 16 bits, 32 bits, ... Le bus peut être décomposé en trois bus distincts :

le bus d'adresses : permet au microprocesseur de spécifier l'adresse de la case mémoire à lire ou à écrire ;

le bus de données : permet les transferts entre le microprocesseur et la mémoire ou les E/S ;

le bus de commande : transmet les ordres de lecture et d'écriture de la mémoire et des E/S.

Pour l'organisation des différentes unités, il existe deux architectures :
- ✓ l'architecture Von Neuman
- ✓ l'architecture Harvard

II.2.1.2 L'architecture Von Neuman

Le μ**P** utilise le même canal d'échange pour lire les instructions et manipuler des données : le bus de données. Les tailles les plus courantes des bus de données sont :

- ► 8 bits pour les petites applications embarquées
- ► 16 bits pour les applications embarquées de moyenne complexité
- ► 32 bits, 64 bits pour les gros calculateurs, les ordinateurs et consoles de jeux

Les adresses délivrées par le μP sont véhiculées par le bus d'adresses !!

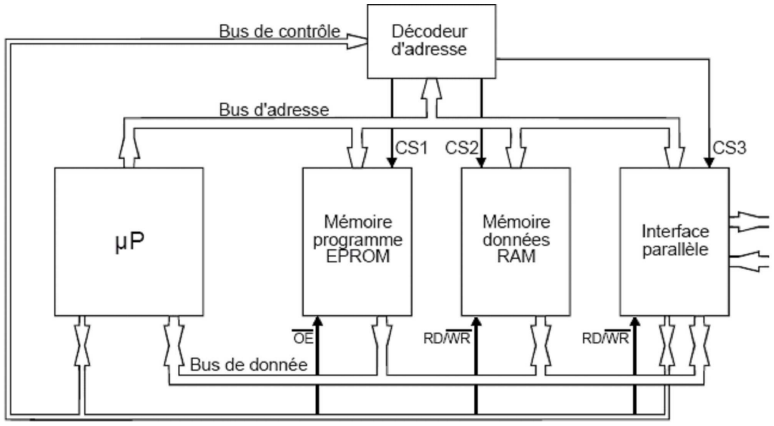


FIGURE II.6 – L'architecture Von Neuman

II.2.1.3 L'architecture Harvard

Le μP utilise 2 canaux d'échange pour lire les instructions et manipuler les données : le bus instruction (ou bus programme) et le bus de donnée. Les tailles de ces bus sont en général différentes. La taille la plus courante du bus de données est de 8 bits. Les deux bus distincts programme et donnée permettent de réaliser simultanément une recherche d'une instruction et l'exécution de l'instruction précédente.

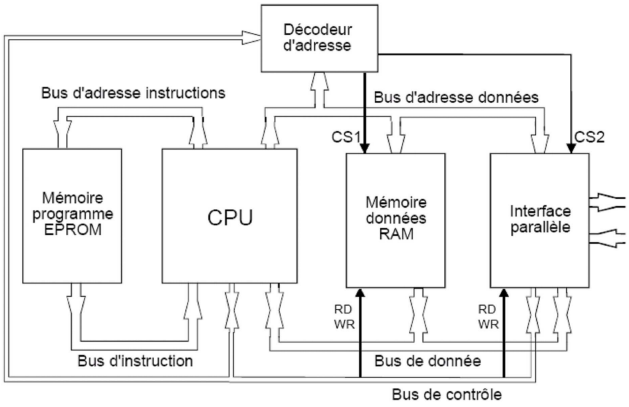


FIGURE II.7 – L'architecture Harvard

II.2.2 Notion d'Entrées/Sorties

Un μP n'est pas fait pour dialoguer uniquement avec sa mémoire. Il doit pouvoir communiquer avec la périphérie et recevoir ou transmettre des informations de ou vers l'environnement extérieur (en particulier avec l'opérateur humain). Cette communication avec l'extérieur reçoit le nom d'entrée / sortie.

Les entrées : clavier, disque, carte d'acquisition, ...

Les sorties : écran, disque, imprimante, table traçante, ...

FIGURE II.8 – Exemple de périphériques d'E/S

L'environnement d'un système à μP peut être soit numérique soit analogique. Dans le cas où l'organe périphérique est de type analogique, le dialogue est adapté grâce aux convertisseurs analogique/numérique (CAN ou DAC) et numérique / analogique (CNA ou ADC).

*** Notion d'interface :**

Les interfaces apportent :

- ✓ une adaptation matérielle entre le bus μP et le périphérique ;
- ✓ une adaptation protocolaire en coordonnant le dialogue et la vitesse d'échange des informations entre le μP et périphérique.

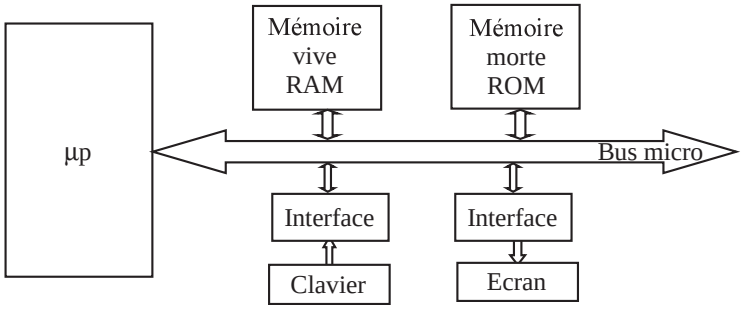


FIGURE II.9 – Interfaces d'entrée/sortie

*** Les E/S parallèles et E/S série**

On peut regrouper les entrées/sorties en deux catégories :

- les entrées/sorties parallèles ;
- les entrées/sorties séries.

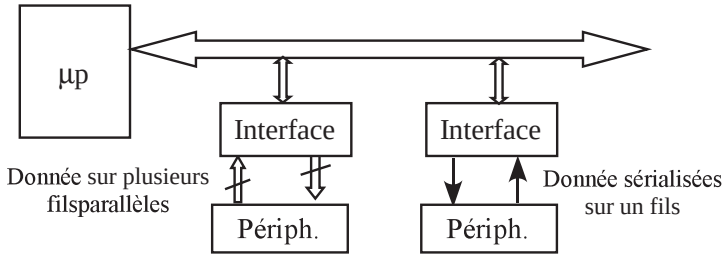


FIGURE II.10 – L'architecture Harvard

*** Méthodes d'E/S**

Afin d'interfacer des périphériques avec un μP, trois méthodes de base sont utilisées :

- ✓ E/S programmées (par sondage ou par scrutation) ;
- ✓ Interruptions ;
- ✓ Accès direct mémoire (DMA).

II.2.3 Notion de SOFTWARE (Logiciel)

C'est l'ensemble des programmes, des méthodes, et des règles de programmation destinées à faire fonctionner et à utiliser un système à μP. Le ***software*** est la partie *abstraite* liée à l'exploitation du système.

Le programme de l'utilisateur : C'est tout programme écrit en langage machine, en langage assembleur ou en langage évolué par l'utilisateur du système pour ses applications.

Interpréteur et compilateur pour langage évolué : Ce sont des programmes spéciaux qui interprètent les programmes de l'utilisateur écrits en langage évolué, en des programmes écrits en langage machine, donc compréhensible par le μP.

Le programme système (système d'exploitation) : Le système d'exploitation (SE, en anglais Operating System ou OS) est un ensemble de programmes responsables de la liaison entre les ressources matérielles d'un ordinateur et les applications informatiques de l'utilisateur (traitement de texte, jeu vidéo...). Il fournit aux programmes applicatifs des points d'entrée génériques pour les périphériques.

Chapitre III

Les Mémoires à semi-conducteurs

III.1 Introduction

On appelle ***mémoire*** tout composant électronique capable de stocker des données. On distingue ainsi deux grandes catégories de mémoires dans un système à μP :

- **la mémoire centrale :** permettant de mémoriser temporairement les données et des programmes. La mémoire centrale correspond à ce que l'on appelle la mémoire vive.
- **la mémoire de masse :** permettant de stocker des informations à long terme, y compris lors de l'arrêt du système. La mémoire de masse correspond aux dispositifs de stockage magnétiques, optique ou autres, tels que :
  - Le disque dur,
  - CD-ROM , DVD-ROM,
  - mémoires mortes ;
  - Clés usb ;
  - Mémoires SD ...

On distingue plusieurs types de mémoires à semi-conducteurs :

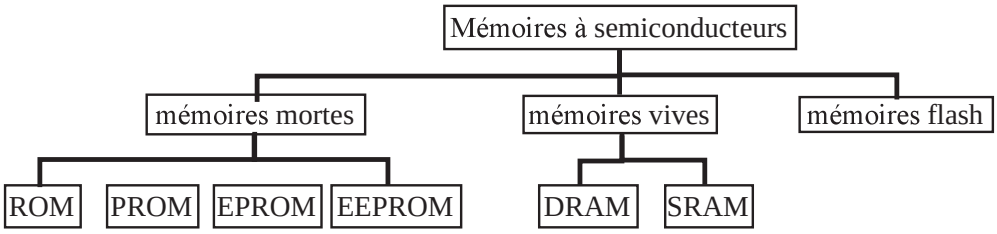


FIGURE III.1 – Schema d'une bascule RS

Mémoires mortes : Elles sont réalisées en technologie MOS et bipolaire. Ces mémoire trouvent leur application dans la conversion de code, à génération de caractères pour l'affichage sous forme de matrice de points, le stockage de programmes systèmes.

Mémoires vives (RAM) : Elles sont souvent qualifiées de RAM (Random Access Memory), mais les mémoires mortes sont également à accès aléatoire. Elle nécessite une énergie pour stocker et garder les données.

Mémoires flash : La mémoire flash est une mémoire à semi-conducteurs, non volatile et réinscriptible, qui fonctionne comme la combinaison d'une RAM et d'un disque dur. La mémoire flash stocke les bits de données dans des cellules de mémoire, comme une DRAM, mais elle fonctionne comme un disque dur, dans la mesure où les données sont conservées en mémoire lorsque l'alimentation électrique est coupée.

III.2 Mémoires mortes

Dans cette catégorie de mémoire, on distingue :

Read Only Memory (ROM) : Mémoire à lecture seule. Son contenu est programmé une fois pour toutes par le constructeur.

Programmable Read Only Memory (PROM) : C'est une mémoire ROM programmable une seule fois par l'utilisateur en faisant sauter des fusibles nécessitant un programmateur spécialisé.

Erasable PROM (EPROM ou UVPROM) : C'est une mémoire ROM programmable électriquement avec un programmateur et effaçable par exposition à un rayonnement ultraviolet pendant 30 minutes.
Exemple : famille 27nnn, : 2764 (8 Ko), 27256 (32 Ko).

Electrically Erasable PROM (EEPROM) : C'est une mémoire ROM programmable et effaçable électriquement. Lecture à vitesse normale est $\leqslant$ 100 ns. Ecriture est très lente ($\approx$ 10 ms). Les EEPROM peuvent contenir des données qui peuvent être modifiées de temps en temps, exemple : paramètres de configuration des ordinateurs.

Avantage : programmation sur la carte et sans programmateur

Inconvénient : coût élevé

III.3 Mémoires vives : RAM

III.3.1 RAM Statique (SRAM)

La réalisation interne est essentiellement faite par une matrice de bascules :

→ m bascules de large, où m est la taille du mot ;
→ 2^n bascules de long, où 2^n est la capacité en mots de la mémoire.

Chaque bascule contient entre 4 et 6 transistors. Cette réalisation est simple mais elle nécessite un décodeur interne à 2^n sorties, donc une surface de boîtier importante. Elle a cependant un accès rapide.

III.3.2 RAM dynamique (DRAM)

Dynamic RAM est une mémoire basée sur la charge de condensateurs : condensateur chargé = 1, condensateur déchargé = 0.

- **Avantage :** intégration élevée, faible coût ;
- **Inconvénient :** nécessite un rafraîchissement périodique à cause du courant de fuite des condensateurs.

Il y'a d'autres types dans cette famille de mémoire RAM : SDRAM, DDR SDRAM (Double Data Rate Synchronous DRAM)

III.4 Critères de choix d'une mémoire

Les principaux critères à retenir sont : Capacité, vitesse d'écriture/lecture , consommation d'énergie, Coût.

- **Notion de hiérarchie mémoire :** Une mémoire idéale serait une mémoire de grande capacité et possédant un temps d'accès très faible afin de pouvoir travailler rapidement avec le μP. Mais il se trouve que les mémoires de grande capacité sont souvent très lente et que les mémoire rapides sont très chères.

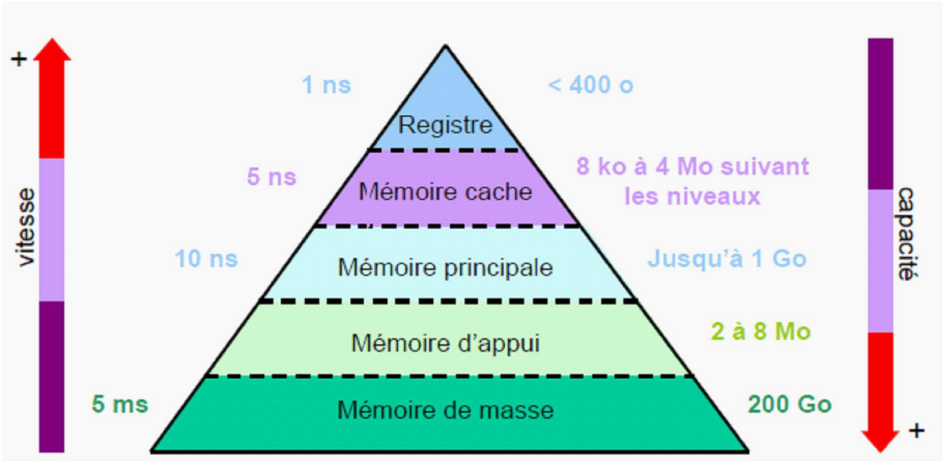


FIGURE III.2 – Hiérarchisation de la mémoire

III.5 Fonctionnement d'une mémoire

III.5.1 Schéma fonctionnel d'une mémoire

n lignes d'adresses permettent d'adresser 2^n cases mémoire : 8 bits d'adresses permettent d'adresser 256 octets, 16 bits d'adresses permettent d'adresser 65536 octets (= 64 Ko), ...

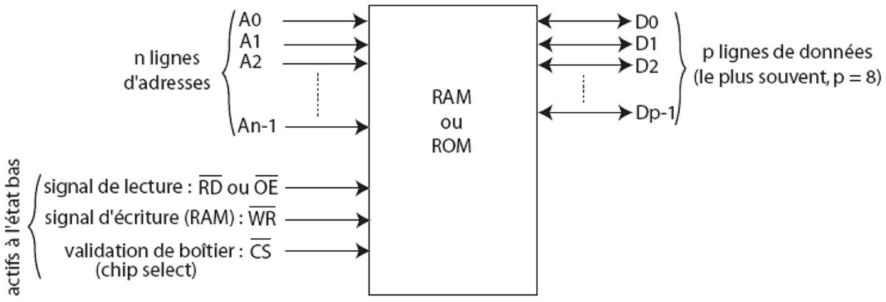


FIGURE III.3 – Boitier Mémoire

Exemple : Mémoire RAM 6264 : 13 broches d'adresses de A_0 à A_{12}, $2^{13} = 8192 = 8$ Ko (capacité = 8K $\times$ 8 bits).

III.5.2 Organisation de la mémoire

La mémoire peut être vue comme un ensemble de cellules ou cases contenant chacune une information : une instruction ou une donnée. Chaque case mémoire est repérée par un numéro d'ordre unique : son adresse.

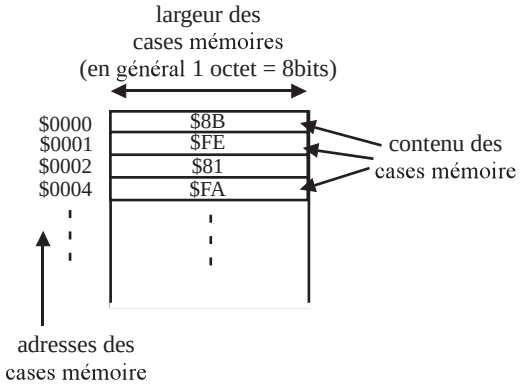


FIGURE III.4 – Organisation interne d'une mémoire

III.6 Interfaçage microprocesseur/mémoire

Une caractéristique importante des mémoires est leur temps d'accès : c'est le temps qui s'écoule entre l'instant où l'adresse de la case mémoire est présentée sur le bus d'adresses et celui où la mémoire place la donnée demandée sur le bus de données. Ce temps varie entre 50 ns (mémoires rapides) et 300 ns (mémoires lentes).

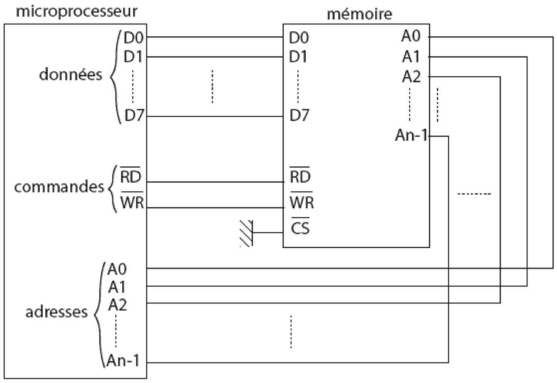


FIGURE III.5 – Branchement d'une mémoire

III.6.1 Chronogrammes de lecture-écriture

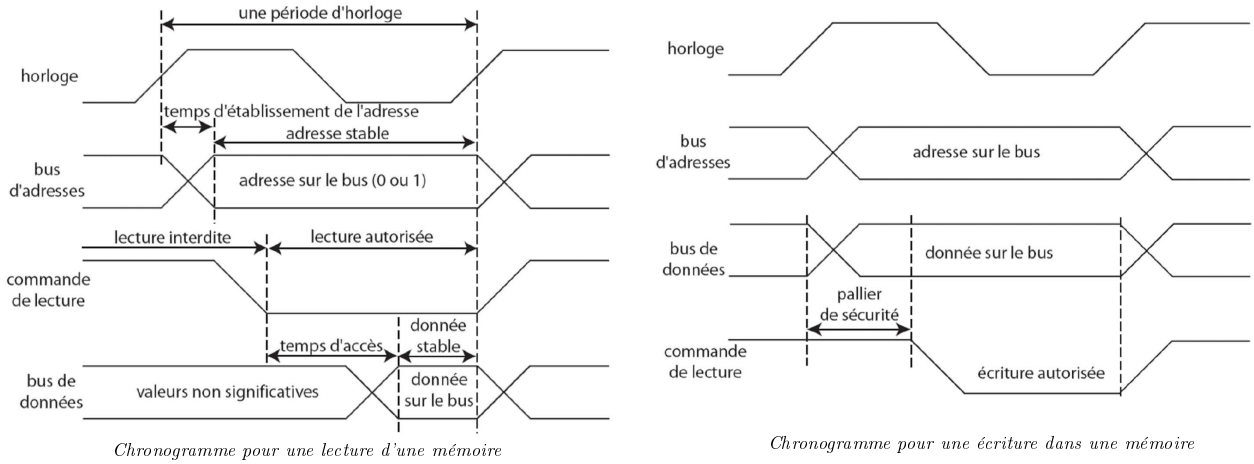


FIGURE III.6 – Chronogrammes de lecture/écriture pour une mémoire

Remarque : si le temps d'accès d'une mémoire est supérieur à une période d'horloge (mémoire lente), le microprocesseur peut accorder à la mémoire un temps supplémentaire à la demande de celle-ci.

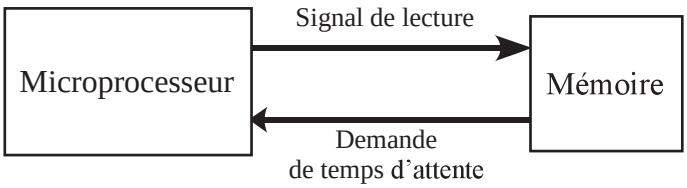


FIGURE III.7 – Demande d'un temps d'attente

III.7 Extension de la capacité mémoire

III.7.1 Connexion de plusieurs boîtiers mémoire sur le bus d'un μP

Les boîtiers mémoires possèdent une broche notée CS : Chip Select. Lorsque cette broche est active (état bas), le circuit peut être lu ou écrit. Lorsqu'elle est inactive (état haut), le circuit est exclu du service : comme si la mémoire était déconnectée du bus de données du μP, d'où la possibilité de connecter plusieurs boîtiers mémoire sur un même bus.

Exemple : connexion de trois boîtiers mémoire d'une capacité de 8 Ko chacun (13 lignes d'adresses) sur un bus d'adresse de 16 bits :

Dans un même boîtier, une case mémoire est désignée par les bits d'adresses A_0 à A_{12} :

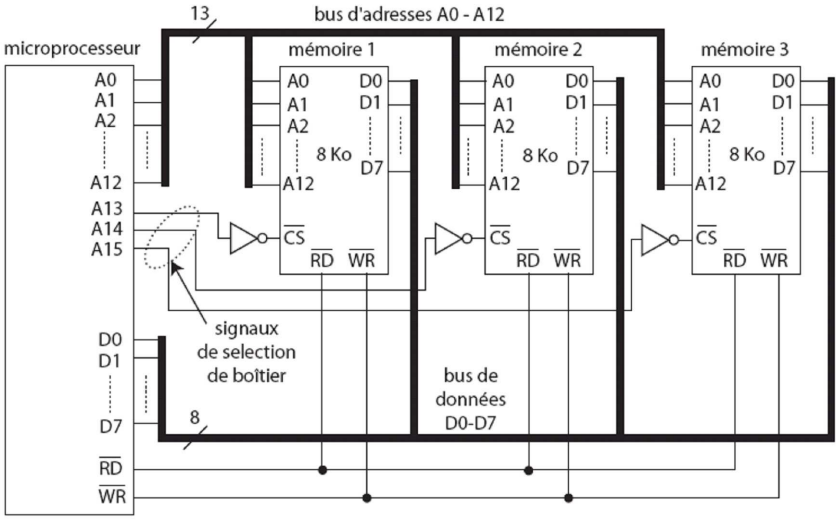


FIGURE III.8 – Selection de boitiers

III.7.2 Le mapping de la mémoire

On en déduit la cartographie ou mapping de la mémoire visible par le microprocesseur :

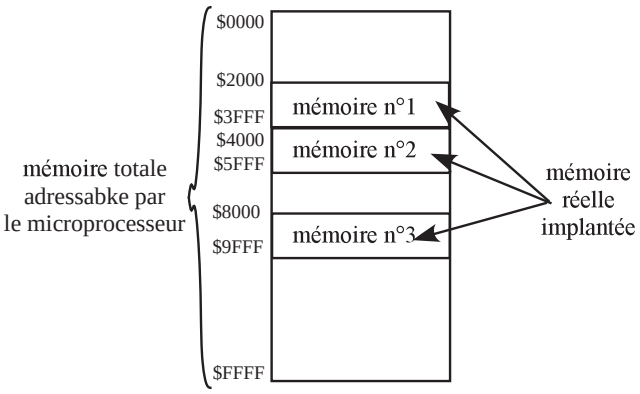


FIGURE III.9 – Espace Mémoire

III.7.3 Décodage d'adresses

Les trois bits A_{13}, A_{14} et A_{15} utilisés précédemment fournissent en fait 8 combinaisons, de 000 à 111, d'où la possibilité de connecter jusqu'à 8 boîtiers mémoire de 8 Ko sur le bus. La mémoire totale implantée devient donc de 8×8 Ko $= 64$ Ko : valeur maximale possible avec 16 bits d'adresses. Pour cela, il faut utiliser un circuit de décodage d'adresses, dans ce cas : un décodeur 3 vers 8 (figure III.10).

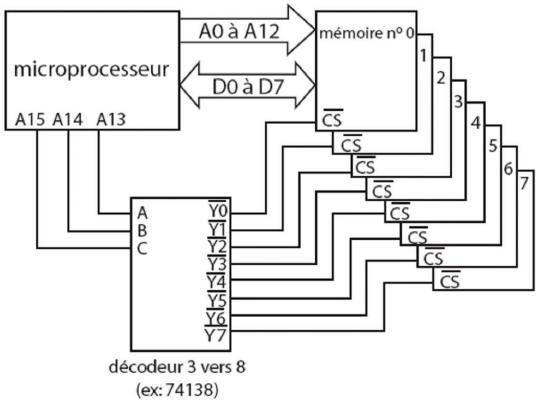


FIGURE III.10 – Décodage d'adresses

Ce décodage permet une division de l'espace mémoire adressable par le μP en zones de 8Ko :

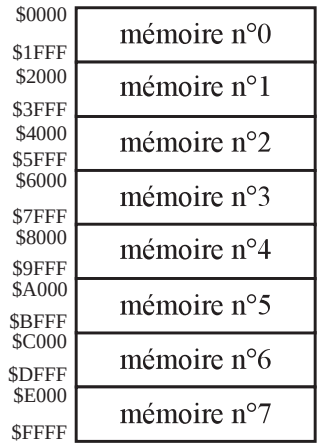


FIGURE III.11 – Espace Mémoire avec décodage

TABLE III.1 – Table de décodage d'adresse

C	B	A	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$	$\overline{Y_4}$	$\overline{Y_5}$	$\overline{Y_6}$	$\overline{Y_7}$
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Chapitre IV

Architecture et fonctionnement d'un microprocesseur

IV.1 Introduction

L'invention du microprocesseur est le fruit de décinies de développement technologique depuis l'invention du transistor à jonction. Quelques dates importantes :

- 1950 : transistor à jonction,
- 1970 : première RAM statique 256 bits,
- 1970 : première RAM dynamique, 1024 bits,
- 1971 : Premier microprocesseur lancé par Intel, le 4004 (60000/s)
- 1972 : le premier microprocesseur 8 bits, 8008, Intel
- 1974 : 6800 de Motorola
- 1977 : 8085 d'Intel,
- 1978 : 6809 de Motorola (version améliorée des μP 8 bits)
- 1979 : Microprocesseurs 16 bits : 8086/8088 d'Intel, 68000 de Motorola.
- 1985 : Microprocesseurs 32 bits en 1986 : 80386 d'Intel et 68020 de Motorola.
- 1993 : Pentium , PowerPC601
- 2000 : pentium 4
- 2008 : Intel Core i7 (Bloomfield)
- ...

La course continue ... !

IV.2 Architecture interne d'un microprocesseur

IV.2.1 Représentation binaire

Dans les systèmes numériques, toute information est codée en binaire. Le système binaire est réalisable plus aisément avec des composants électroniques simples comme le transistor (figure IV.1).

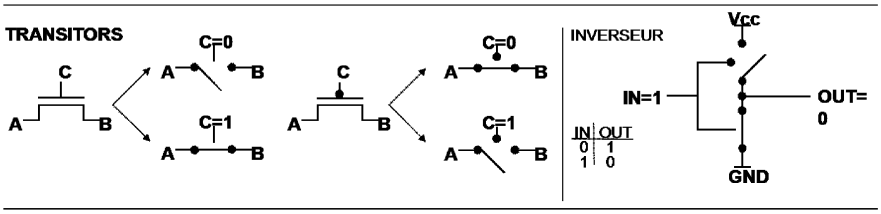


FIGURE IV.1 – réalisation d'un bit logique

Logique combinatoire : A partir des transistors, on réalise les portes logiques simples (figure IV.2). A partir des portes logique on peut réaliser des fonctions plus complexes comme une unité arithmétique et logique ou un décodeur (figure IV.3). Ceci est possible en usant des outils mathématiques logiques de synthèses :

- Algèbre booléenne
- Minimisation (tables de Karnaugh)
- ...

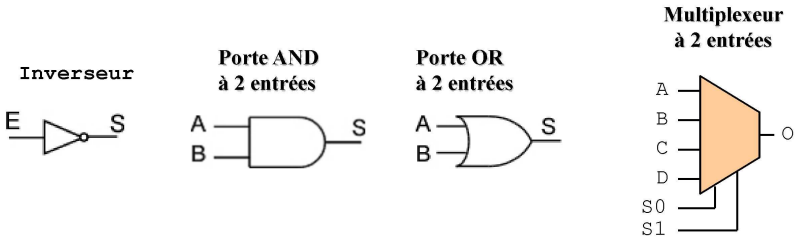
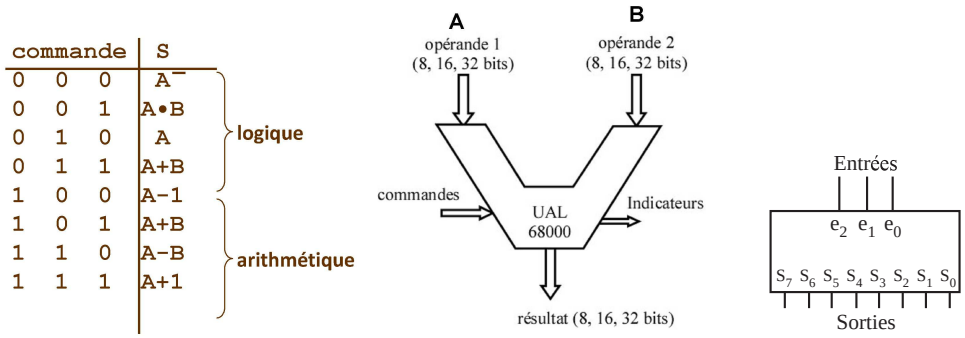


FIGURE IV.2 – Portes logiques

Logique séquentielle : A la base des circuits séquentiels, on trouve les bascules et en particulier la bascule D.



(a) UAL (b) Décodeur

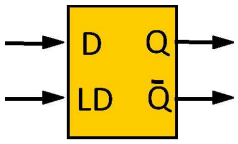
FIGURE IV.3 – Exemple de circuits logiques combinatoires complexes

- Verrou (Latch)

  D : entrée d'excitation

  LD : entrée de contrôle

  Q : sortie ou état du latch



- Autres circuits séquentiels (figure IV.4).

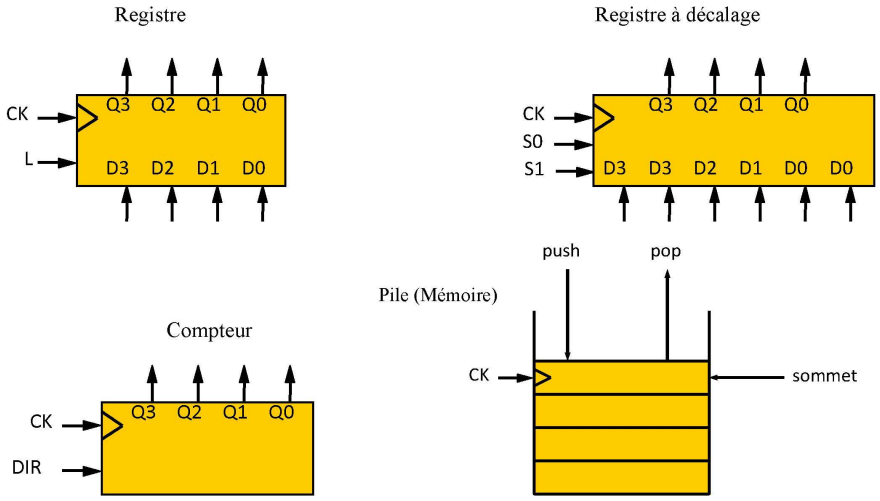


FIGURE IV.4 – Des circuits séquentiels de base

IV.2.2 Architecture interne d'un microprocesseur type

Les différents constituants d'un microprocesseur (μP) peuvent être regroupés dans deux blocs principaux : l'unité de calcul et l'unité de contrôle.

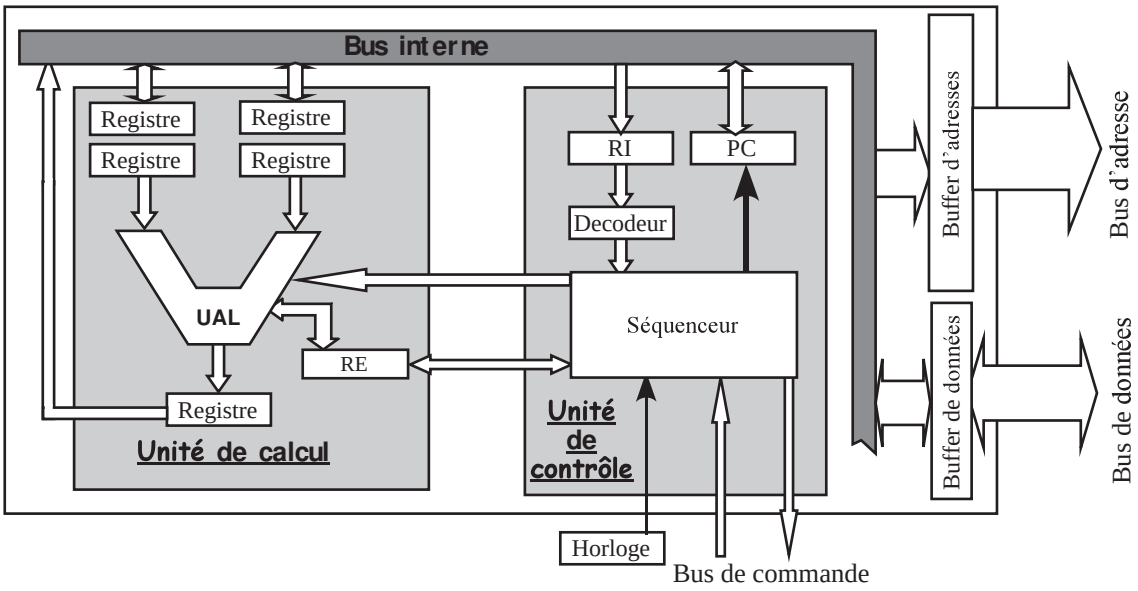


FIGURE IV.5 – Architecture interne d'un μP type simple

RI : registre d'instruction

PC : compteur de programme

RE : registre d'état

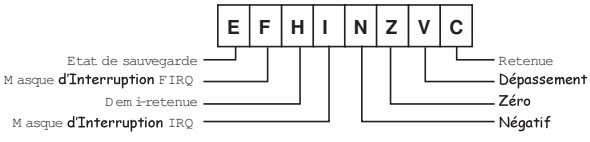
IV.2.3 L'unité de calcul

Elle est constituée de :

Registres : Ce sont des mémoires élémentaires pouvant contenir chacun un opérande. Les registres peuvent être de taille de 8, 16 ou 32 bits.

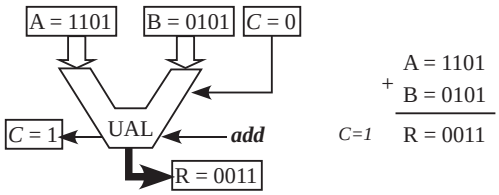
Registre d'état (RE) : Le registre d'état est formé de plusieurs bits appelés indicateur (Flags) qui sont positionnés par l'UAL après chaque opération.

Exemple : registre d'état du μP 6809
L'indicateur **Z** (=1) indique que le résultat de l'opération est égal à Zéro ;
L'indicateur **C** indique que l'opération a généré une retenue ;
Le bit **N** indique que le résultat est négatif ...



UAL : unité arithmétique et logique permet la réalisation d'opérations arithmétiques et logiques (additions et soustractions binaires, comparaison, OR, AND, XOR...).

Exemple :
Dans cet exemple, une addition est effectuée entre le registre A et le registre B. Le résultat est mis dans le registre R et la retenue dans la bascule C du registre d'état.



IV.2.4 L'unité de contrôle

C'est l'unité de control qui supervise le déroulement de toutes les opérations au sein du μP. Elle est constituée principalement de :

Horloge : C'est l'horloge qui génère les signaux qui permettent le cadencement et la synchronisation de toutes les opérations. L'horloge est constituée par un oscillateur à quartz dont les circuits peuvent être internes ou externes au microprocesseur.

Le compteur programme PC (Program Counter) : contient l'adresse de la case mémoire où est stockée la prochaine instruction à exécuter. Au début de l'exécution d'un programme, le registre PC est initialisé à l'adresse mémoire où est stockée la première instruction du programme. Le compteur programme est incrémenté chaque fois qu'une instruction est chargée dans le μP.

Le registre d'instruction RI : C'est là où le μP stocke le code opération de l'instruction qui va être d'exécution.

Le décodeur : C'est lui qui va "décoder" l'instruction contenue dans RI et générer les signaux logiques correspondant et les communiquer au séquenceur.

Le séquenceur : Il gère le déroulement séquentiel des opérations et génère :

- ► les signaux du bus de commande (RD,WR , etc.),
- ► les signaux internes aux μP (gestion des registres, de l'UAL, aiguillages des bus internes, etc...).

on peut citer deux types de technologie de séquenceur :

Séquenceur câblé : Il est réalisé entièrement à base de logique combinatoire et de la logique séquentielle (figure IV.6).

Remarque : Exécution rapide, mais réalisation complexe !

Séquenceur micro programmé : Le séquenceur est réalisé avec une structure qui comprend une mémoire ROM intégrée (figure IV.7). Celle-ci contient des micro-instructions (à ne pas confondre avec les instructions contenues dans la mémoire programme).

Remarque : à noter :

- ça nécessite un compteur ordinal pour la lecture séquentielle des micro commandes à partir de la mémoire de microprogrammes.
- la mémoire des micro-instructions n'est pas accessible à l'utilisateur.

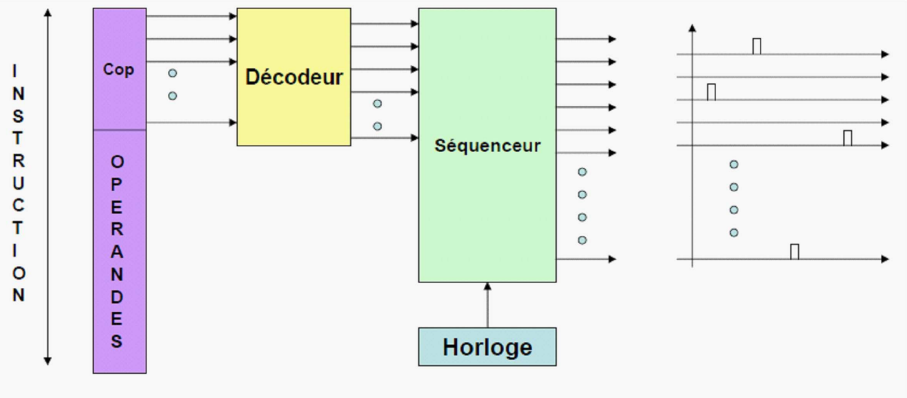


FIGURE IV.6 – Séquenceur câblé

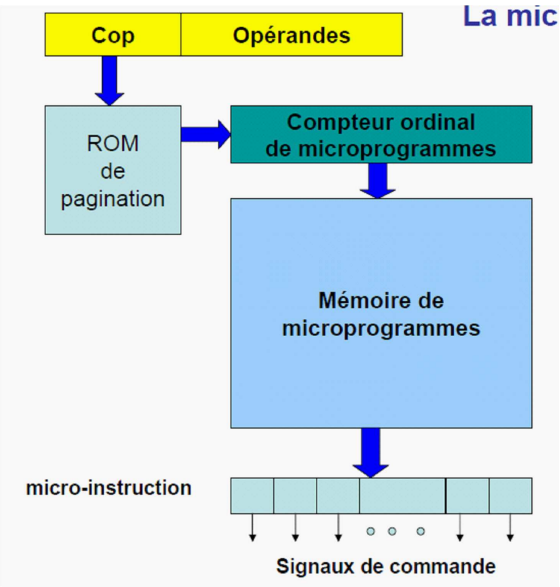


FIGURE IV.7 – Sequence à base de mémoire

IV.3 Fonctionnement d'un Microprocesseur

Le fonctionnement du microprocesseur (μP) se décompose en quatre étapes principales :

1. aller chercher le code d'opération de l'instruction à exécuter en mémoire → cycle de chargement ;

2. décoder le code opération pour déterminer l'action qu'il doit effectuer ;
3. exécuter l'opération → cycle d'exécution (charger la /les données, traitement, sauvegarder le résultat) ;
4. revenir à l'étape 1.

Chaque instruction est caractérisée par le nombre de périodes d'horloge (ou ***microcycles***) qu'elle utilise (donnée fournie par le fabricant du μP).
Exemple : horloge à 5 MHz, période T = $\frac{1}{f} = 0,2\mu s$. Si l'instruction s'exécute en 3 microcycles, la durée d'exécution de l'instruction est : 3 × 0,2 = 0, 6 μs.

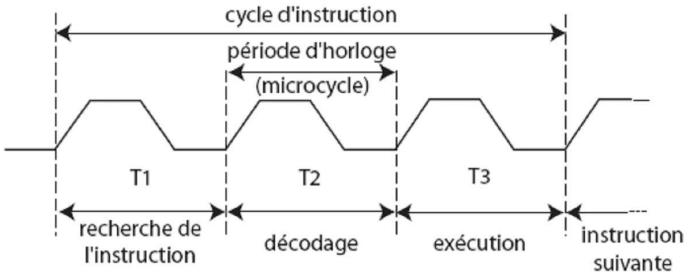


FIGURE IV.8 – Cycles du μP

Un microprocesseur exécute un programme. Le programme est constitué d'une suite d'instructions stockées dans la mémoire.

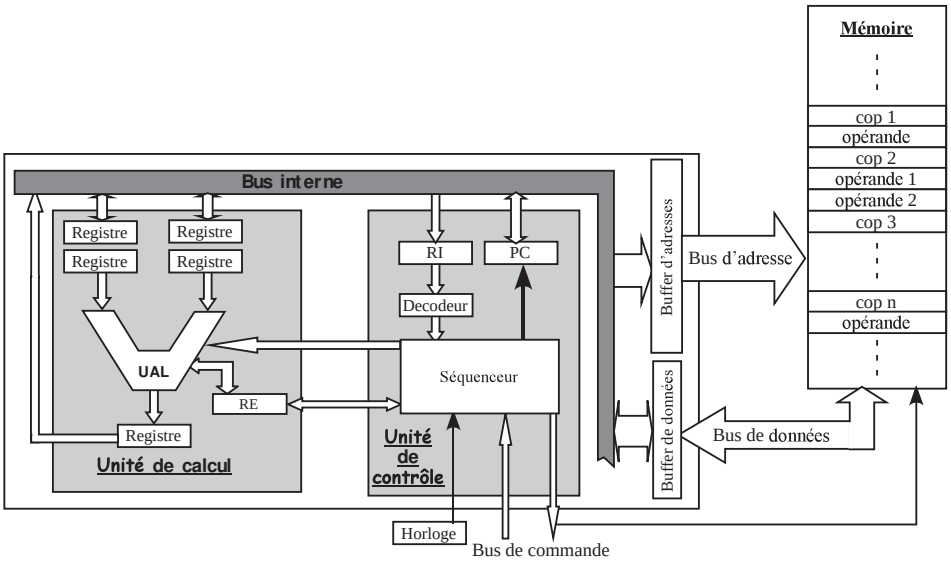
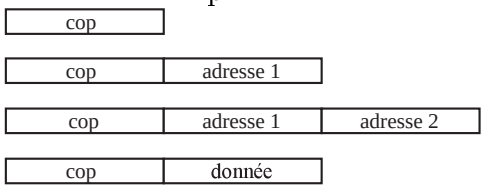


FIGURE IV.9 – fonctionnement du μP

Une instruction peut être codée sur un ou plusieurs octets.



$\Rightarrow$ **Le Mode d'Adressage**

Exemple : opération d'addition entre deux nombres
ADDA \$2000 $\Rightarrow$ instruction en langage assembleur
$\rightarrow$ effectuer l'addition entre le contenu du registre interne "**A**" et le contenu de la case mémoire [\$2000] puis le résultat est mis dans le registre interne "**A**" ($\mathbf{A} + [\$2000] \mapsto A$) !!

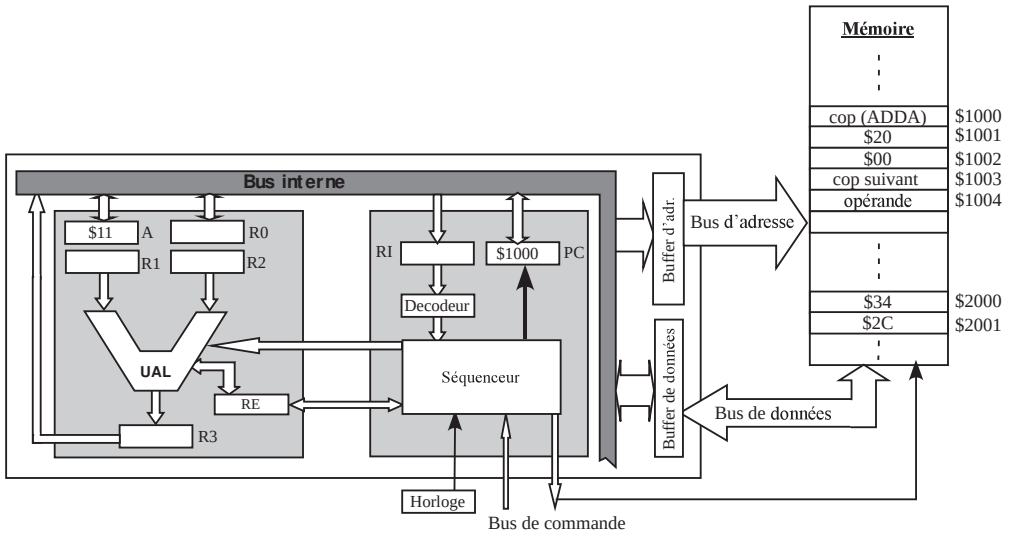


FIGURE IV.10 – Fonctionnement du μP

Alors, on a : ADDA \$2000 (A + [\$2000] $\mapsto$ A)
$\rightarrow$ on suppose que A = \$11 et PC = \$1000

Etape 1 : lecture du code Op. de l'instruction ;

Etape 2 : décoder le code Op. stocker dans **RI** ; puis le séquenceur va exécuter les micro-instructions !!

Etape 3 : charger les données A et [\$2000] !! ($\rightarrow$ 03 cycles !!)

Etape 4 : exécution de l'opération par l'UAL !!

Etape 5 : revenir à l'étape 1 (l'instruction suivante).

IV.3.1 Type d'architecture d'un microprocesseur

Architecture CISC (Complex Instruction Set Computer)

C'est une architecture avec un grand nombre d'instructions. Le processeur doit exécuter des tâches complexes par instruction unique. Donc, pour une tâche donnée, une machine

CISC exécute un petit nombre d'instructions mais chacun nécessite un plus grand nombre de cycles d'horloge (Intel 8086, Pentium..., Motorola 6809, 68000, PowerPC). Actuellement les deux technologies convergent : les processeurs CISC (Pentium par exemple) utilisent des instruction de plus en plus simples et exécutent parfois plusieurs instructions en un cycle d'horloge.

Architecture RISC (Reduced Instruction Set Computer)

Architecture dans la quelle les instructions sont en nombre réduit (chargement, branchement, appel sous-programme) et elle sont fréquemment utilisées. Le but est d'éliminer les instructions rarement employées et de consacrer les ressources matérielles à exécuter les instructions relativement simples en un cycle d'horloge et à émuler les autres instructions à l'aide des séquences basées sur les instructions élémentaires. On trouve donc une meilleure performance à une vitesse donnée (le gain en performance envisageable est important mais dépend de la qualité du compilateur). Processeurs RISC : PowerRISC (IBM/Motorola), SPARC (SUN), PA-RISC (HP) ; PIC de Microchip, AVR de Atmel, ...

IV.4 Les processeurs Spécialisés

IV.4.1 Micro-contrôleurs (μC)

Ils contiennent un CPU, de la RAM, de la ROM, quelques ports d'E/S parallèles, des ports séries, des compteurs programmables (timers), des CAN/CNA, des interfaces pour réseaux de terrain ... Ils sont en général utilisés pour contrôler des simples machines (appareils électroménagers, lecteurs de carte à puce...).

Exemple de circuits μC :

- 80C186XX (80186, 16 bits, Intel)
- 68HC11, 68HC12 (6809, 8 bits, Motorola)
- 68HC16 (68000, 16 bits, Mototrola)
- PIC 16F84, 16F877 (Microchip)

Par exemple le PIC 16F877A a comme caractéristiques architecturales :

✓ Architecture matériel de Harvard
✓ Architecture RISC des instructions

IV.4.2 Digital Signal Processor (DSP)

Ce sont des processeurs dédiés aux traitements des signaux numériques. Une architecture particulière leur permet un traitement efficace des fonctions complexes telles que FFT,

convolution, filtrage numérique ...
Exemples :

- TMS320 (Texas Instrument)
- 2100 et 21000 (Analog Device)
- 56000 (Motorola)
- PIC30F5011 (Microchip)

IV.4.3 Microprocesseurs sur mesure

Actuellement, la technologie permet de concevoir des microprocesseurs sur mesure pas l'utilisateur. Ceci est possible, en utilisant les circuits logiques programmables, communément appelés **FPGA** (**F**ield-**P**rogrammable **G**ate **A**rray : réseau de portes programmables in situ). Il existe plusieurs grands fabricants de FPGA comme : ***ALTERA***, ***XILINX*** et d'autre ...

Dans le domaine des systèmes numériques, il existe deux grandes sortes de composants :

- ► les processeurs (et dérivés : microcontrôleurs, DSP...) qui font du traitement séquentiel ;
- ► les composants programmables qui font du traitement parallèle.

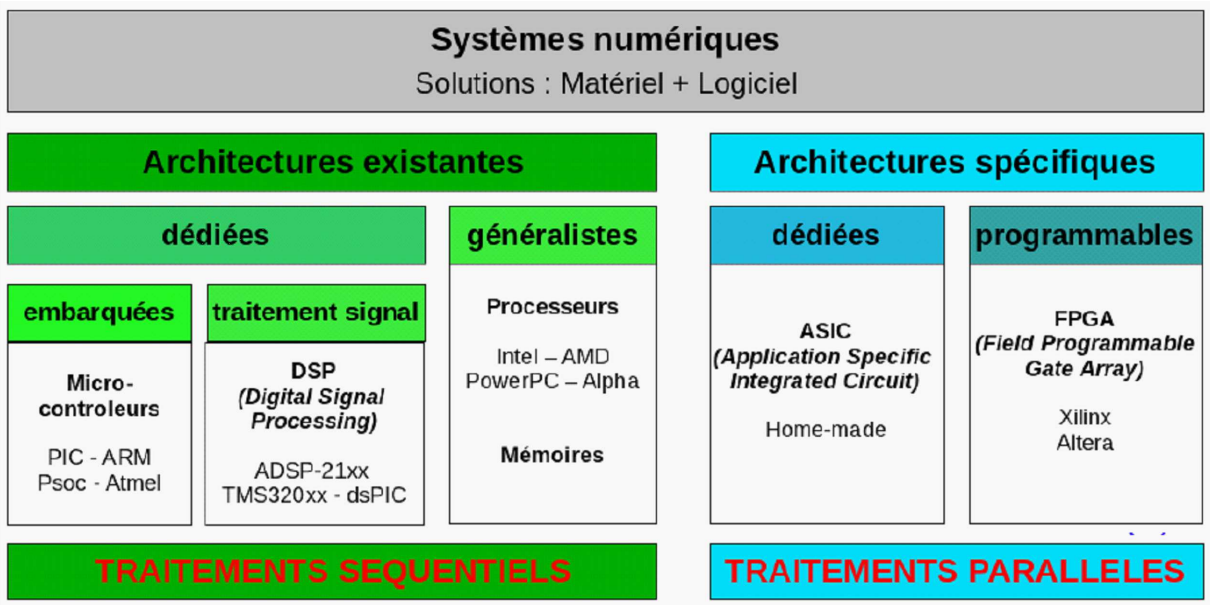


FIGURE IV.11 – Registre d'état du 6809S

Chapitre V

Etude d'un microprocesseur 8 bits : MC6809 de Motorola

V.1 Introduction

Le MC6809 est un microprocesseur 8 bits de Motorola. Il fut introduit vers 1977-1978. Ce microprocesseur fut une amélioration majeure par rapport à ses deux prédécesseurs : le 6800 et le 6802 de Motorola . L'organisation interne du MC6809 est orienté 16 bits, il permet un adressage mémoire de 64 Ko. Il existe deux version du microprocesseur, le 6809 et le 6809E (figure V.1). Leur brochage diffère ainsi que leurs signaux mais leur jeu d'instruction est identique.

FIGURE V.1 – Le MC6809

V.2 Boitier et brochage

Le μP 6809, fabriqué en technologie *MOS Canal N*, comporte 40 broches (figure V.2) et est du type mono-tension (5V).

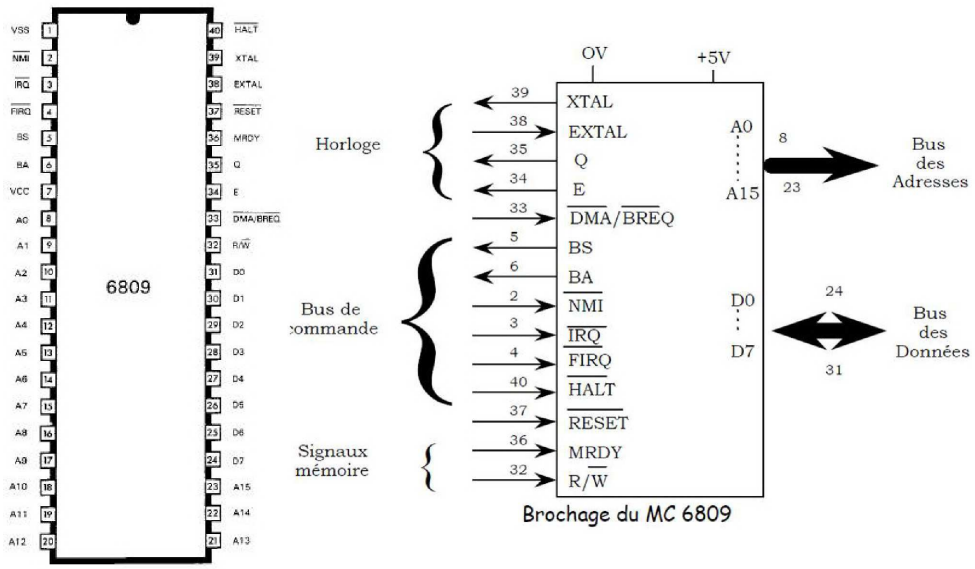


FIGURE V.2 – Brochage du MC6809

Il possède trois bus indépendants :

- Bus de données sur 8 bits (pin 24 à 31) bidirectionnel ;
- Bus d'adresse sur 16 bits (pin 8 à 23) unidirectionnel ;
- Bus de contrôle de 10 bits pour le 6809 et de 12 bits pour le 6809E.

Le bus de contrôle comporte les signaux suivants :

► $R/\overline{W}$ (Read/Write) détermine le sens de transfert des données ⇔ mémoire ;

► $\overline{MRDY}$ (Memory ready) permet l'allongement du cycle d'horloge pour les mémoire lentes (figure V.3) ;

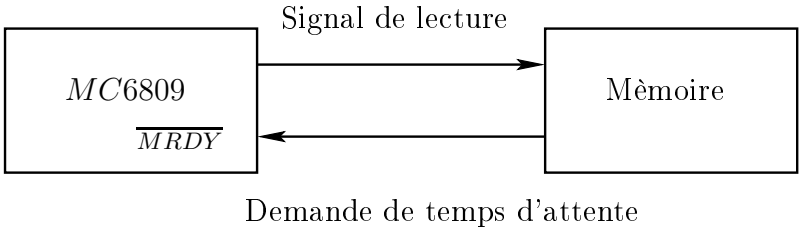


FIGURE V.3 – Dialogue avec mémoire lente

► BA (Bus available) et BS (Bus state) : Information qui permet de connaître l'état du microprocesseur à tout Moment.

BA	BS	Etat
0	0	Normal
0	1	reconnaissance d'interruption
1	0	reconnaissance de synchronisation externe
1	1	arrêt : bus disponible

TABLE V.1 – Signaux BA et BS

► Entrées d'horloge XTAL et EXTAL (Extension crystal) ;

La fréquence du quartz (horloge) est quatre fois la fréquence du μP (fréquence externe divisée par 4).

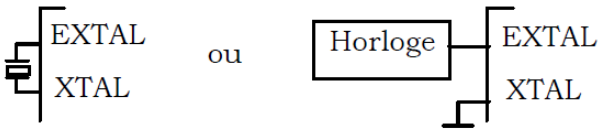


FIGURE V.4 – Horloge externe du MC6809

► E représente le signal d'horloge commun au système. Il permet la synchronisation du microprocesseur avec la périphérie.

► Q représente le signal d'horloge en quadrature avec E.

Remarque :

✓ Les données sont lues ou écrites sur le front descendant de E.

✓ Les adresses sont correctes à partir du front montant de Q.

► Les broches d'interruption :

- NMI (No Masquable Interrupt)
- IRQ (Interrupt Request)
- FIRQ (Fast Interrupt Request)

Entrées (actives sur un niveau bas) qui peuvent interrompre le fonctionnement normal du microprocesseur sur front descendant de Q.

► HALT (Arrêt du microprocesseur) :

Un niveau bas sur cette broche provoque l'arrêt du microprocesseur (mais à la fin de l'exécution de l'instruction en cours). Il n'y a pas perte des données. (BA = BS = 1)

► Broche d'initialisation RESET :

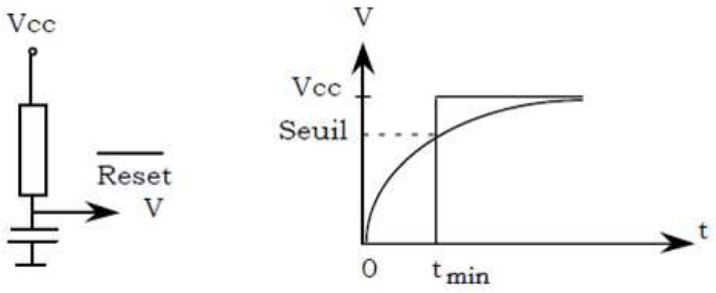


FIGURE V.5 – Le circuit RESET

Un niveau bas sur cette broche entraîne une réinitialisation complète du circuit. Pour être active cette ligne doit être maintenue à un niveau bas durant un temps suffisamment long (plusieurs cycles d'horloge). Le registre **PC** est initialisé avec le contenu des vecteurs d'initialisation qui se trouvent aux adresses $\$FFFE$ et $\$FFFF$. Ce contenu représente **l'adresse du début du programme principal (P.P.)** qui sera exécuté par le microprocesseur.

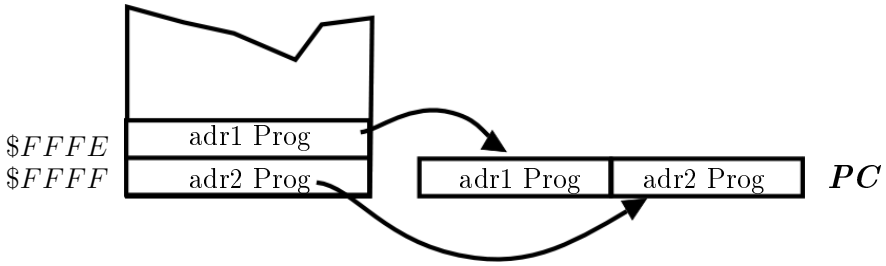


FIGURE V.6 – L'adresse du P.P.

V.2.1 Minimum pour démarrer

D'un point de vue matériel (figure V.7), le μP 6809 a besoin des éléments suivants pour fonctionner correctement :

- ✓ Alimentation VSS – VCC (0 – 5v)
- ✓ Circuit Reset
- ✓ Quartz pour l'horloge
- ✓ Mémoire ROM (programme) et RAM

V.2.2 Adressage de la mémoire

Le 6809 possède 16 lignes d'adresse ($A_0 \cdots A_{15}$). Il peut adresser un espace mémoire de 64Ko addressable entre \$0000 - $\$FFFF$ avec un adressage standard de type octet.

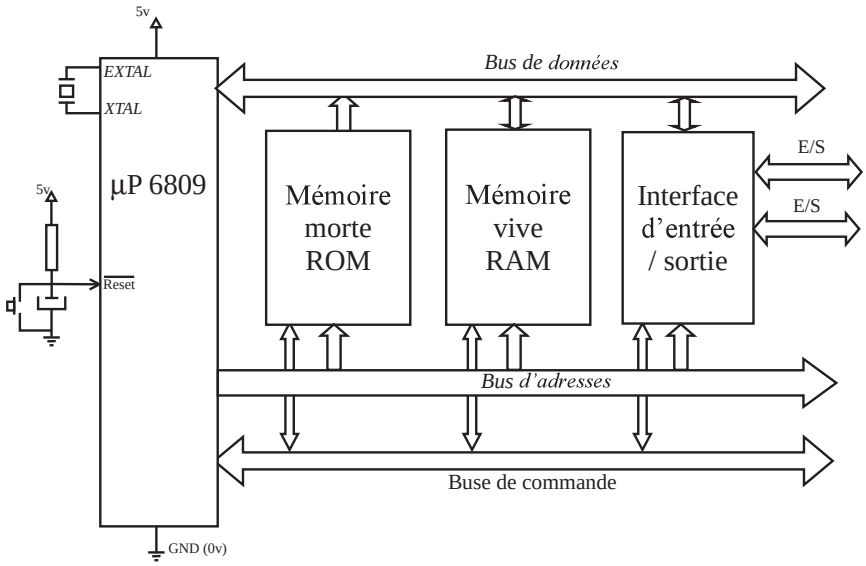
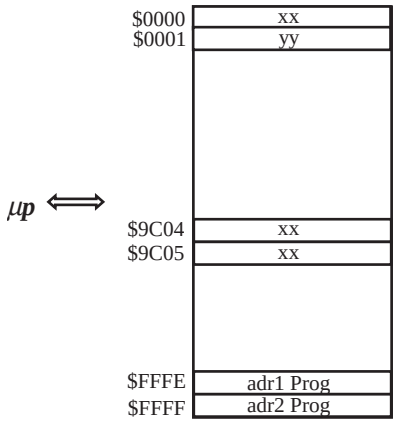


FIGURE V.7 – la configuration minimale matériel pour le 6809

Rem1 : Il est impérativement nécessaire que les adresses $\$FFFE$ - $\$FFFF$ doivent être incluses par une mémoire ***ROM***.

Rem2 : Dans la suite de ce cours, on va supposer les espaces mémoire suivants :

- ⋆ RAM → \$0000 - $\$03FF$
- ⋆ ROM → $\$FC00$ - $\$FFFF$



V.3 Architecture interne du 6809

La figure V.8 montre l'architecture simplifiée interne du MC6809.

V.3.1 Les registres internes du 6809

Le μP 6809 possède un certain nombre de registres internes, l'utilisateur (programmeur) a accès aux registres suivants (figure V.9).

∗ ***Les accumulateurs A, B ou D***

Les accumulateurs **A** et **B** sont des registres de calcul de 8 bits dans lesquels toutes les

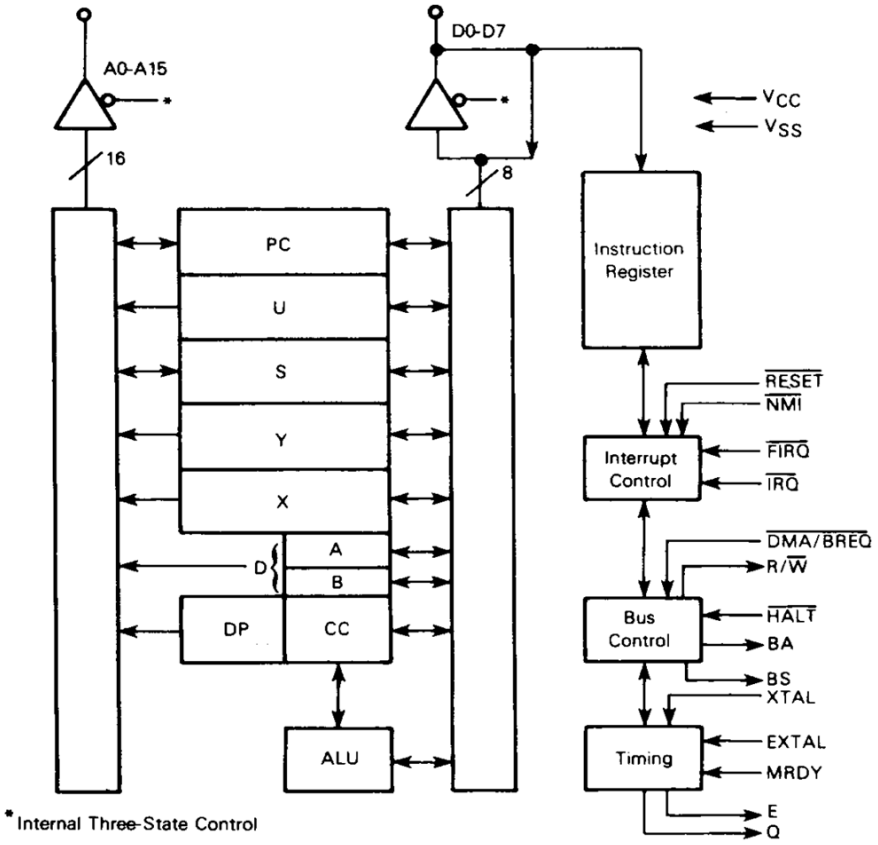


FIGURE V.8 – Architecture interne du 6809

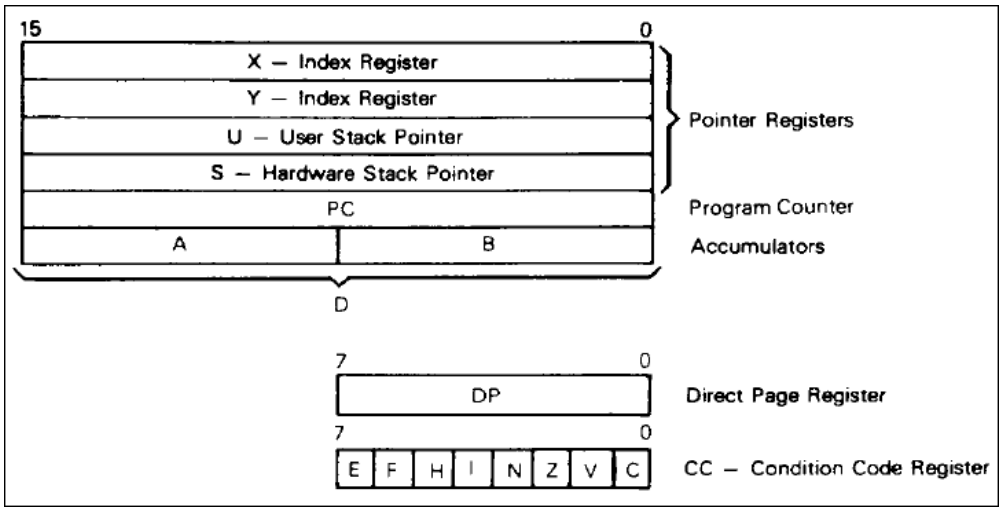


FIGURE V.9 – Les registres Internes du 6809

opérations arithmétiques et logiques peuvent s'effectuer.
Certaines instructions regroupent les registres A et B pour former un seul registre (accumulateur) **D** de 16 bits. Dans ce cas l'accumulateur **A** représente l'octet de poids fort et l'accumulateur **B** représente l'octet poids faible.

∗ ***Les registres d'index X et Y (registres 16 bits)***

Les registres X et Y sont dits registres d'index car ils permettent d'adresser tout l'espace mémoire avec en plus la capacité d'être pré-décrémenté ou poste-incrémenté.

∗ ***Les registres S et U (registres 16 bits)***

Les registres S et U sont des pointeurs de ***pile*** (zone mémoire réservée) (figure V.10) qui fonctionnent tous deux identiquement. Ils opèrent en mode dernier entré-premier sortie (**LIFO** : Last In – First Out).

Rôles de la pile système pour le μP : elle permet 3 facilités de programmation :
- ✓ les sous programmes
- ✓ les interruptions
- ✓ stockage de données temporaires

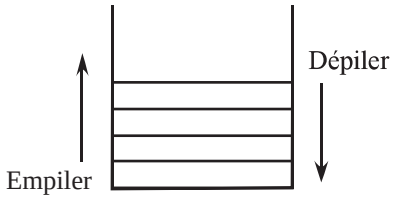


FIGURE V.10 – Le principe de la pile

⇒ le pointeur de pile **S** (Système) est utilisé automatiquement par le μP ;
⇒ le pointeur de pile **U** (Utilisateur) est géré par le programmeur.

∗ ***Le registre DP (Direct Page)***

C'est un registre 8 bits utilisé pour l'adressage direct court, il forme la partie haute de l'adresse. Il est automatiquement remis à 00 par un Reset.

∗ ***Le registre PC (Program Counter)***

C'est le registre (16 bits) qui contient toujours l'adresse mémoire où se trouve l'instruction devant être exécutée.

∗ ***Le registre de condition CC (CCR : Condition Code Register)***

C'est le registre d'état du 6809. Il comporte 8 bits jouant chacun un rôle important pour les instructions de sauts ou de branchements conditionnels (figure V.11).

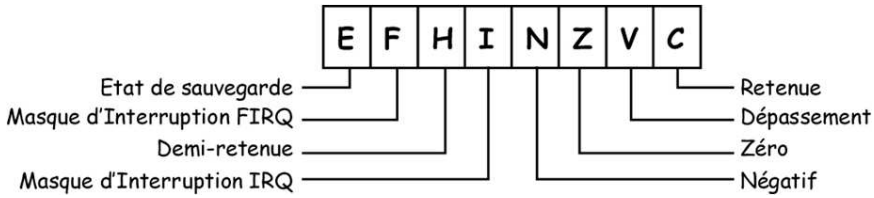


FIGURE V.11 – Registre d'état

- ✓ L'indicateur **C** (**Carry**) indique que l'opération effectuée a généré une retenue.
- ✓ L'indicateur **V** (**oVerflow**) est positionné à « 1 » s'il y'a dépassement de capacité suite à une opération sur des nombres codés en complément à 2.
- ✓ L'indicateur **Z** (**Zero**) indique que le résultat de l'opération effectuée est nul.
- ✓ Le bit **N** (**Negative**) indique que le résultat est négatif. C'est la copie du bit de poids fort du résultat de l'opération effectuée.
- ✓ L'indicateur **H** (**Half-Carry**) est un bit de demi-retenue. Entre le quartet faible et la quartet fort du registre A ou B.

Les bits **I**, **F** et **E** sont des bits système qui vont être introduits lors du chapitre des interruptions.

V.4 Langage assembleur

V.4.1 Langage machine

C'est le seul langage qui soit directement compréhensible par un μP. La longueur des instructions en code machine varie de 1 à 4 octets pour le 6809. Le premier octet constitue le ***code opération de l'instruction*** (COP).

V.4.2 Langage assembleur

C'est un langage symbolique du langage machine et qui est stocké sous forme de texte. Chaque instruction assembleur représente exactement une instruction machine. Le langage assembleur est plus compréhensible par l'être humain que le langage machine!!

Exemple : le mot ***ADD*** est un mnémonique pour l'instruction ***d'addition***, son code machine est le suivant :

1111 1011 0010 0000	$\$FB$ \$20 \$00
$\Rightarrow$	ADDB \$2000

La forme générale d'une instruction assembleur est composée de quatre (04) champs :

```
Etiquette:    Mnémonique Opérandes     * commentaire
```

Exemple :

```
Loop:    LDA #$5C          * instruction de chargement
```

ou

```
* instruction de chargement
Loop:
    LDA #$5C
```

V.5 Le jeu d'instruction du 6809

Le μP 6809 possède 59 instructions de base, et 9 modes d'adressage $\Rightarrow$ combinés, ***ils fournissent 1464 codes opératoires*** différents. Dans cette section, on présentera les instructions les plus utiles pour débuter la programmation en assembleur du μP 6809.

V.5.1 Instructions de chargement de données

Le 6809 possède trois types d'instruction de chargement :

* ***Instruction de chargement de registre***

LD (LOAD) : instruction de chargement des registres ;
LDA source $\Rightarrow$ charger le registre A avec la valeur de la source (8bits) ;
LDB source $\Rightarrow$ charger le registre B ;
LDD source $\Rightarrow$ charger les registres A et B (16 bits) ;
LDX, LDY, LDU et LDS $\rightarrow$ même opération mais la destination est un registre différent.

* ***Instruction de chargement mémoire (stockage)***

ST (STOR) : instruction de stockage vers la mémoire ;
STA adr $\Rightarrow$ copier le contenu du registre A vers la case mémoire de l'adresse « ***adr*** » ; même chose pour STB (8bits).
STD, STX, STY, STU, STS même opération mais la taille de transfert est de 16 bits.

∗ ***Instructions d'échange de registres***

► ***TFR*** R1, R2 ⇒ copier le contenu du registre R1 dans le registre R2 (R1 → $R2$) sans que le contenu de R1 change.

► ***EXG*** R1,R2 ⇒ permet d'échanger le contenu des deux registres ($R1 \leftrightarrow R2$).

Exemple 1 :

Avant exécution, on a : $A = \$10$; $DP = \$5A$

TFR A,DP	∗	$A \mapsto DP \Rightarrow$ après : $A = \$10$; $DP = \$10$

Avant exécution : $U = \$03FF$; $D = \$22CC$

TFR U,D	∗	$U \mapsto D \Rightarrow$ après : $U = \$03FF$; $D = \$03FF$

Exemple 2 :

EXG B,A	∗	Avant exécution : $A = \$41$; $B = \$0A \Rightarrow$ après : $A = \$0A$; $B = \$41$

Rem : R1 et R2 doivent avoir la même taille (8 bits ou 16 bits) ! !

✓ Les registres 8 bits : A, B, CC ou DP

✓ Les registres 16 bits : X, Y, U, S, D ou PC

V.5.2 Les modes d'adressage du 6809

On désigne par mode d'adressage, la manière d'accéder à un opérande ⇒ c'est la manière de définir la donnée ou l'adresse au niveau de l'instruction.

V.5.2.1 Adressage immédiat

L'opérande est exprimé directement par sa valeur. Le symbole ♯ (***dièse***) identifie le mode d'adressage immédiat.

Exemple :

LDA	#$47	∗ ⇒ charger A avec la valeur $47 ↦ ($A = \$47$)
LDY	#$A200	∗ ⇒ charger Y avec la valeur $A200 ↦ ($Y = \$A200$)

V.5.2.2 Adressage implicite (ou registre)

L'adressage implicite est utilisé par les instructions qui agissent seulement sur les registres internes du microprocesseur ;

Exemple :

INCA $\Rightarrow (A + 1 \mapsto A)$: pas d'opérande (taille instruction 1 octet : $\$4C$) ;
TFR CC,A $\Rightarrow$ pas d'adresse, ni de donnée immédiate, mais l'instruction est composée de 2 octets !! (cop + Post-octet). le Post-octet représente les codes « source/déstination » $\Rightarrow$ dans ce cas on le code machine suivant : $\$1F\$A8$

TABLE V.2 – Tableau pour le codage de l'octet post-octet

Code	Registre
0000	D
0001	X
0010	Y
0011	U
0100	S
0101	PC
1000	A
1001	B
1010	CCR
1011	DP

V.5.2.3 Adressage absolu

L'opérande est désigné par **une adresse** qui indique son emplacement en mémoire ; le défaut de symbole exprime ce mode. Il y a deux types :

* ***L'adressage absolu étendu (adressage direct long)***

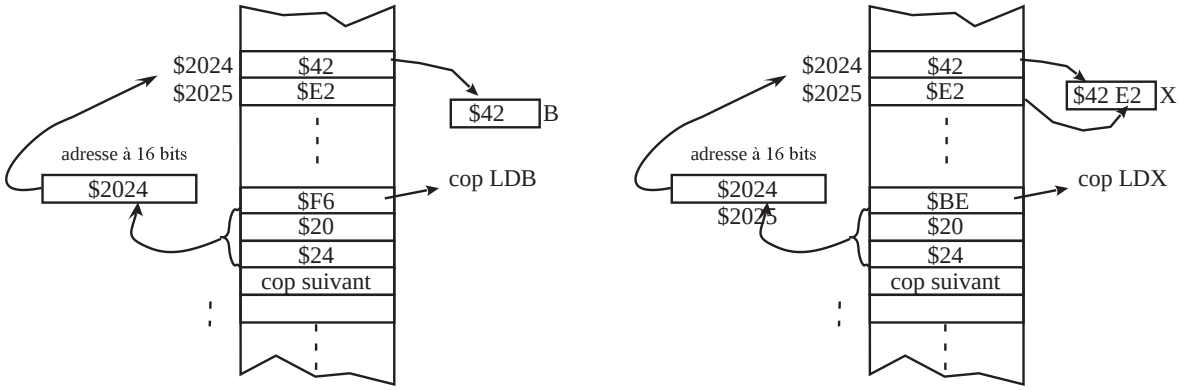
LDB $2024	* $\Rightarrow$ charger B avec le contenu de l'adresse mémoire [$2024]

LDX $2024	* $\Rightarrow$ charger X avec le contenu de la mémoire [$2024] et [$2025] (16 bits)

* ***L'adressage absolu direct (adressage direct court)***

L'octet de poids faible de l'adresse est indiqué au niveau de l'instruction, l'octet poids fort se trouve dans le registre **DP** (*Direct Page*) ;

LDA $12	
STD $25	* avec $DP = \$2A$, $A = \$34$ et $B = \$5F$!!



(a) registre 8bits (b) registre 16bits

FIGURE V.12 – Exemple d'adressage absolu étendu

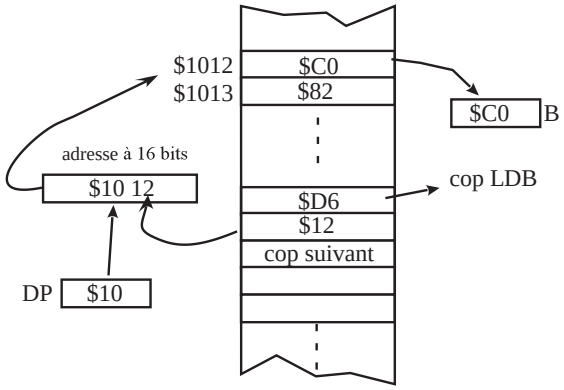


FIGURE V.13 – Exemple d'adressage absolu direct

V.5.2.4 Adressage indexé simple (indirect)

Dans ce mode d'adressage, un registre d'index de 16 bits est utilisé pour indiquer l'adresse de l'opérande c-à-d que le registre d'index indique l'adresse de l'opérande au niveau de la mémoire.

Exemple :

$X = \$1012$, $A = \$00$

LDA	,X	*	figure V.14

Exercice : $Y = \$4002$, $A = \$A5$ et $B = \$5A$

STD	,Y	*	quel est le résultat de cette instruction ?

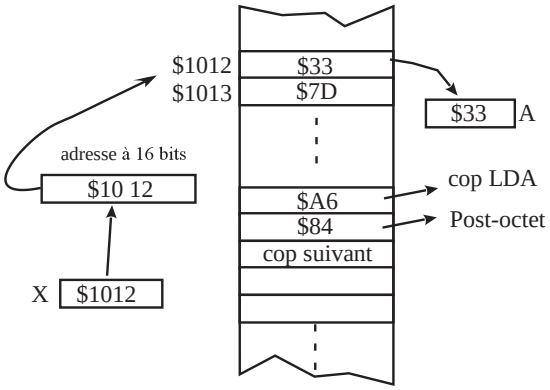


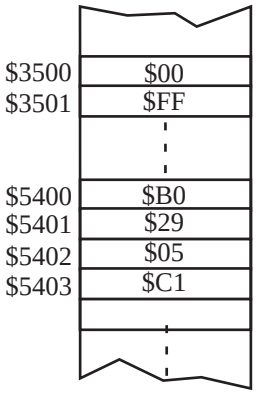
FIGURE V.14 – exemple d'adressage indexé simple

V.5.3 Affectation des indicateur

Les résultats d'exécution des instruction affectent les indicateurs du registre d'état CC.
Exemple : on suppose l'état initial suivant :

* $A = \$00, B = \$FF$
* $X = \$00FF$
* $Y = \$FF00$

	H	N	Z	V	C	Résultat
Avant exécution	0	0	0	0	0	
Après exécution de :						
LDB $3500	0	0	1	0	0	B=$00
Après						
LDY $5400	0	1	0	0	0	Y=$$B029$
Après						
STA ,X	0	0	1	0	0	[$$FF00$]=$00



V.5.4 Calcul Arithmétique et Logique

V.5.4.1 Les instructions Arithmétiques

* ***Les instructions d'addition***

  - **ADDA M** $\Rightarrow$ Registre $A + M \mapsto A$! !

Cette instruction permet l'addition du contenue du registre **A** avec la valeur désignée par **M** (selon le mode d'adressage de **M**). Le résultat est retourné dans le registre **A**.

Exemple : avant execution le registre $A = \$04$

ADDA	#$80	*	Registre $A + M \mapsto A \Rightarrow$ après : $A = \$04 + \$80 = \$84$

avant execution le registre $A = \$04$

```
ADDA    $0080   * RegistreA+[$0080] ↦ A
        *  ⇒ après : A = $04 + $30 = $34
```

- **ADDB M :** même rôle que ADDA mais avec le registre B $\Rightarrow RegistreB + M \mapsto B$.
- **ADDD MM :** même rôle que ADDA mais avec le registre D (16 bits) $\Rightarrow RegistreD + MM(16bits) \mapsto D$.

Exemple :

avant execution le registre $D = \$8105$

```
ADDD    $0100    * RegistreD + [$0100 : $0101] ↦ D
                 *  ⇒ après : D = $8105 + $9045 = $114A
```

- **ADCA M** $\Rightarrow RegistreA + M + bitC \mapsto A$.
- **ADCB M** $\Rightarrow RegistreB + M + bitC \mapsto B$.
- **ABX** $\Rightarrow$ instruction particulière $\Rightarrow RegistreB + RegistreX \mapsto X$.

* ***Les instructions de soustraction***

- **SUBA M** $\Rightarrow RegistreA - M \mapsto A$ (format sur 8 bits).
- **SUBB M** $\Rightarrow RegistreB - M \mapsto B$ (format sur 8 bits).
- **SUBD MM** $\Rightarrow RegistreD - MM \mapsto D$ (format sur 16 bits).

- **SBCA M** $\Rightarrow RegistreA - M - bitC \mapsto A$.
- **SBCB M** $\Rightarrow RegistreB - M - bitC \mapsto B$.

* ***Instruction de multiplication***

- **MUL** $\Rightarrow$ Registre $A \times B \mapsto D$!!

Remarque : on considère que les nombres sont non signés !!

Exemple : Écrire un programme qui effectue le produit $[P = a \times b]$; avec a et b sont deux nombres non signés codés sur 8 bits et stockés en mémoire à l'adresse \$0100 et \$0101 respectivement. Le résultat (P) doit être stocké à partir de l'adresse \$0200.

```
Debut :
    LDA     $0100    *  a ↦ A
    LDB     $0101    *  b ↦ B
    MUL              *  A × B ↦ D ⇒ [A : B] (16 bits)
    STD     $0200
    END
```

∗ ***Les instructions d'incrémentation***

- **INCA** $\Rightarrow A+1 \mapsto A$ (les indicateurs $[NZV]$ sont affectés).
- **INCB** $\Rightarrow B+1 \mapsto B \Leftrightarrow$ **ADDB** ♯\$01.
- **INC adr** $\Rightarrow [adr]+1 \mapsto adr$ (une case mémoire).

∗ ***Les instructions de décrémentation***

- **DECA** $\Rightarrow A-1 \mapsto A \Leftrightarrow$ **SUBA** ♯\$01 .
- **DECB** $\Rightarrow B-1 \mapsto B$ (les indicateurs $[NZV]$ sont affectés).
- **DEC adr** $\Rightarrow [adr]-1 \mapsto adr$ (une case mémoire).

∗ ***Les instructions de décalage arithmétique***

- **ASLA** $\Rightarrow$ les bits du registre A sont *décalés à gauche* d'une place. Le MSB du registre A est décalé vers le bitC.
- **ASLB** $\Rightarrow$ même chose avec le registre B.
- **ASL adr** $\Rightarrow bitC \leftarrow\!- [\overleftarrow{adr}] \leftarrow\!- 0$
- **ASRA** $\Rightarrow$ *décalage à droite* du registre A.
- **ASRB** $\Rightarrow$ *décalage à droite* du registre B.
- **ASR adr** $\Rightarrow [\overrightarrow{adr}] \dashrightarrow bitC$

Remarque : Le MSB est conservé!!

∗ ***Les instructions de négation*** (complément à 2)

- **NEGA** $\Rightarrow \overline{A}+1 \mapsto A$.
- **NEGB** $\Rightarrow \overline{B}+1 \mapsto B$.
- **NEG adr** $\Rightarrow \overline{[adr]}+1 \mapsto adr$

Exercice : Ecrire un programme en langage assembleur pour 6809 qui effectue l'addition de trois nombres $[S = a+b+c]$; avec a, b et c sont des nombres non signés codés sur 8 bits et stockés aux adresses \$0080, \$0081 et \$0082 respectivement. Le résultat S (16 bits) doit être stocké à partir de \$0100.

Solution 1 :

```
Debut :
    LDA     #$00     *  ⇔ CLRA ; $00 ↦ A
    LDB     $0080    *  ⇔ LDB $80 avec DP = $00
    ADDB    $81      *  ⇔ ADDB $0081 avec DP = $00
    ADCA    #$00
```

```
ADDB    $82
ADCA    #$00
STD     $0100
END
```

Solution 2 : extension de format 8 bits → 16 bits !

```
Debut :
    CLRA            *  $00 ↦ A
    LDB     $0081
    STD     $0200
    LDB     $0082
    STD     $0202
    LDB     $0080
    ADDD    $0200
    ADDD    $0202
    STD     $0100
    END
```

V.5.4.2 Les instructions logiques

* ***L'opérateur "ET" logique*** (opération bit/bit)

```
ANDA    M
ANDB    M
ANDCC   #m
```

Exemple :

```
LDA     #%00011101    *  $1D ↦ A
ANDA    #%10010001    *  A · (10010001) = (00010001) ↦ A
```

* ***L'opérateur "OU" logique*** (opération bit/bit)

```
ORA     M
ORB     M
ORCC    #m
```

* ***Les instructions de complément à 1***

```
COMA            *  ⇒ Ā ↦ A
COMB            *  ⇒ B̄ ↦ B
COM     adr     *  ⇒ [adr] ↦ adr
```

* ***Les instructions de décalage logique***

LSL, LSLA, LSLB : décalage à gauche $\Leftrightarrow$ ASL, ASLA, ASLB.

LSR, LSRA, LSRB : décalage à droite $\Rightarrow$ 0 --→ $[\overrightarrow{valeur}]$ --→ $bitC$.

* ***Les instructions de rotation***

ROL, ROLA, ROLB $\Rightarrow$ rotation à gauche avec le bitC ;

ROR, RORA, RORB $\Rightarrow$ rotation à droite avec le bitC ;

* ***Les instructions de test de bit***

```
BITA   M     *  ⇔ ANDA M mais le résultat n'est pas conservé
             *  juste pour l'affectation des indicateurs N et Z
BITB   M     *  ⇔ ANDB M
TSTA         *  ⇔ A - 00 : pour l'affectation des indicateurs N et Z
TSTB         *  ⇔ B - 00 : pour l'affectation des indicateurs N et Z
TST    adr   *  ⇔ [adr] - 00 : pour l'affectation des indicateurs N et Z
```

V.6 Les directives en assembleur

Une directive est une pseudo-instruction destinée à l'assembleur (le programme traducteur du langage assembleur vers le langage machine). C'est une commande spéciale que le programmateur utilise pour effectuer une initialisation de données en mémoire, déclaration de symboles, ...etc.

► ***Directive origine : ORG***

La pseudo-instruction ORG indique à l'assembleur l'adresse mémoire à partir de la quelle il faut mettre le code machine des instructions qui suivent.

Exemple :

```
ORG  $FD00
LDB  #$05    *  le code machine de cette instruction sera situé en mémoire
             *  à partir de l'adresse $FD00 (voir tableau V.3)
STB  $0110
```

► ***Directive fin de fichier : END***

Il y'a une seule directive END dans un fichier qui indique la fin du texte source. Elle doit impérativement terminer la suite des instructions du programme.

► ***Directive d'équivalence : EQU***

Cette directive (EQU) est utilisée pour affecter une valeur à un symbole.

Exemple :

TABLE V.3 – Programme assembleur et code machine

Adresses	code Op.	Assembleur	commentaires
		ORG $FD00	directive d'origine
$FD00	$C6	LDB ♯$05	$B = \$05$
$FD01	$05		
$FD02	$F7	STB $0110	$B \mapsto \$0110$
$FD03	$01		
$FD04	$10		

```
ORG     $FC00
LDB     #$44
LDX     #$0100
STB     $0200
END
        * d'une autre manière ⇝
```

```
Debut   EQU   $FC00
val1    EQU   $44
val2    EQU   $0100
adr     EQU   $0200

ORG     Debut
LDB     #val1
LDX     #val2
STB     adr
END
```

► ***Directive d'initialisation mémoire : DB (define Byte)***

La directive DB permet d'initialiser des cases mémoires par des données. Cette pseudo-instruction respecte la syntaxe suivante :

```
;adresse  DB  valeur (1 octet)

;$00A0  DB  $52
;$00A1  DB  $C8
;$00A2  DB  $04
```

$009F	
$00A0	$52
$00A1	$C8
$00A2	$04
$00A3	

► ***Les commentaires***

Un commentaire est une chaîne de caractères ignorée par l'assembleur si elle est précédée par une étoile (∗).

V.7 Les instructions de comparaison et de branchement

Exemple :

Établir un programme en assembleur 6809 qui effectue le calcul de la somme 100 nombres signés (codés sur 8 bits) ($S = \sum_{i=0}^{99} n_i$). Les nombres signés (n_i) sont rangés

dans une table de 100 cases mémoires située à l'adresse \$0200. Le résultat (S) doit être ranger à partir de l'adresse \$0300.

Remarque :

n_i est un nombre signé et codé sur 8 bits $\Rightarrow n_i \in [+127 \cdots -128]$.

Le maximum de $S = 100 \times 128 = 12800 = \$3200 \Rightarrow S$ est codé sur 16 bits ($S \in [+32767 \cdots -32768]$).

Solution 1 :

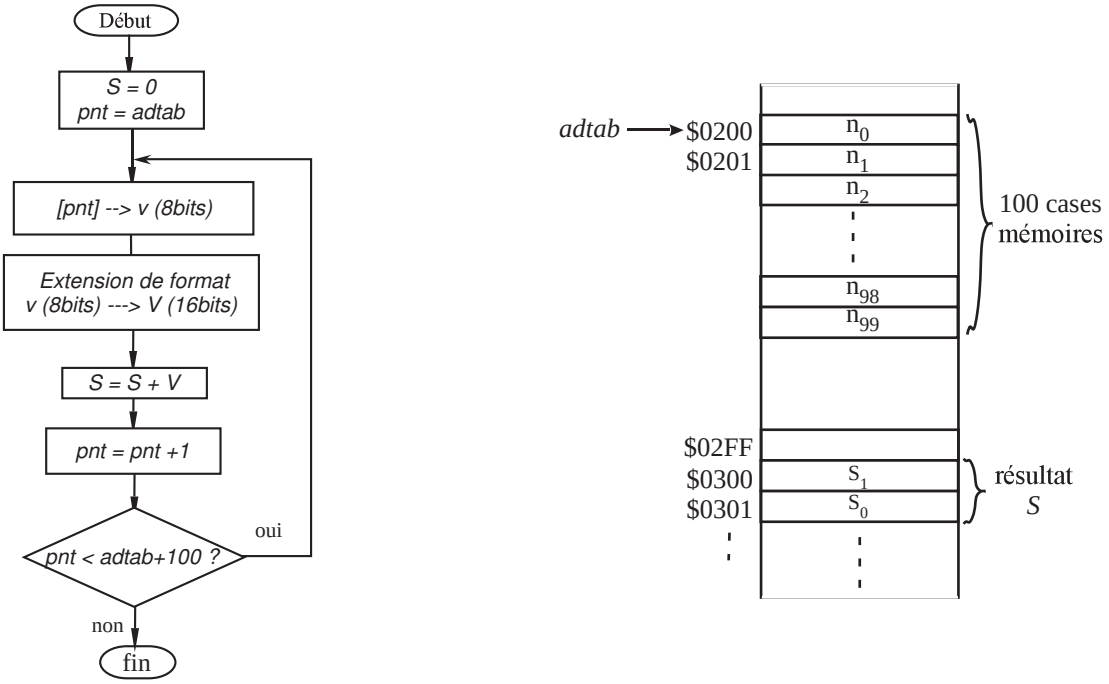


FIGURE V.15 – L'organigramme de la solution 1

***L'instruction* d'extension de signe** : *SEX B --→ A* (8 bits --→ 16 bits), l'instruction permet de recopier le bit poids fort du registre B vers tous les bits du registre A ; c'est-à-dire :

si MSB de B = 0 alors A =\$00
si MSB de B = 1 alors A =\$*FF*

Le programme :

```
adtab   EQU  $0200
adfin   EQU  $0264

    ORG    $FC00
    CLR    $0300      * S = 0
```

```
    CLR   $0301
    LDX   #adtab      *   X = $0200 ⇒ pnt = adtab
boucle:
    LDB   ,X+         *   Registre B = n_i puis pnt = pnt + 1
    SEX               *   extension de signe : Registre D = n_i
    ADDD  $0300
    STD   $0300
    CMPX  #adfin
    BLO   boucle
    END
```

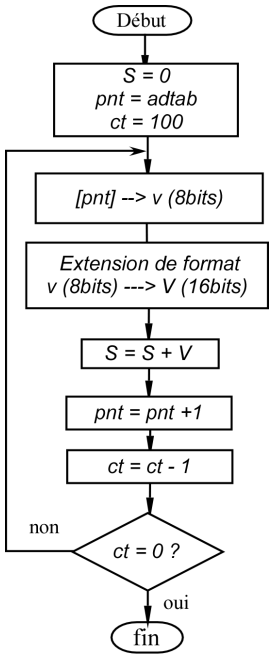
Solution 2 :

Le programme :

```
somme   EQU  $0300
adtab   EQU  $0200
ct      EQU  $0310

        ORG   $FC00
        CLRA
        CLRB
        STD   somme
        LDX   #adtab
        LDU   #ct
        LDB   #$64
        STB   ,U
loop:
        LDB   ,X+       *   Registre B = n_i
        SEX             *   extension de signe : Registre D = n_i
        ADDD  somme
        STD   somme
        DEC   ,U
        BNE   loop
        END
```

L'organigramme :



V.7.1 Les instructions de comparaison

- **CMPA M** ⇒ A - M : ça permet l'affectation des indicateurs mais le résultat n'est pas gardé!!

Exemple : $\$42 - \$51 = \$F1$ --→ $bitC = 1$

	N	Z	V	C
Avant : A =$42	0	0	0	0
CMPA ♯$51				
Après : A =$42	1	0	0	1

- **CMPB M** ⇒ B - M : même opération mais sur le registre B (8 bits) ;
- **CMPD MM** ⇒ D - MM : même opération mais sur le registre D (16 bits) ;
- **CMPS, CMPX, CMPY, CMPU** ⇒ sur les registres d'index !!

Remarque : l'instruction **TSTA** ⇔ **CMPA** ♯$00 !!

V.7.2 les Instructions de branchement (saut)

∗ ***Saut inconditionnel : GOTO***

- **JMP adr** (adresse effective) ;
- **BRA Etiquette** (adresse relative) ⇒ différence au niveau du code machine

Exemple :

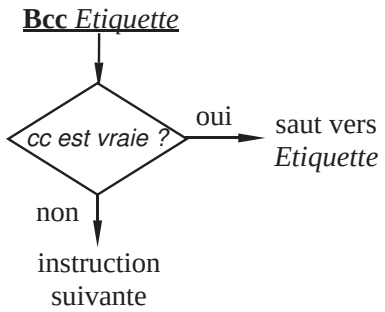
```
        ⋮
        JMP     $FD10
        STA     $0010
        ⋮
$FD10   STB     $0200
```

```
boucle: LDA     #$30
        ADDA    #$05
        LDX     #$0400
        STA     ,X
        ⋮
        BRA     boucle   * boucle représente
en code machine le nombre d'octets à sauter !!
```

∗ ***Saut conditionnel***

Les instructions de cette catégorie ont un mode d'execution qui dépend d'une condition.

- **Bcc** *Etiquette* ⇒ ***cc*** représente la condition



Les différentes conditions de branchement pour l'instruction ***Bcc*** sont regroupés dans le tableau V.4.

V.8 Mode d'adressage indexé poste-incrémenté et pré-décrémenté

Dans ce mode d'adressage, on utilise un registre d'index (Registre X, Y ou même S ou U).

TABLE V.4 – table des conditions cc

cc	signification	calcul pour le test
VS	Si dépassement	$V = 1$
VC	Si pas dépassement	$V = 0$
CC	Si retenue à 0	$C = 0$
CS	Si retenue à 1	$C = 1$
PL	Si positif	$N = 0$
MI	Si négatif	$N = 1$
EQ	Si égal	$Z = 1$
NE	Si non égal	$Z = 0$
GE	Si supérieur ou égal	$NV + \overline{N}\,\overline{V} = 1$
GT	Si supérieur	$NV\overline{Z} + \overline{N}\,\overline{V}\,\overline{Z} = 1$
HI	Plus grand	$\overline{C}\,\overline{Z} = 1$
HS	Plus grand ou égal	$C = 0$
LE	Si inférieur ou égal	$Z + \overline{N}V + N\overline{V} = 1$
LT	Si inférieur	$\overline{N}V + N\overline{V} = 1$
LS	Si plus petit ou égal	$C + Z = 1$
LO	Si plus petit	$C = 1$

V.8.1 Indexé poste-incrémenté

Exemple 1 :

on suppose que $Y = \$0300$

```
    ⋮
LDA  ,Y+
    ⋮
```

$0300	$41
$0301	$B8
$0302	

1. le registre A est chargé avec la valeur de la case mémoire pointée par $Y \Rightarrow A = \$41$;
2. **puis** le registre Y est incrémenté $(+1) \Rightarrow Y = Y + 1 = \$0301$;

$\Rightarrow$ deux actions : charger le registre A (8 bits) puis incrémenter l'index Y par $(+1)$.

Exemple 2 :

on suppose que $X = \$1000$

```
    ⋮
LDD  ,X++
    ⋮
```

$1000	$F1
$1001	$05
$1002	$3C

1. le registre D est chargé avec la valeur des deux cases mémoires pointées par $X \Rightarrow D = \$F105$;
2. **puis** le registre X est incrémenté (+2) $\Rightarrow X = X + 2 = \$1002$;

$\Rightarrow$ deux actions : charger le registre D (16 bits) puis incrémenter l'index X par (+2).

V.8.2 Indexé pré-décrémenté

Exemple 1 :

on suppose que $U = \$0200$

```
      ⋮
LDB  ,–U
      ⋮
```

$\$01FE$	\$21
$\$01FF$	\$01
\$0200	$\$A5$

1. le registre U est décrémenté (-1) $\Rightarrow U = U - 1 = \$01FF$;
2. **puis** le registre B est chargé avec la valeur de la case mémoire pointée par $U \Rightarrow B = \$01$;

$\Rightarrow$ deux actions : décrémenter l'index U par (-1) puis charger le registre B (8 bits).

Exemple 2 :

on a $Y = \$3000$ et $D = \$FFD3$

```
      ⋮
STD  ,––Y
      ⋮
```

$\$2FFD$	
$\$2FFE$	\$10
$\$2FFF$	\$58
\$3000	$\$3C$

1. le registre Y est décrémenté (-2) : $Y = Y - 2 = \$2FFE$;
2. **puis** stocker le contenu du registre D dans les deux cases mémoires pointées par $Y \Rightarrow$

$\$2FFD$	
$\$2FFE$	$\$FF$
$\$2FFF$	$\$D3$
\$3000	$\$3C$

$\Rightarrow$ deux actions : décrémenter l'index Y par (-2) puis stocker le registre D (16 bits) en mémoire.

V.9 Les Sous Programmes (S.P.)

Exemple : Soit à calculer $R = (a^2 - a + 1) + (b^2 - b + 1) + (c^2 - c + 1)$ avec a, b et c sont des nombres non signés, codés sur 8 bits et se trouvant aux adresses mémoires respectives : *ada*, *adb* et *adc*.
On a a, b et c codés sur 8 bits $\Rightarrow$ a^2, b^2 et c^2 sont codés sur 16 bits $\Rightarrow$ R est codé sur 18 bits, donc il faut prévoir un format de 32 bits pour le résultat R.
On réalise un sous programme (S.P.) qui calcul $(n^2 - n + 1)$ avec les considérations suivantes :

- quand le S.P. est appelé, on doit trouver la valeur (n) dans le registre B (8 bits) ;
- le S.P. retourne le résultat dans le registre D (16 bits).

```
adcalcul  EQU  $FC80

    ORG    $FC80
calcul:                 * Début du sous programmes (S.P.)
    TFR  B, A           * A = n; B = n
    DECB                * B = n - 1
    MUL                 * D = A x B = n x (n - 1) = n^2 - n
    ADDD #$0001         * D = n^2 - n + 1

    RTS                 * instruction de retour vers le programme appelant !!
```

on va écrire le programme principal qui va utiliser le S.P. ***calcul*** pour calculer la formule R :

```
ORG    $FC00
    LDS    #$0400
    CLRA
    CLRB
    STD    $1000        * R = 0
    STD    $1002
    LDB    ada          * B = a
    JSR    adcalcul     * saut vers le sous programme qui se trouve à l'adresse adcalcul

    STD    $1002        * R = (a^2 - a + 1)
    LDB    adb          * B = b
    JSR    adcalcul     * saut vers S.P. pour calculer (b^2 - b + 1)

    ADDD   $1002        * R = (a^2 - a + 1) + (b^2 - b + 1)
    BCC    suite1       * addition sur 16 bits alors si retenue => ajouter 1 au 17^me bit !!
    INC    $1001
suite1:
    STD    $1002
    LDB    adc          * B = c
    BSR    calcul       * saut vers S.P. pour calculer (c^2 - c + 1)
```

```
    ADDD   $1002       * R = (a²−a+1)+(b²−b+1)+(c²−c+1)
    BCC    suite2
    INC    $1001
suite2:
    STD    $1002

    END
```

V.9.1 Définition

un sous programme est une séquence d'instructions qui est écrite une seule fois en mémoire mais qui peut être exécutée plusieurs fois dans un même programme grâce à des appels différents de cette séquence.

V.9.2 Appel et retour de S.P.

a) Appel : il s'effectue grâce aux deux instructions **JSR** ou **BSR**.

On écrit :

JSR adSP * adressage absolu

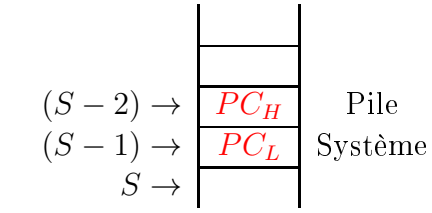
BSR Etiquette * adressage relatif cours (8bits signé)

LBSR Etiquette * adressage relatif long (16 bits signé)

Important Lorsque le μP rencontre l'instruction **JSR** (ou **BSR**), il sauvegarde le contenu du registre **PC** (compteur de programme) dans la pile système pointée par le registre **S** ; puis il se branche à l'adresse spécifiée "$adSP$" (étiquette).

on a donc les opérations suivantes :

$$\left\| \begin{array}{l} S-1 \rightarrow S, PC_L \rightarrow [S] \\ S-1 \rightarrow S, PC_H \rightarrow [S] \\ adSP \rightarrow PC \end{array} \right.$$



$\Rightarrow$ Lors d'un appel de S.P. le registre S est décrémenté par le μP de deux unités.

b) Retour Lorsque le μP rencontre l'instruction **RTS** ; il va chercher l'adresse de retour qui se trouve dans la pile système :

$$\left\| \begin{array}{ll} [S] \rightarrow PC_H & ; \text{puis } S+1 \rightarrow S \\ [S] \rightarrow PC_L & ; \text{puis } S+1 \rightarrow S \end{array} \right.$$

⇒ Lors de l'exécution de l'instruction **RTS**, le pointeur de pile système S est incrémenté par le μP de deux unités.

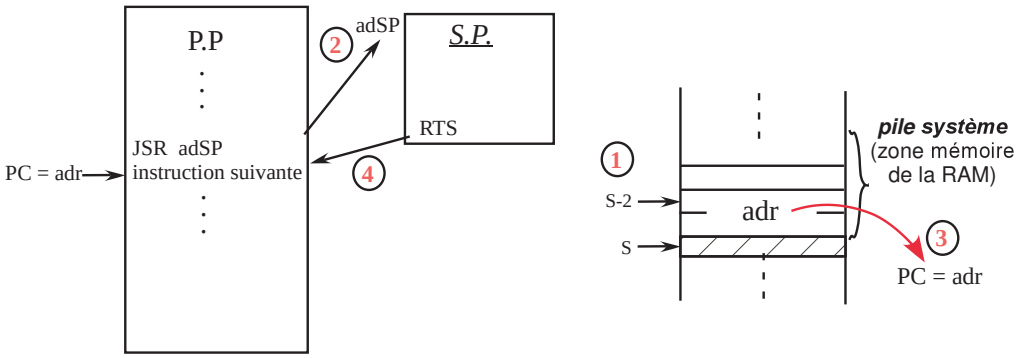


FIGURE V.16 – Mécanisme d'appel et de retour d'un S.P.

V.9.3 Notion d'imbrication

Un programme peut appeler un S.P qui à son tour appel un autre S.P et ainsi de suite. Le nombre d'imbrications maximal dépend en grande partie de la taille de la pile système.

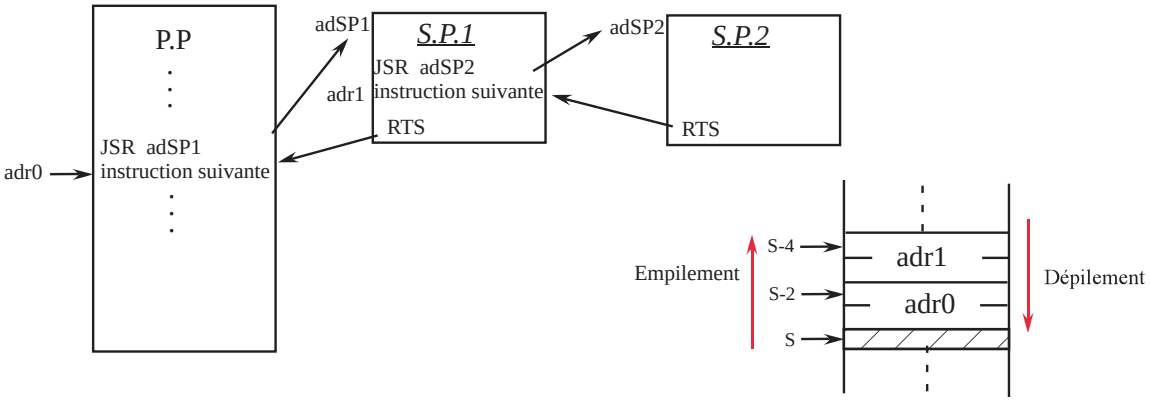


FIGURE V.17 – Mécanisme d'appel et de retour d'un S.P.

V.9.4 Passation de paramètres

De manière générale, un S.P. nécessite des données et fournit des résultats. Données et résultats sont appelés *paramètres* (Entrée et sortie).

1. **Type de paramètres**

Passation par valeur : la valeur du paramètre est directement transmise vers le S.P.

Passation par adresse : c'est l'adresse du paramètre qui est transmise vers le S.P.

2. **Techniques de passation**

Passation dans les registres : Données et résultats sont placés dans les registres internes du μP.

Passation dans le pile : le programme appelant dépose les données dans la pile avant d'effectuer l'appel du S.P. Le S.P. aussi peut déposer les résultats dans la pile qui peuvent être récupérés par le programme appelant.

Exemple :

```
ada    EQU  $0080
adb    EQU  $0081
adc    EQU  $0082
ORG    $FC00
       LDS    #$0400
       LDY    #$1002
       CLRA
       CLRB
       STD    ,Y            * R = 0
       STD    $FE,Y
       LDX    #ada          * X = ada : passation par adresse
       JSR    $FCA0         * saut vers le sous programme qui se trouve à l'adresse $FCA0

       STD    ,Y            * R = (a^2 - a + 1)
       LDX    #adb          * X = adb : passation par adresse
       BSR    calcul        * saut vers S.P. pour calculer (b^2 - b + 1)

       ADDD   ,Y            * R = (a^2 - a + 1) + (b^2 - b + 1)
       BCC    suite1        * addition sur 16 bits alors si retenue ⇒ ajouter 1 au 17^me bit !!
       INC    $FF,Y
suite1 :
       STD    ,Y
       LDX    #adc          * X = adc : passation par adresse
       BSR    calcul        * saut vers S.P. pour calculer (c^2 - c + 1)

       ADDD   ,Y            * R = (a^2 - a + 1) + (b^2 - b + 1) + (c^2 - c + 1)
       BCC    suite2
       INC    $FF,Y
suite2 :
       STD    ,Y

       SWI
```

```
ORG    $FCA0            * le S.P. calcul
calcul:
    LDA     ,X          * lecture A = n : passation par adresse
    TFR   A,B           * A = n et B = n
    DECA                * A = n - 1
    MUL                 * D = n^2 - n
    ADDD  #$0001        * D = n^2 - n + 1

    RTS                 * retour vers le programme appelant

    END
```

V.9.5 Mode d'adressage indexé avec déplacement fixe

Ce mode d'adresse se présente comme suit, par exemple :
LDD **d**,X $\Leftrightarrow$ LDD adr
$\Rightarrow$ où **d** est un déplacement codé sur 8 ou 16 bits signé !
$\Rightarrow$ et l'adresse effective (adr) est la somme signée sur 16 bits de : adr = X + d.

Exemple :
LDA \$02,Y ; avec Y = \$0010
$\Rightarrow$ résultat : A = \$21 et Y = \$0010 (ne change pas)
LDB \$*FF*,Y
$\Rightarrow$ résultat : B = \$33 et Y = \$0010 (ne change pas)

\$000*D*	
\$000*E*	\$1*F*
\$000*F*	\$33
\$0010	\$56
\$0011	\$*A*4
\$0012	\$21
\$0013	\$68
\$0014	

Chapitre VI

Les interfaces d'entrées /sorties

VI.1 Introduction

Les circuits d'interfaces permettent au μP de communiquer avec des dispositifs périphériques. On distingue deux catégories de périphériques :

- les périphériques parallèles ;
- les périphériques séries.

Dans ce chapitre on présentera le circuit d'interface parallèle le PIA 6821.

VI.2 Interface parallèle programmable : PIA6821

L'interface parallèle le PIA 6821 (figure VI.1) (***Peripherical Interface Adapter***) fournit un moyen d'interface des appareils périphériques avec le microprocesseur 6809 (la famille 68XX de MOTOROLA).

VI.2.1 Boîtier et brochage du PIA6821

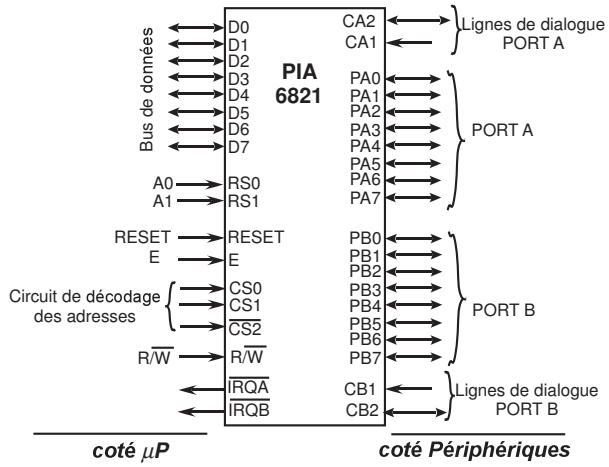
Les broches du PIA6821 sont groupées comme suit (figure VI.1 (b)) :

Bus de données ($D0$ à $D7$) : ce sont 8 lignes bidirectionnelles qui peuvent être directement reliées au bus de données du μP 6809.

Bus d'adresses :

- $RS0$, $RS1$ (Register Select) : permettent de Sélectionner les registres internes du PIA6821 (4 octets mémoires).
- $CS0$, $CS1$, $\overline{CS2}$ (Chip Select) : permettent la Sélection du boîtier PIA ($CS0 \cdot CS1 \cdot \overline{CS2} = 1 \cdot 1 \cdot 0$).

(a) boîtier



(b) brochage

FIGURE VI.1 – Le circuit PIA6821

Bus de contrôle :

- **E :** Signal d'activation des échanges
- **RESET :** Initialisation du PIA. Les registres internes sont mis à zéro.
- $R/\overline{W}$ **:** Lecture - Écriture
- $\overline{IRQA}$**,** $\overline{IRQB}$ **:** Lignes d'interruptions

Lignes de transfert :

- $PA0$ à $PA7$, $PB0$ à $PB7$: Ces 16 broches (2×8) seront utilisées en entrée ou en sortie (recevoir des données ou envoyer des données).
- 04 lignes de dialogue $(CA1, CA2, CB1 et CB2)$;

VI.2.2 Registres internes du PIA6821

Pour l'interfaçage du PIA6821 avec des périphériques, il dispose de :
- 2 ports d'E/S parallèles de 8 bits Port A (PA) et Port B (PB) ;
- 04 lignes de dialogue (CA1, CA2, CB1 et CB2).

Pour son utilisation le PIA dispose de 06 registres internes de 8 bits chacun accessibles au programmeur (figure VI.2)). Ces 06 registres sont divisés en deux groupes, le groupe **A** qui gère le **PORT A**, et le groupe **B** qui gère le **PORTB**.

- **DDRA** (Data Direction Register A) : c'est le registre de direction du port A, en le programmant, on détermine la direction de chacune des lignes PA0 –PA7 (entrée ou sortie).

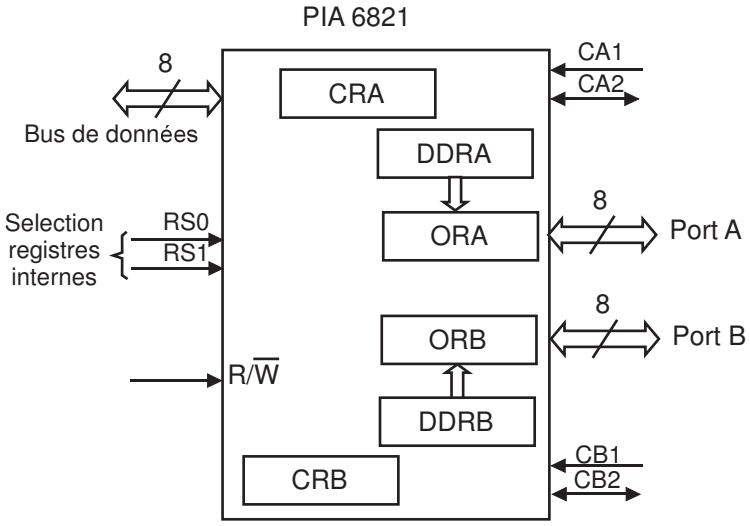


FIGURE VI.2 – composition interne simplifiée du PIA6821

- ► Si par exemple : bit3 de $DDRA = 0 \Rightarrow PA3$ est configurée en entrée ($\leftarrow$)
- ► Et si bit3 de $DDRA = 1 \Rightarrow PA3$ est configurée en sortie ($\rightarrow$).

- **DDRB** : Idem mais pour le PORT B.
- **ORA** (Output Register A) : le registre de sortie de données du port A. C'est ce registre dans lequel vont transiter les données. Pour lire une donnée reçue par le port, il suffira de lire le contenu de ce registre.
- **ORB** : Idem mais pour le PORT B.
- **CRA** : registre de configuration des lignes de dialogues (CA1, CA2) et de la ligne d'interruption IRQA d'une part, et d'autre part permet à distinguer entre le registre DDRA et ORA puisqu'ils ont la même adresse.
- **CRB** : Idem mais pour le PORT B.

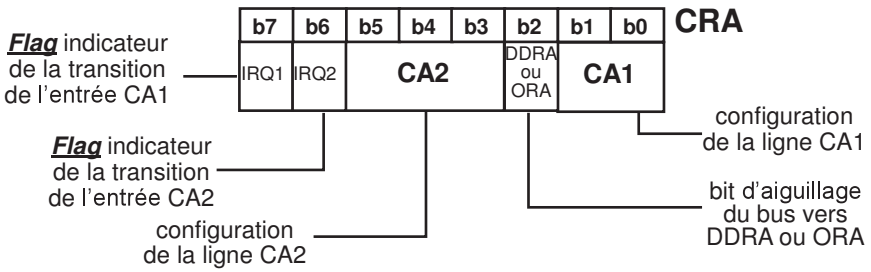


FIGURE VI.3 – les bits du registre CRA (même chose pour CRB)

Sélection des registres internes du PIA6821

TABLE VI.1 – Sélection des registres interne du PIA6821

Bus d'adresse		μP	A15 ... ... A2 Logique de décodage			A1	A0			Adresse
		PIA	$CS0$	$CS1$	$\overline{CS2}$	$RS1$	$RS0$	CRA_2	CRB_2	
Registres	A	DDRA	1	1	0	0	0	0	-	\$8000
		ORA	1	1	0	0	0	1	-	\$8000
		CRA	1	1	0	0	1	-	-	\$8001
	B	DDRB	1	1	0	1	0	-	0	\$8002
		ORB	1	1	0	1	0	-	1	\$8002
		CRB	1	1	0	1	1	-	-	\$8003

VI.2.3 Configuration du PIA

Le PIA6821 doit être initialisé et configuré par programmation afin de pouvoir l'employer comme interface d'entrée/sortie.

Exemple de configuration des ports A et B

- ▷ PA7 – PA0 ⇒ en sortie
- ▷ PB3 – PB0 ⇒ en entrée
- ▷ PB7 – PB4 ⇒ en sortie

Ci-dessous, une suite d'instructions qui permet la configuration des ports A et B du PIA6821 :

```
CLR  $8001        ;  mettre à zéro b2 du CRA pour accéder au DDRA
LDB  #$FF
STB  $8000        ;  DDRA = $FF ⇒ les lignes du PORT A sont tous en sortie!!
CLR  $8003        ;  mettre à zéro b2 du CRB pour accéder au DDRB
LDA  #%11110000
STA  $8002        ;  DDRB = $F0 ⇒ PB0 – PB3 → en entrée , et PB4 – PB7 → en sortie

LDA  #%00000100
STA  $8001        ;  mettre à « 1 » au b2 du CRA pour accéder au ORA
STA  $8003        ;  mettre à « 1 » au b2 du CRB pour accéder au ORB
```

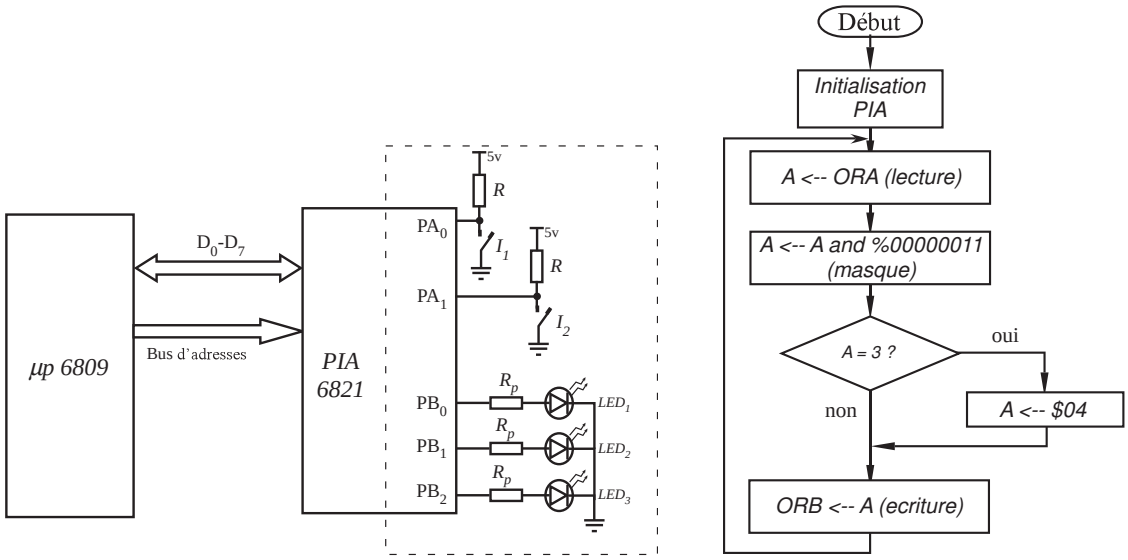
VI.2.4 Premier Exemple

On a le montage de la figure (a) VI.4. Écrire un programme en langage assembleur du 6809 qui réalise les fonctions suivantes :

- initialisation du port A en entrée ;
- initialisation du port B en sortie ;
- selon le code reçu via le **PortA**, il faut allumer la LED correspondante connectées au **PortB** ;

(exemple : code (10)2 est reçu ⇒ la $LED2$ sera allumée)

► **L' organigramme** est donné à la figure (b) VI.4.



(a) Montage de l'exemple 1 (b) L'organigramme

FIGURE VI.4 – montage et organigramme de l'exemple 1

► **Programme de l'exemple 1** :

```
ORG     $FC00
CLR     $8001           ;  mettre à zéro b2 du CRA pour accéder au DDRA
LDB     #$00
STB     $8000           ;  DDRA = $00 ⇒ PORT A est en entrée!!
CLR     $8003           ;  mettre à zéro b2 du CRB pour accéder au DDRB
LDA     #$FF
STA     $8002           ;  DDRB = $FF ⇒ PORT B est en sortie

LDA     #%00000100
STA     $8001           ;  mettre à « 1 » au bit b2 du CRA pour accéder au ORA
STA     $8003           ;  mettre à « 1 » au bit b2 du CRB pour accéder au ORB

loop:
LDA     $8000           *  lire les entrées
ANDA    #%00000011      *  masquer les bits non nécessaires!
CMPA    #%00000011
BNE     suite
LDA     #%00000100      * pour allumer la LED 3 !
suite:
STA     $8003
BRA     loop

END
```

VI.2.5 Deuxième Exemple

On a le montage de la figure VI.5. on désire réaliser le fonctionnement suivant :
Le thermomètre mesure de façon continue la température et fournit sa valeur en code binaire sur 8 bits ($0°C$ – $60°C$).
Pour afficher un chiffre de 0 à 9, il suffit d'envoyer son code BCD sur 4 bits vers le décodeur 7447 et de maintenir ce code aussi longtemps qu'il le faut.

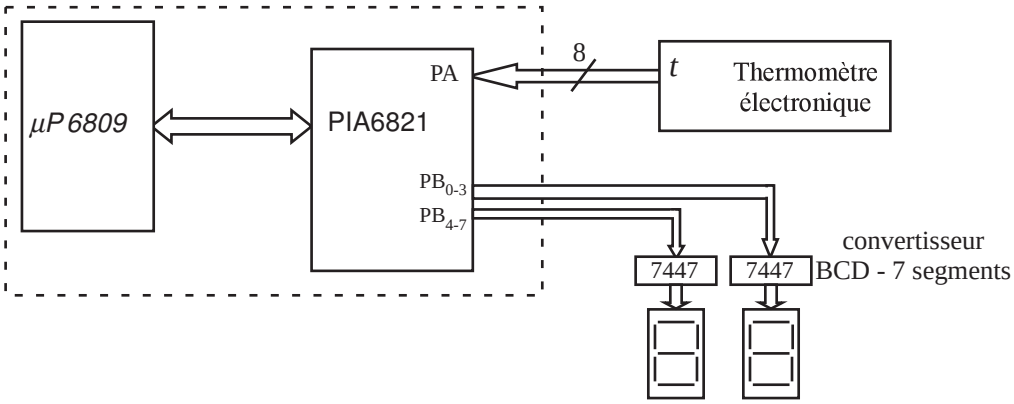


FIGURE VI.5 – Montage de l'exemple 2

► **L' organigramme** est donné à la figure VI.6.

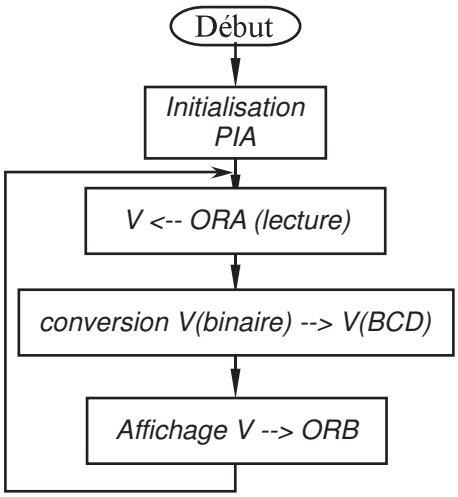


FIGURE VI.6 – Organigramme de l'exemple 2

► **Programme de l'exemple 2** :

```
CRA     EQU   $8001
DDRA    EQU   $8000
```

```
ORA     EQU   $8000
CRB     EQU   $8003
DDRB    EQU   $8002
ORB     EQU   $8002

ORG   $FC00
CLR   CRA          ;  mettre à zéro b2 du CRA pour accéder au DDRA
CLR   DDRA         ;  DDRA = $00 ⇒ les lignes du PORT A sont tous en entrée!!
CLR   CRB
LDB   #$FF
STB   DDRB       *  portB en sortie
LDB   #$04
STB   CRA          ;
STB   CRB

loop:
LDB   ORA        ;  lecture de la donnée
CLRA             ;
suite:
CMPB  #$0A       ;  conversion binaire -> BCD
BLO   fin
SUBB  #$0A
ADDA  #$80
BRA   suite
fin:
STB   ORB        ;  affichage chiffre des unités
ORA   ORB
STA   ORB        ;  affichage chiffre des dizaines

BRA   loop

END
```

VI.2.6 Troisième Exemple

On a le montage suivant :

Il s'agit de transmettre vers le périphérique un bloc de 100 octets rangés à l'adresse *adbloc*, en respectant le chronogramme de poignée de main imposé par le périphérique.

Avant de donner la solution, on va introduire les lignes de dialogue pour le PortA qui sont configurable via le registre CRA (figure VI.8).

Lignes de dialogue du PortA

La configuration de la ligne $CA1$ **:** un des cas suivants :

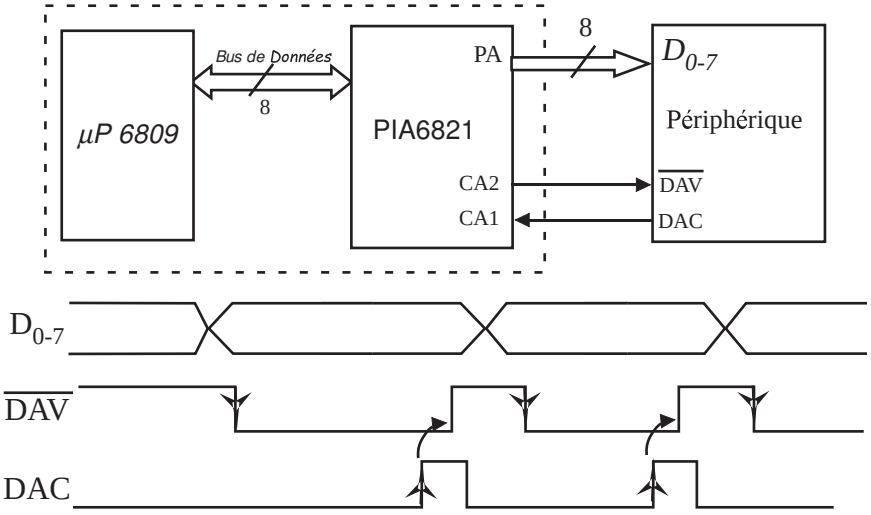


FIGURE VI.7 – Montage de l'exemple 3 et le chronogramme poignée de main du périphérique

- Si $b1 = 0 \Rightarrow CA1$ est sensible sur front descendant ;
- Si $b1 = 1 \Rightarrow CA1$ est sensible sur front montant ;
  Dans les deux cas, pour chaque front sensible sur $CA1 \rightarrow b7$(IRQ1) =1 !!
  Le $b7$ revient à « 0 » suite à une lecture du registre **ORA** !!
- Si $b0 = 0$ l'interruption (**IRQA**) est masqué (désactivée) ;
- Si $b0 = 1$ l'interruption (**IRQA**) est envoyée au μP ; si elle est connectée à l'entrée d'une interruption de ce dernier (μP).

La configuration de la ligne CA2 : un des cas suivants :

- Si $b5 = 0 \Rightarrow CA2$ est utilisé comme entrée fonctionnant de manière identique à $CA1$ avec $b4$ identique à $b1$ et $b3$ identique à $b0 \rightarrow$ c'est le bit $b6$ (IRQ2) qui a le même fonctionnement que $b7$ dans ce cas là !!
- Si $b5 = 1$, la ligne $CA2$ est utilisée comme une sortie !!!
  $\rightarrow$ $b4 = b3 = 0$: $CA2 = 0$ après une lecture de ORA, $CA2 = 1$ lors d'une transition active de CA1,
  $\rightarrow$ $b4 = 0$, $b3 = 1$: $CA2 = 0$ après une lecture de ORA, $CA2 = 1$ lors du premier front descendant du signal d'horloge après remise à zéro de $CA2$ (*mode impulsionnel*)
  $\rightarrow$ $b4 = 1$: $CA2$ recopie $b3$ (*mode manuel*)

► **L'Organigramme de l'exemple 3** est donné à la figure VI.9.

Programmation des lignes de dialogue :

- $CA1$: entrée sensible à un front montant (car **DAC**). Si un front montant arrive $\Rightarrow$ 1 mis dans $IRQ1$ (RAZ logiciel). $b1$ $b0$ = '10'

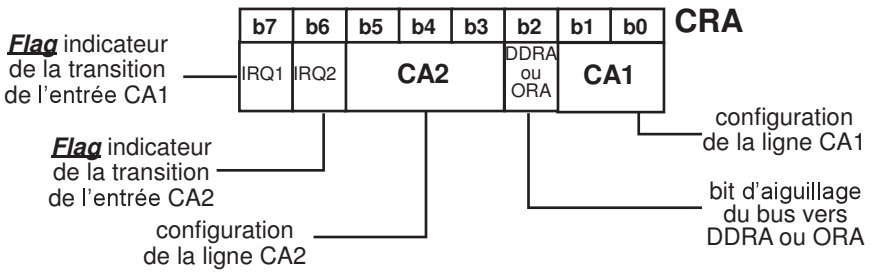


FIGURE VI.8 – les bits du registre CRA (même chose pour CRB)

- $CA2$: Sortie manuelle, initialisée à l'état désactivée (1). $b5$ $b4$ $b3$= '111'

► **Le programme de l'exemple 3** :

```
CRA     EQU  $8001
DDRA    EQU  $8000
ORA     EQU  $8000
adbloc  EQU  $0010

ORG   $FC00
CLR   CRA          ;  mettre à zéro b2 du CRA pour accéder au DDRA
LDA   #$FF
STA   DDRA         ;  DDRA = $FF ⇒ les lignes du PORT A sont tous en sortie!!

LDA   #%00111110
STA   CRA          ;

LDX   #adbloc
LDB   #$64

loop:
LDA   ,X+
STA   ORA
LDA   #%00110110
STA   CRA

att:
LDA   CRA
BITA  #%10000000
BEQ   att
LDA   #%00111110
STA   CRA
DECB
BNE   loop

END
```

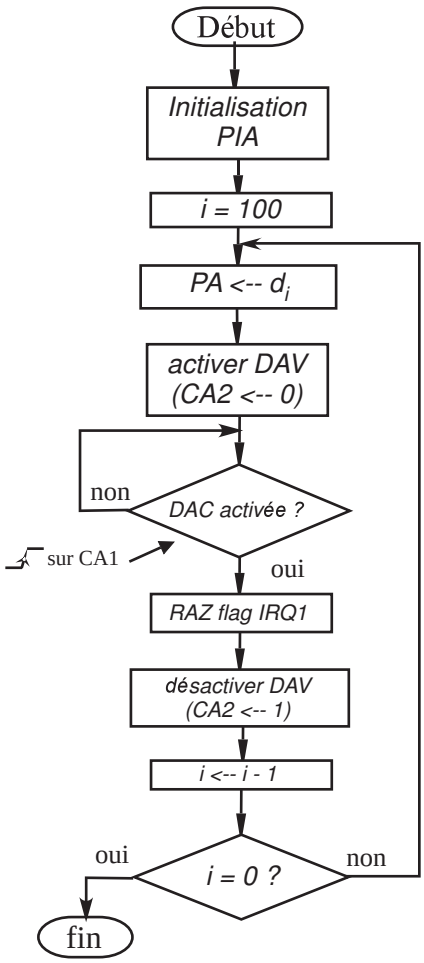


FIGURE VI.9 – Organigramme pour l'exemple 3

Chapitre VII

Les Interruptions

VII.1 Introduction

Il y a 3 modes de mise en oeuvre d'une procédure de gestion des entrées/sorties :

- Mode programmée (par sondage ou par scrutation) ;
- Les interruptions
- Accès direct mémoire (DMA)

VII.1.1 Mode programmé

Dans ce mode là, le μP scrute logiciellement ses interfaces, pour savoir à quel moment il doit effectuer une entrée ou une sortie.

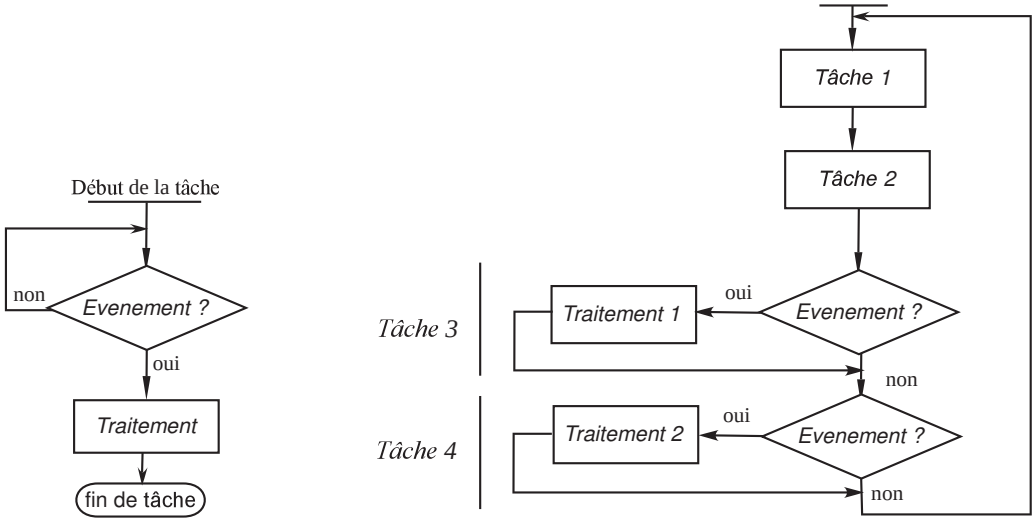
- Quand le μP est totalement libre pour la tâche d'E/S considérée, on peut lui faire alors exécuter des boucles d'attente aussi longue que nécessaire (figure (a) VII.1).
- Quand le μP n'est pas totalement libre, mais peut venir scruter les indicateurs des interfaces régulièrement (figure (b) VII.1).

VII.1.2 Mode Interruption

Dans ce mode, il n'y a pas de scrutation logicielle de la part du μP, quand l'interface reconnaît le moment d'effectuer un traitement d'E/S, il avertit, par voie matérielle le μP (**IRQ** : *Interrupt ReQuest*).

Exemple :

Donnée envoyée par le μP $\Rightarrow$ le PIA transmit cette donnée via le port A et active la ligne $\overline{DAV}$ $\Rightarrow$ le périphérique reçoit la donnée et active la ligne DAC pour demander la suite des



(a) Mono tâche (b) Multi tâches

FIGURE VII.1 – Mode programmé par scrutation

données ⇒ le PIA enregistre cette événement et envoie une demande d'interruption au μP via la ligne IRQ ⇒ le μP suspend son travail en cours, va lire le caractère reçu et effectue éventuellement son traitement ⇒ ensuite, le μP continue son travail qu'il a suspendu.

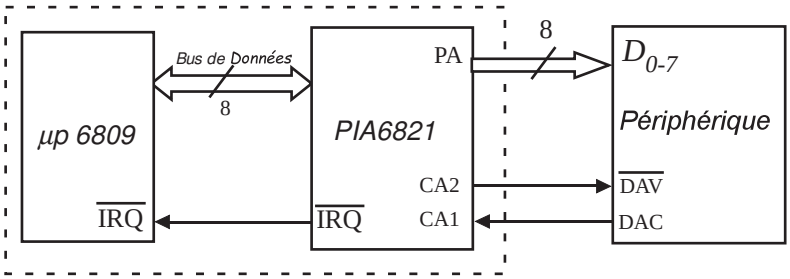


FIGURE VII.2 – exemple : mode interruption

VII.1.3 Mode DMA

L'accès direct à la mémoire (DMA : Direct Memory Access) est une technique de transfert de données direct entre un périphérique et la mémoire centrale ou vice versa, effectué sans intervention du μP. La mise en oeuvre de cette technique nécessite l'ajout d'un circuit spécifique qui gère cet accès à la mémoire afin d'éviter les conflit d'accès à la mémoire entre le périphérique et le μP. Cette technique permet de décharger le μP et augmenter la vitesse de transfert des données entre les périphériques et le système.

Exemples :

- transfert de fichier entre disque dure et la mémoire centrale d'un PC ;

► transfert de données d'affichage vers le processeur graphique ...

VII.2 Les Interruptions matérielles du μP 6809

VII.2.1 Définition

Les interruptions sont des entrées que le μP examine matériellement, à chaque cycle d'horloge. Ces entrées permettent au μP de réagir à des évènements quand ceux-ci exigent une réponse rapide, la réponse consiste en l'exécution d'un sous programme (S.P.) propre à l'évènement appelé Sous-Programme d'Interruption : SPI.

Le microprocesseur 6809 possède quatre lignes d'interruption matérielles qui sont par ordre de priorité :

- $\overline{RESET}$: Ré-initialisation du microprocesseur.
- $\overline{NMI}$: *Non Masquable Interrupt.*
- $\overline{FIRQ}$: *Fast Interrupt Request.*
- $\overline{IRQ}$: *Interrupt Request.*

Remarque : ces entrées sont actives par un ***niveau "bas"***.

VII.2.2 Mécanisme d'une interruption matérièle

Initialement le μP est à l'état normal (exécution normale du programme principal). Dans ce cas là, tous les lignes d'interruption sont à l'état haut. Quand une interruption surgit, c-à-d une ligne d'interruption passe à l'état bas, le μP enregistre une demande d'interruption.

phase a : le μP termine l'exécution de l'instruction en cours puis :
- Si l'interruption enregistrée est non autorisée alors elle est ignorée par le μP. Ce dernier continue l'exécution normale ;
- Si l'interruption enregistrée est autorisée alors le μP prend en compte la demande d'interruption et suit une procédure d'exception selon le chronogramme de la figure VII.3.

phase 1 : le μP effectue une sauvegarde du contexte dans la pile du système pointée par le registre d'indexe S, il y 'a deux contextes de sauvegarde :
- `sauvegarde partielle` : les contenus du compteur de programme PC et du registre d'état CC sont sauvegardés dans la pile système et l'indicateur E est mis à 0 (E = 0) ;
- `sauvegarde complète` : les contenus des registres PC, CC, U, Y, X, DP, B et A sont sauvegardés dans la pile système et l'indicateur E est mis à 1 (E = 1) ;

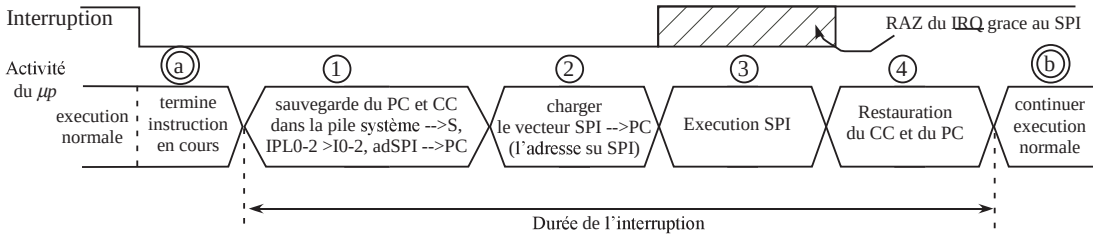


FIGURE VII.3 – chronogramme d'une interruption autorisée

phase 2 : le μP charge le registre PC par l'adresse du SPI qui doit se trouver en mémoire à un emplacement qui lui est réservé : `autovecteur` = `adresse du SPI` ;

phase 3 : exécution du SPI dont l'adresse vient d'être chargée dans le PC. Le SPI doit être terminé impérativement par l'instruction **RTI** (*Return from Interruption*) ! !

phase 4 : exécution de l'instruction RTI : restauration à partir de la pile système des registres sauvegardés en particulier les registres CC et PC.

Phase b : le μP reprend ainsi l'exécution du programme principal, avec comme seule perturbation un temps de retard.

Remarques importantes

- L'exécution du SPI doit être assez courte, il ne doit certainement pas comporter des boucles infinies, ou bien des boucles d'attente à des évènements externes.
- Le SPI doit veiller à désactiver la ligne d'interruption qui a provoqué son exécution sous peine de provoquer une série d'interruptions non-désirées.
- *La sauvegarde des registres SR et PC est systématique, celle des autres registres ne l'est pas dans le cas de sauvegarde partielle.* ***Il faut donc veiller à sauvegarder au début du SPI puis restaurer à sa fin tous les registres susceptibles d'être changés (utilisés) par le SPI.***

VII.2.3 Contrôle de la prise en compte d'interruption

Les bits I, F et E, du registre d'état (figure VII.4), sont des flags (drapeaux) pour les interruptions.

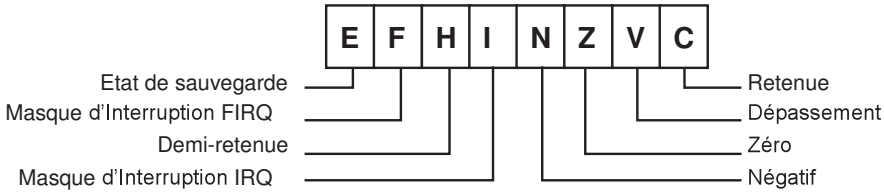


FIGURE VII.4 – les différents bits du registre d'état CCR

Le flag I : c'est un masque concernant l'interruption IRQ :
- I = 1 → l'interruption IRQ est ignorée !
- I = 0 → l'interruption IRQ est autorisée (prise en compte par le μP) ! → Avec ce type d'interruption, le contexte total du μP (12 octets) est sauvegardé sur la pile Système. L'autovecteur **IRQ** se trouve au adresses $[\$FFF8 : \$FFF9] \mapsto PC$!

Le flag F : c'est un masque concernant l'interruption FIRQ :
- F = 1 → l'interruption FIRQ est ignorée !
- F= 0 → l'interruption FIRQ est autorisée (prise en compte par le μP) ! → Ici, le contexte partiel du microprocesseur (3 octets) est sauvegardé sur la pile Système (seuls les contenus des registres PC et CCR sont concernés). L'autovecteur **FIRQ** se trouve au adresses $[\$FFF6 : \$FFF7] \mapsto PC$!

Le flag E : Ce flag est positionné par le μP afin d'indiquer le type de sauvegarde devant être réalisée (partielle ou complète). Il fait office en quelque sorte de mémoire interne.
- E = 1 → Etat de la sauvegarde est ***complète*** !
- E = 0 → Etat de la sauvegarde ***partielle*** !

Remarque : la ligne d'interruption **NMI** permet de générer une interruption nom masquable de haute priorité ! Dans ce cas là, la sauvegarde est complète. L'autovecteur **NMI** se trouve au adresses $[\$FFFC : \$FFFC] \mapsto PC$!

Pour la configuration des bits I et F, on dispose de deux instructions : **ANDCC** et **ORCC** comme le montre le tableau VII.1.

TABLE VII.1 – Tableau des instructions pour autoriser ou inhiber les interruption IRQ et FIRQ

L'instruction	État de l'indicateur	Interruption
ANDCC ♯$\$EF$	I = 0	IRQ autorisée
ORCC ♯\$10	I = 1	IRQ inhibée
ANDCC ♯$\$BF$	F = 0	FIRQ autorisée
ORCC ♯\$40	F = 1	FIRQ inhibée

Exemple de programme pour la gestion d'interruption IRQ : On suppose qu'on a le montage de la figure suivante :

Hypothèse générale : Sans précision préalable, on suppose que :
- un programme principal se trouve déjà en mémoire et qu'il comprend un certains nombres de tâches ;

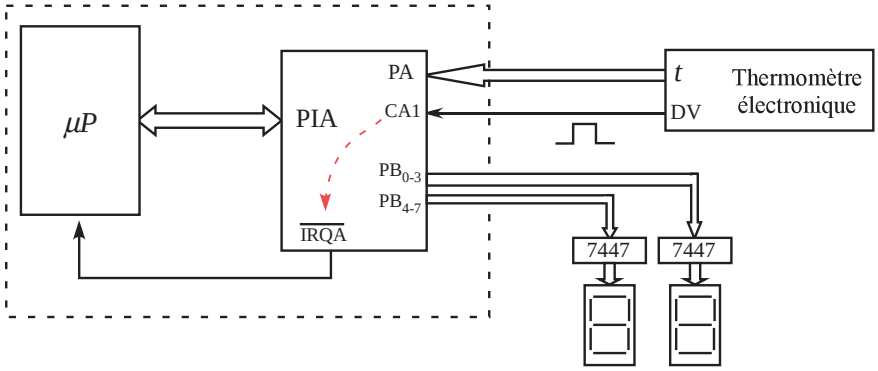


FIGURE VII.5 – Montage de l'exemple d'interruption

- l'autovecteur IRQ est initialisée par l'adresse `adrSPI` ;

Séquence d'initialisation principale : elle doit être effectuée dans le programme principal :

1. Configuration du PIA6821 :
   - PortA en entrée
   - CA1 entrée sensibles à un front positif, autoriser l'interruption de CA1
   - PortB en sortie
2. Initialisation du masque d'interruption I = 0.

Séquence d'initialisation

```
CRA     EQU   $8001
DDRA    EQU   $8000
ORA     EQU   $8000
CRB     EQU   $8003
DDRB    EQU   $8002
ORB     EQU   $8002

ORG     $FC00
        ⋮
CLR     CRA          ;  mettre à zéro b2 du CRA pour accéder au DDRA
CLR     DDRA         ;  DDRA = $00 ⇒ les lignes du PORT A sont tous en entrée!!
LDA     #%00000111   ;  CA1 entrée sensibles à un front positif,
STA     CRA          ;  et autoriser l'interruption de CA1, choix ORA

CLR     CRB
LDB     #$FF
STB     DDRB      *  portB en sortie
LDB     #$04
STB     CRB
        ⋮
```

Sous Programme d'Interruption

```
ORG    adrSPI

LDB    ORA        ;  lecture de la donnée et mise à zéro du flag d'interruption IRQ1 du PIA
* traitement      ;  dans ce cas là tous les registres sans sauvegardés (IRQ -> sauvegarde compiète)
CLRA              ;  alors on peut utiliser les registres pour le traitement
loop:
CMPB   #$0A       ;  conversion binaire -> BCD
BLO    fin
SUBB   #$0A
ADDA   #$80
BRA    loop
fin:
STB    ORB        ;  affichage chiffre des unités
ORA    ORB
STA    ORB        ;  affichage chiffre des dizaines

* instruction de retour vers le programme principal !
RTI
```

Annexe A : Les Registres

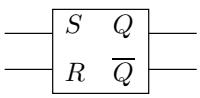
Rappel sur les bascules

Définition

Une bascule est un circuit bistable pouvant prendre deux états logiques : 0 ou 1. La bascule a la capacité de conserver son état (état des sorties) $\Rightarrow$ elle est utilisée comme **mémoire**.

Bascule RS

C'est une bascule **Asynchrone** : l'état de la bascule peut changer à n'importe quel instant.



S pour SET (mise 1)
R pour reset (mise 0)
Q sortie, $\overline{Q}$ sortie complementaire

FIGURE VII.6 – Schema d'une bascule RS

R	S	$Q(t)$	$\overline{Q(t)}$	**Remarque**
0	0	$Q(t-1)$	$\overline{Q(t-1)}$	Etat précédent
0	1	1	0	Mise à 1
1	0	0	1	Mise à 0
1	1	-	-	Etat indéterminé

TABLE VII.2 – La table de vérité de la bascule RS

Bascule RSH

Cette bascule présente une entrée H de plus que la bascule RS, qui est une entrée d'horloge, suivant le schéma :

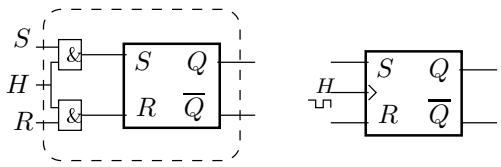


FIGURE VII.7 – Schema d'une bascule RSH

Bascule D (Delay) ou latch ou mémoire

Il s'agit en fait d'une bascule RSH où l'entrée R est le complément de S : C'est le

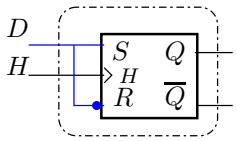


FIGURE VII.8 – Schema d'une bascule RSH

composant de base d'une mémoire d'ordinateur : est mise à 1 ou à 0 au moment voulu et figé le reste du temps.

Bascule JK (bascule synchrone)

C'est une bascule issue d'une association de deux bascules RSH en montage maître-esclave.

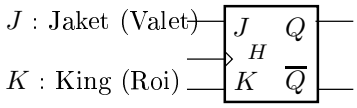


FIGURE VII.9 – Schema d'une bascule RSH

La bascule JK divise les impulsions d'horloge par 2 lorsque $J = K = 1$.

J	K	Q_t	**Remargues**
0	0	Q_{t-1}	fonction mémoire
0	1	0	Mise à 0 de la sortie Q
1	0	1	Mise à 1 de la sortie Q
1	1	$\overline{Q_{t-1}}$	Complement de la sortie

TABLE VII.3 – La table de vérité de la bascule JK

Les Registres

Un registre est un ensemble de bascules mémoires. Le contenu d'un registre peut donc être considéré comme une information binaire (un "**mot**" de n bits).

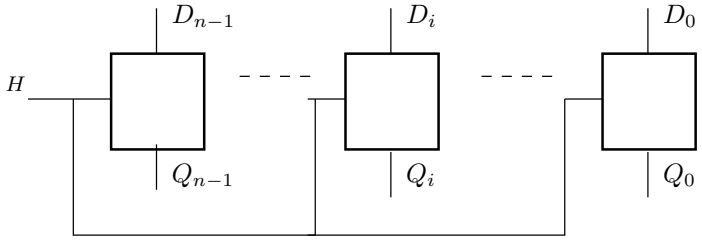


FIGURE VII.10 – Schema d'un registre

Les applications des registres sont multiples :
- ► Registre mémoire ;
- ► Registre à décalage (conversion parallèle $\leftrightarrow$ série) ;
- ► Compteurs ;
- ► Multiplication ou division par une puissance de 2.

Registre mémoire

La fonction d'un tel registre est de "**stocker / mémoriser**" une information de n bits.

Exemple : 1 1 0 0, chacun des bits sera stocké par une bascule.

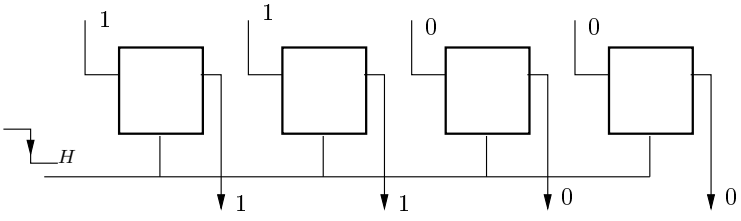


FIGURE VII.11 – Schema d'un registre mémoire 4 bits

Registre à décalage

Les registres à décalage sont caractérisés par les méthodes d'introductions et de restitution de l'information. Les principaux types sont :

- Registre à mode d'écriture (ou chargement) parallèle et mode de lecture (ou sortie) série ;
- Registre à mode d'écriture (ou chargement) série et mode de lecture (ou sortie) parallèle ;
- Registre à mode d'écriture (ou chargement) série et mode de lecture (ou sortie) série.

Registre à mode d'écriture parallèle et mode de lecture série

En charge le mot à stocker $(E_3\,E_2\,E_1\,E_0)$ avec $(\overline{e}/L = 0)$;

En suite , il faut quatre impulsions sur H pour effectuer la lecture via S avec $(\overline{e}/L = 1)$.

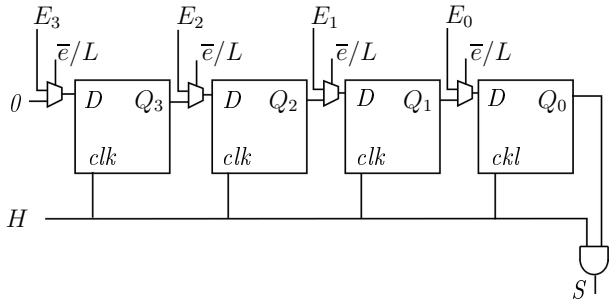


FIGURE VII.12 – Registre EP-LS

Rem : la lecture est destructive du mot chargé !

Registre à mode d'écriture série et mode de lecture parallèle

Lecture

- $L = 0 \rightarrow S_i = 0 \Rightarrow$ l'information n'est pas valide
- $L = 1 \Rightarrow$ la lecture est valide.

Écriture

Les quatre bits du mot à mémoriser sont présentés les uns après les autres à l'entrée E et se propagent dans le registre par décalages successifs au rythme de l'horloge H.

Registre à mode d'écriture série et mode de lecture série

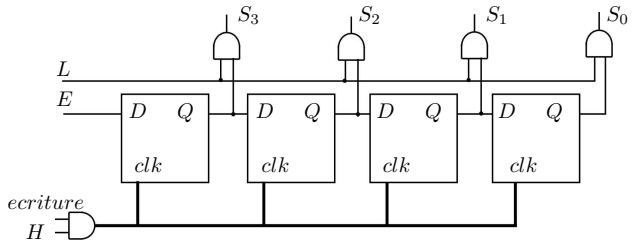


FIGURE VII.13 – Registre ES-LP

L'écriture se fera par des décalages à droit ;

Exemple :

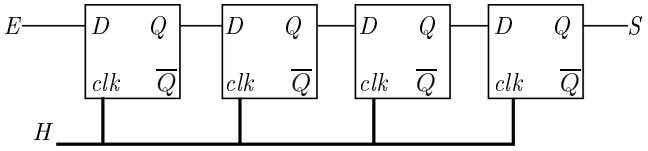


FIGURE VII.14 – Registre EP-LS

→ 1er impulsion 1 x x x
→ 2ème impulsion 1 1 x x
→ 3ème impulsion 0 1 1 x
→ 4ème impulsion 1 0 1 1

Rem : la lecture est destructive du mot chargé !

Un registre universel

La figure montre un registre avec écriture série ou parallèle et lecture série ou parallèle :

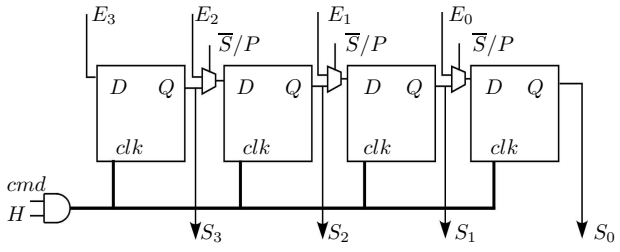


FIGURE VII.15 – Schema d'un registre universel

- **cmd** : commande pour activer les opération du registre ;
- $\overline{S}/P$commande du chargement série/parallele.

Annexe B : Les fiches des Travaux Dirigés SAM

Fiche TD 1

Exercice 1

Exprimer en binaire les nombres suivants : $(965)_{10}$, $(607)_8$, $(A8B)_{16}$.

Exprimer en hexadécimal les nombres suivants : $(3589)_{10}$, $(10110110011101)_2$, $(7106)_8$.

Exprimer en octal les nombres suivants : $(1157)_{10}$, $(10111010)_2$, $(F1F)_{16}$.

Exprimer en code BCD les nombres suivants : $(39)_{10}$, $(184)_{10}$, $(5036)_{10}$.

Exprimer en code ASCII les caractères suivants : 'H' , 'h' , '8' , '#' , espace.

Exercice 2

Donner le nombre de bits nécessaires pour représenter les nombres décimaux de 0 à 999 selon le code binaire pur, puis selon le code BCD

Exercice 3

Effectuer les additions binaires suivantes :

1010 +0101	1101 +0101	01011011 +00001111	00111111 +00011111

Effectuer les soustractions suivantes :

1110 -1000	1010 -0101	01100110 -00011010	01111000 -00111111

Effectuer les multiplications suivantes :

1001 x 11	1111 x 101	1101 x 1001	01011011 x 1111

Exercice 4

Donner le codage en compléments à 2 sur 8 bits des nombres décimaux signés suivants :

(+65) , (+121), , (-11), (-42) , (-23)

Donner les équivalents décimaux des nombres suivants codés en compléments à 2:

(11111011), (01110111), (10001111) , (00001111)

Exercice 5

En binaire, additionner les nombres suivants (utiliser le complément à deux pour le codage des nombres négatifs): [27 + 54] [25 + (-127)] [(-23) + 27] [(-122) + (-145)]

Exercice 6

Que peuvent représenter les codes binaires suivants stockés en mémoire d'un système à microprocesseur :

$(00110000)_2$, $(01000001)_2$, $(11100010)_2$, $(11011011)_2$

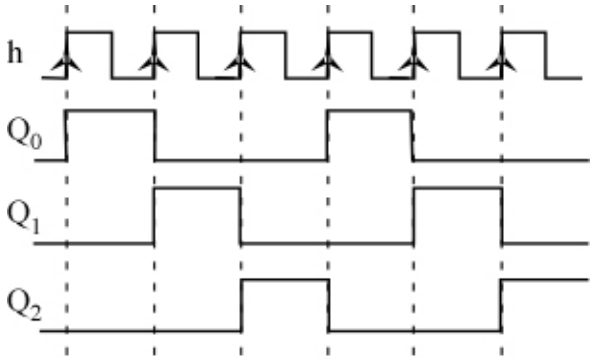
Fiche TD 2

Exercice 1

1. Effectue la synthèse d'un compteur synchrone modulo-5.
2. Réalisé le compteur à l'aide d'un compteur modulo-8.

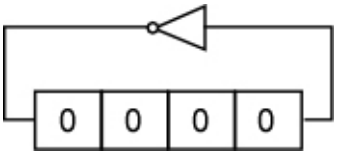
Exercice 2

Réaliser un compteur cyclique (*ring counter*) qui génère les états suivants : (100) → (010) → (001)



Exercice 3

Un compteur Johnson est constituer par un registre à décalage est un inverseur reliant les deux bascules extrêmes du registre. On suppose que le registre est à 4 bits et initialisé à la valeur "0000".



Donner la séquence générée par ce compteur.

Fiche TD 3

Exercice 1

1. Combien de mots de données contient une **ROM** de **8Koctets** ?
2. Quelle est la capacité d'une mémoire **RAM** ayant un boitier avec un bus d'adresse de **11** lignes ?
3. Un microprocesseur peut adresser un espace mémoire de **64Ko**, combien de lignes comporte t-il son bus d'adresse ?
4. Quelle est la taille de l'espace mémoire adressable par un microprocesseur ayant un bus d'adresse de **24** lignes ?

Exercice 2

Dans la plupart des micro-ordinateurs, les adresses des cases mémoire sont exprimées en hexadécimal.

1. L'adresse basse d'une zone mémoire est $500. Son adresse haute est $F9F. Combien de cases mémoire contient – elle ?
2. Une zone mémoire possède 16K octets. Si l'adresse basse de cette zone est $00, quelle est l'adresse haute ?

Exercice 3

On utilise une mémoire ROM de 16 Ko et une mémoire RAM de 8 Ko. Sachant que le microprocesseur a un bus d'adresse de 16 bits :

1. Indiquer les adresses de début et de fin de chaque mémoire. Faire un plan d'adressage mémoire.
2. Faire la logique de décodage d'adresse avec des circuits logiques.

Exercice 4

On considère un système à base d'un microprocesseur (***figure 1***) comprenant les éléments suivants :

- Un microprocesseur 8 bits ;
- Une mémoire ***ROM*** ;
- Une mémoire ***RAM*** ;
- Un circuit d'interface entrée/sortie (***PIO***) ayant 04 registres internes.

1. Calculer la capacité de la mémoire RAM et celle de la mémoire ROM ;
2. Que signifie le terme $\overline{CS}$? Sur quel état logique cette entrée est-elle valide ?
3. Afin de réaliser la fonction décodage d'adresses, quel circuit peut-on utiliser ?
4. Déterminer les plages d'adresses permettant la sélection des différents circuits.

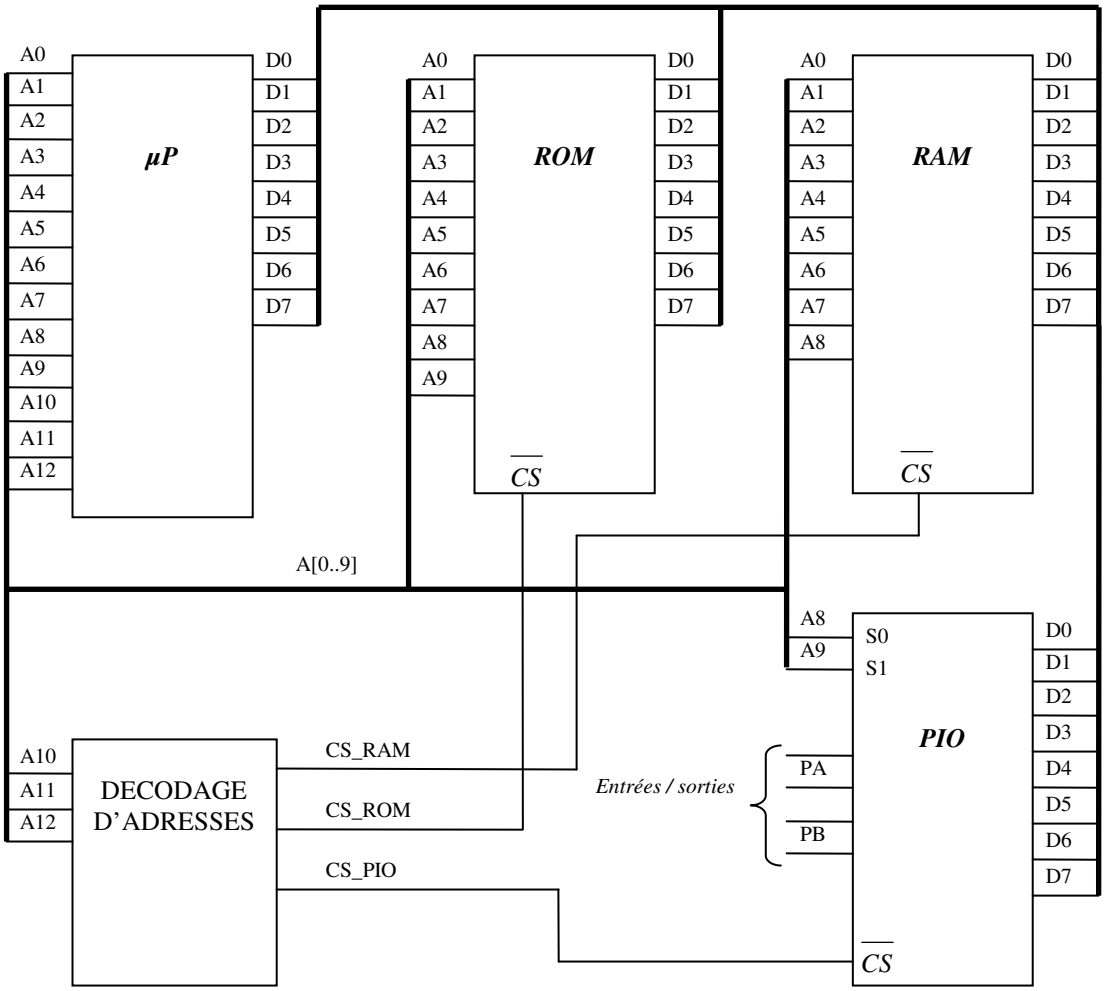


Figure 1 : Le schéma simplifié du système

Fiche TD 4

Exercice 1

Commentez le déroulement des instructions suivantes en complétant les champs libres correspondant au contenu des registres, au contenu des cases mémoires et à l'état des indicateurs N, Z, V, C du registre d'état.

			Registre	**Espace mémoire**	**Indicateurs**
	ORG $FC00				**N Z V C**
START	**NOP**				
	LDA	#$9E	A=		
	ADDA	#$0F	A=		
	STA	$0200		[$0200]=	
	LDX	#$0400	X=		
	TFR	X,Y	Y=		
	LDD	#$4321	A=		
			B=		
	STB	$0200		[$0200]=	
	ADDA	$0200	A=	[$0200]=	
	LDD	#$5AB2			
	NEGB		B=		
	LDB	#$E4	B=		
	STB	$0100		[$0100]=	
	NEG	$0100		[$0100]=	
	END				

Exercice 2

Donner, grâce au tableau des instructions, les codes machines en hexadécimal des instructions suivantes :

Adresses	**Codes machines**	**Langage assembleur**	**commentaires**
$FC00	$86 $9E	**LDA** #$9E	
$FC02		**ADDA** #$0F	
		STA $0200	
		LDS #$3FFF	
		LDB #%00000001	
		TFR B, DP	
		INCA	
		STA $20	
		LDD #$E402	
	…… $84	**STB** ,X	
		END	

- Quels sont les modes d'adressage utilisés dans les instructions suivantes :
  a. **TFR** B,DP
  b. **STA** $20
  c. **STB** ,X

Exercice 3

Ecrire un programme en assembleur 6809 qui permet d'effectuer l'échange des contenus de 4 octets situés à l'adresse **$0AC0** avec les contenues de 4 octets situés à l'adresse **$03F0**.

Exercice 4

On veut calculer l'équation suivante : $r = \frac{x+y}{2}$, avec ***x*** et ***y*** sont des nombres **signés** sur 8 bits. Ecrire le programme en assembleur qui effectue ce calcul sachant que les adresses de ***x*** et ***y*** sont respectivement ***adrx*** et ***adry***.

Rem : on suppose que (***x*** + ***y***) ne dépasse jamais 8bits !!

Exercice 5

Un nombre ***a*** non signé sur ***16*** bits se trouve à l'adresse **$0300**. Un autre nombre ***b*** non signé sur ***8*** bits se trouve à l'adresse **$0310**. On veut réaliser la multiplication entre ces deux nombres. Le résultat sera stocké à l'adresse **$0400**.

Exercice 6

Après assemblage et chargement du programme, quelle est le résultat d'exécution des directives suivantes :

```
; $00A0  DB     $3C
; $00A1  DB     $C1
; $00A2  DB     $02

val1  EQU  $56
val2  EQU  $E402

ORG   $FC00

      LDA   #val1
      ANDA  #%00001111
      STA $0200
      LDB   #$01
      TFR   B, DP
      JMP   $FC80

ORG   $FC80

Suite:  INCA
      STA  $20
      LDD  #val2
      STB  ,X
      BRA  Suite
END
```

Fiche TD 5

Exercice 1

Ecrire un programme en assembleur 6809 permettant de mettre la valeur **$00** dans 100 cases mémoires à partir de l'adresse **$0200**.

Exercice 2

Une chaîne de caractères se trouve en ***RAM*** à partir de l'adresse ***adtab***. Le dernier octet de cette chaîne contient le code ASCII du caractère de contrôle "***CR***" (retour chariot).

Ecrire un programme en assembleur 6809 qui détermine le nombre de caractères de l'alphabet (A-Z , a-z) que contient cette chaîne de caractère.

Exercice 3

On considère une table de 30 données signées sur 8 bits, est rangée en mémoire à l'adresse ***adtab*** = $0300. Développer un programme qui détermine le nombre d'éléments positifs, négatifs et nuls et qui ranger ces trois résultats respectivement dans les adresses **$0020**, **$0021**, **$0022**.

Vérifier le programme en initialisant au préalable la table de données.

Exercice 4

Soit une table de **10** nombres non signés codés sur 8 bits, débutant à partir de l'adresse **$0100**.

Donner l'organigramme puis le programme effectuant le calcul de la moyenne de ces **10** nombres et stockant le résultat à l'adresse **$0020**.

Exercice 5

Ecrire un programme qui effectue le calcul suivant : $p = \frac{(a^2-1)}{2} + \frac{(b^2-1)}{2} + \frac{(c^2-1)}{2}$, avec ***a, b*** et ***c*** des ***nombres signés*** sur 8 bits et stockés en mémoire aux adresses ***ada***, ***adb*** et ***adc***. Utiliser un ***sous programme*** pour réduire la taille du programme global.

Exercice 6

On suppose le montage de la figure 1.

- ✓ Les deux interrupteurs I_1 et I_2 (connectés au port A : **PA0** et **PA1**) commandent les deux LED 1 et 2 (connectées au port A : **PA4** et **PA5**).
- ✓ Les deux interrupteurs I_3 et I_4 (connectés au port B : **PB4** et **PB5**) commandent les deux LED 3 et 4 (connectées au port B : **PB0** et **PB1**).

- Donner l'organigramme et le programme en langage assembleur qui effectue l'initialisation du circuit d'interface parallèle et la commande des LED.

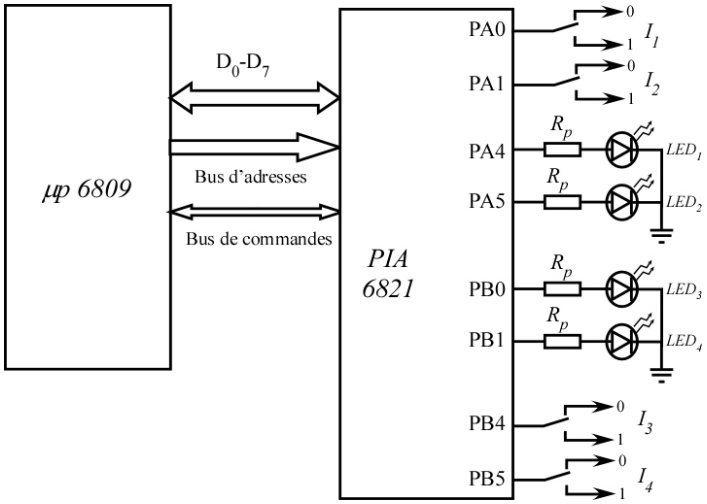


Figure 1

Exercice 7

On a le schéma suivant :

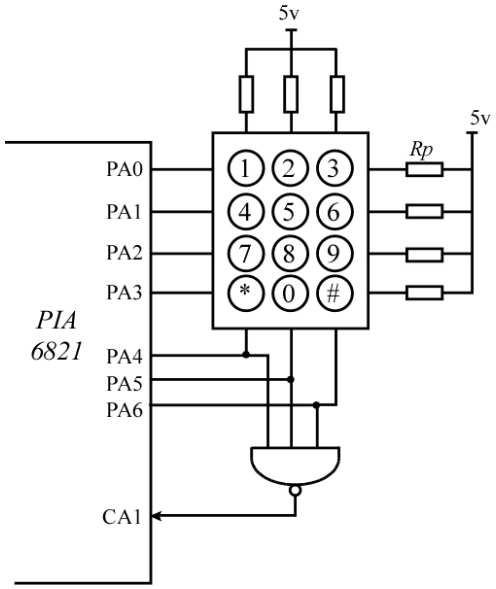


Figure 2

On veut réaliser la gestion d'un clavier téléphonique (12 touches) par un système à base d'un *µp* 6809 + PIA6821.

- 1er solution : par ***scrutation*** (sans CA1), proposer un sous programme (***SPun***) en assembleur qui effectue la détection et l'identification d'une touche actionnée ;
- 2ème solution : par ***sondage*** (avec CA1), proposer un sous programme (***SPdeux***) en assembleur qui effectue la détection et l'identification d'une touche actionnée.

Rem : dans les deux cas, on suppose que le sous programme fournit le résultat de la détection dans l'adresse ***$0380*** (le code ASCII de la touche actionnée, sinon la valeur ***$00***).

Annexe C : Le Polycopié des Travaux Pratiques SAM

Université de Mostaganem
Faculté des Sciences et de la Technologie
Département de Génie Electrique

TP n° 1

3éme année **licence ELN**
Systèmes à Microprocesseurs
(SAM)

Date : …….

Sous Groupe : ……

Objectifs du TP :

- Étude du circuit 74194 : registre à décalage bidirectionnel.
- Etude d'un compteur (7493).

	Nom & prénom	Note	Observation
1			
2			

I. Étude du circuit 74194

Charger le fichier « **~\SAM_L3\tp1a_l3** » (***figure B***), ouvrir l'analyseur logique s'il n'est pas encore ouvert, puis appuyer sur le bouton « ***BRUST*** » du *Word Generator* pour lancer la simulation vous permettant de tracer les chronogrammes des sorties du registre.

Remarques :

- ✓ N'oubliez pas d'effacer l'analyseur logique par le bouton « ***RESET*** » ;
- ✓ Pour l'analyseur, garder la base de temps « ***Clocks per Division*** » = 4 ;
- ✓ Pour le « ***Word Generator*** » : l'horloge est choisie = **4 Hz** et la valeur finale = 2**F** et la table des entrées est pré chargée par une séquence de test ;
- ✓ Le schéma bloc du circuit 74194 est donné dans la figure A ;

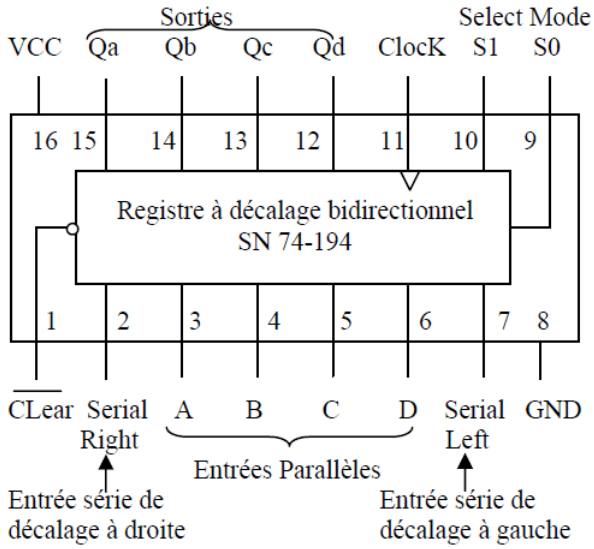


Figure A : schéma bloc du circuit 74194

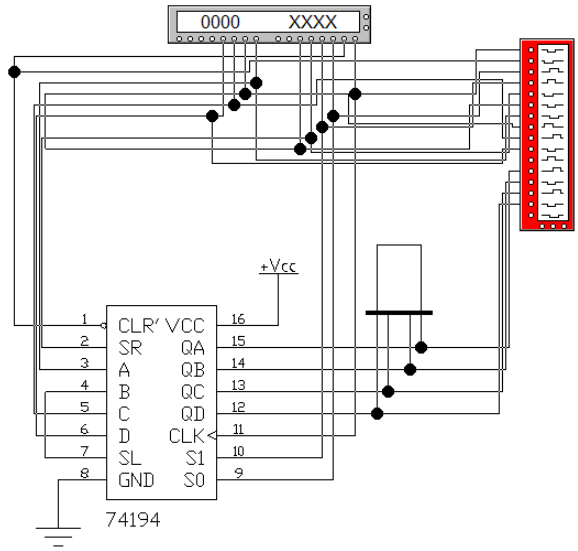
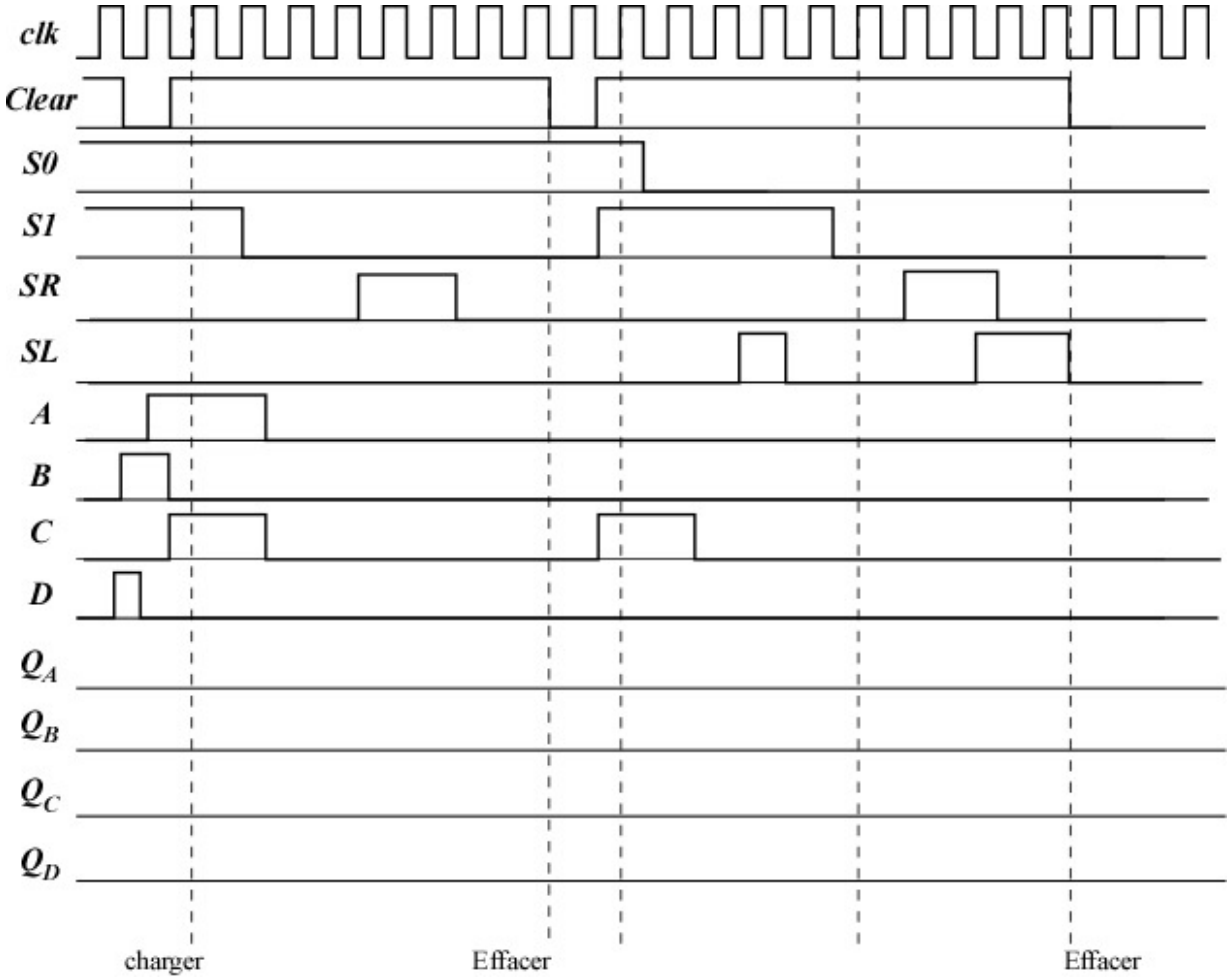


Figure B : schéma du circuit de travail

1) Compléter les chronogrammes des sorties $\mathbf{Q_A}$ à $\mathbf{Q_D}$ du registre :



2) Compléter la table de vérité traduisant le fonctionnement du registre à partir des chronogrammes tracés précédemment

Clock	*Mode*		$\overline{Clear}$	*Fonctionnement*
	S1	S0		
↑	0	0	1	
↑	0	1	1	
↑	1	0	1	
↑	1	1	1	
x	x	x	0	

II. Etude d'un compteur (7493)

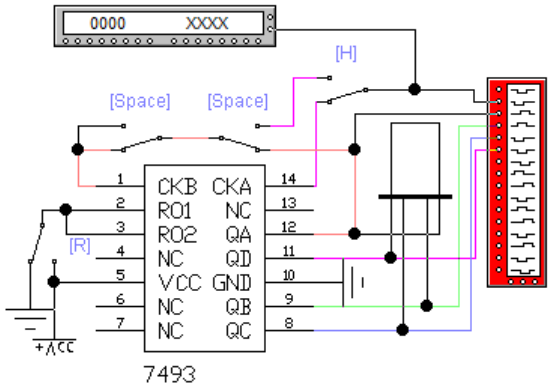
Charger le fichier «**~\SAM_L3\tp1b_l3** », ouvrir l'analyseur logique s'il n'est pas encore ouvert, puis appuyer sur le bouton « ***BRUST*** » du *Word Generator* pour lancer la simulation vous permettant de tracer les chronogrammes des sorties du compteur (garder **R01**=**R02**=0).

Remarques :

- ✓ N'oubliez pas d'effacer, à chaque fois, l'analyseur logique par le bouton « ***RESET*** » ;
- ✓ Pour l'analyseur, garder la base de temps « ***Clocks per Division*** » = 8 ;
- ✓ Pour le « ***Word Generator*** » : l'horloge est choisie = **1 Hz** et la valeur finale = **F** par contre la table des entrées est vide ;
- ✓ **CKA**, **CKB** : sont des entrées horloge (***CLOCK***) ;
- ✓ **NC** : « *No Connected* ».

Etudier les cas suivants (à l'aide des commutateurs *H* et *Space*) :

1) L'entrée **CKA** reçoit le signal d'horloge **H** et l'entrée **CKB** reçoit le signal de sortie **QA** ;
2) L'entrée **CKA** reçoit le signal d'horloge **H** et l'entrée **CKB** est non reliée ;
3) L'entrée **CKB** reçoit le signal d'horloge **H** et l'entrée **CKA** est non reliée ;



Séquences délivrées par le compteur (Séquence des nombres affichés) :

Cas 1 :……...

……

Cas 2 :…….

……

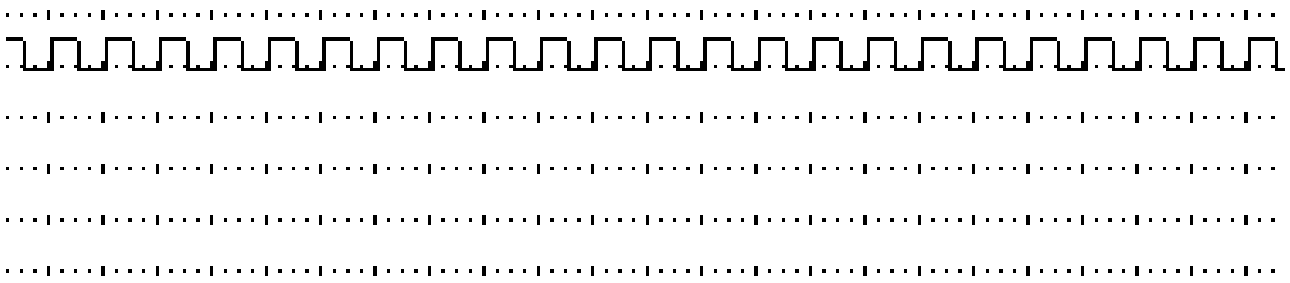
Cas 3 :…….

……..……..

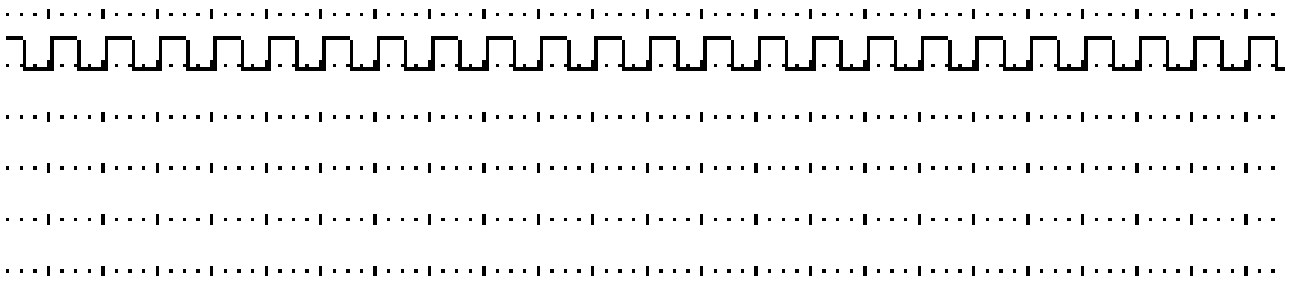
Quel est le rôle des entrées **R01** et **R02** :

……
……
……

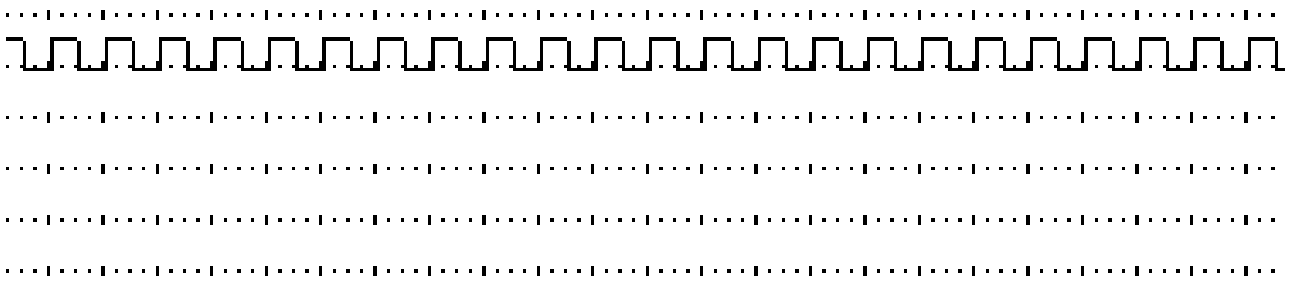
Chronogramme Cas 1 : (CKA = H, CKB = QA)



Chronogramme Cas 2 : (CKA = H, CKB Libre)



Chronogramme Cas 3 : (CKA Libre, CKB = H)



<u>Dessiner le schéma bloc du compteur 7493 :</u>

Université de Mostaganem
Faculté des Sciences et de la Technologie
Département de Génie Electrique

TP n° 2

3éme année **licence ELN**
Systèmes à Microprocesseurs
(SAM)

Date : …….

Sous-Groupe : ……

Objectifs du TP :

Le but de ce TP consiste en une prise en main du logiciel de simulation pour l'assembleur 6809 : ***MOTO6809***. Ce dernier permet la saisie d'un programme en assembleur, l'assemblage de programme et la simulation de l'exécution de programmes par le ***µp*** 6809 de **MOTOROLA**.

À l'aide d'un exercice écrit en assembleur 6809, vous devez être en mesure de maîtriser suffisamment l'utilisation du logiciel ***MOTO6809***.

	Nom & prénom	Note	Observation
1			
2			

Remarques :

1) **Avant d'effectuer le travail demandé dans le TP, il faut lire le tutoriel de présentation du logiciel qui est donné à la suite de ce document !!**

2) **A la fin de la séance du TP, vous rendez uniquement les feuilles « 1 » et « 2 » pour le compte rendu. Vous gardez les feuilles du tutorat de présentation du logiciel MOTO6809 !!**

Responsable de la matière : *Mr BENTOUMI M.*

I. Travail demandé

Une fois l'ordinateur démarré, aller dans le menu "***Démarrer***" puis "***Programmes***" et choisir le programme ***« MOTO6809*** » (ou un double clic sur l'icône ***MOTO6809*** dans le bureau 

!!).

Puis dans le menu « ***Fichier*** » ouvrer un nouveau fichier et copier le programme suivant dans l'éditeur :

```
*exercice test
;$0001 db $07
adr EQU $0020

Debut:
  LDA #$29
  LSLA
  LDB $01
  INCB
  STB $50
  ADDA $50
  STA adr
END
```

Pour comprendre l'exécution, il faut lancer l'exécution en mode pas à pas.

1. Quelle est l'adresse du programme dans la mémoire **ROM** ? (l'adresse de la première instruction)

……..

2. En observant la fenêtre **programme** et la fenêtre **ROM**, donner le code machine équivalent des instructions suivantes :

   a) **LSLA** ➔ …….

   b) **LDB** $01 ➔ ……..

   c) **END** ➔ ……..

3. Le résultat final est sauvegardé à l'adresse mémoire **RAM** « **adr** », quel est ce résultat ?

…….

II. Tutoriel sur MOTO6809

MOTO6809 est un environnement de programmation en assembleur et de simulation du microprocesseur 6809 de MOTOROLA sous Windows. L'ensemble contient un éditeur, un assembleur et un simulateur intégrant la simulation d'un circuit d'interface parallèle le PIA 6821.

Ce logiciel est fourni gratuitement et généreusement par son auteur (*http://www.siloged.fr*).

II.1 Présentation de l'interface de MOTO6809

L'environnement de développement ***MOTO6809*** est riche en fenêtres et d'autres éléments (Figure 1). Chaque élément dispose d'une Info Bulle qui apparaît lorsque le pointeur de la souris est placé pendant 1 secondes sur l'élément. Par ailleurs une ligne d'aide est affichée à la base de la fenêtre principale.

Lorsque toutes les fenêtres sont ouvertes, l'interface de ***MOTO6809*** se présente de la façon suivante :

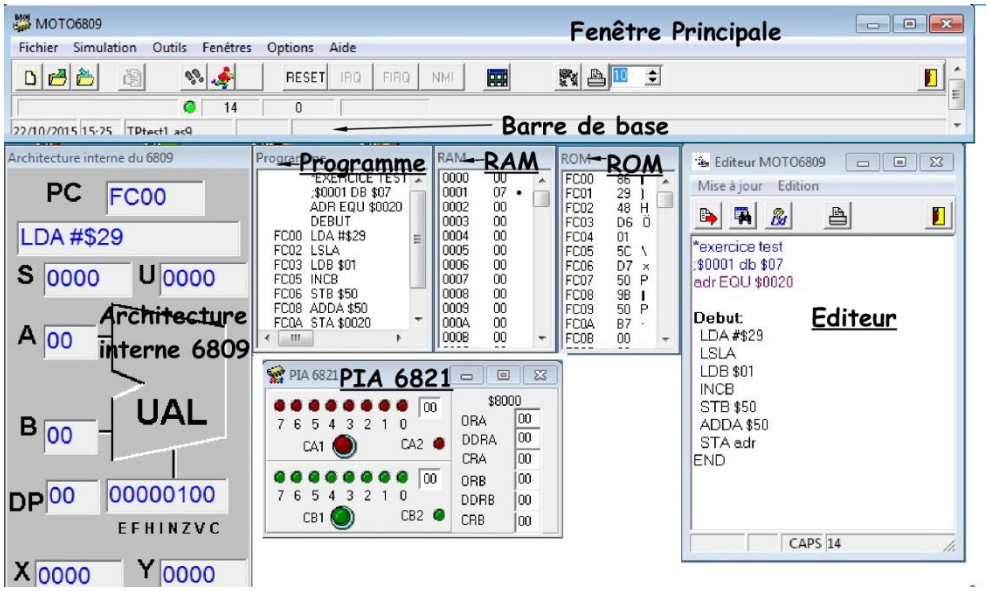


Figure 1

II.2 La fenêtre Architecture interne du 6809

Cette fenêtre présente non seulement l'architecture interne du 6809, mais aussi le contenu des différents registres internes lors de la simulation.

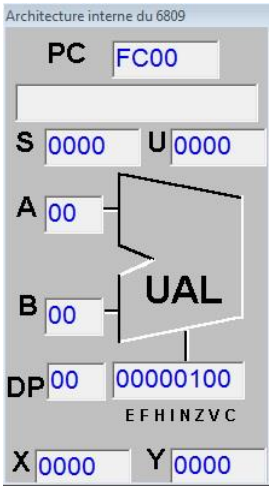


Figure 2

Dans cette fenêtre (Figure 2), on peut voir le contenu de tous les registres du ***µp*** 6809 et ainsi on peut les modifier à n'importe quel moment.

Un ***double clic*** sur la valeur actuelle d'un des registres (***A, B, DP, U, S, X, Y***) permet l'édition et la modification de sa valeur. Pour prendre en compte la nouvelle valeur hexadécimale, il suffit d'appuyer sur la touche ***ENTREE***.

Toutes les valeurs affichées dans les registres sont en hexadécimal à l'exception du registre d'état (**CC**) qui est affiché en binaire (les indicateurs "***E, F, H, I, N, Z, V, C***").

Au-dessous du registre ***PC***, il y'a un champ d'affichage de l'instruction qui va être exécutée (il joue le rôle du registre d'instruction ***RI***).

II.3 Menu Fenêtres

Le menu ***Fenêtres*** (Figure 3) permet de choisir l'affichage ou la fermeture des différentes fenêtres (***Programme***, ***RAM***, ***ROM***, ***PIA***) fenêtres du simulateur. Chaque fenêtre offre une fonctionnalité bien spécifique.

Figure 3

II.3.1 La fenêtre Programme

C'est une fenêtre à travers laquelle le programme assemblé est affiché avec l'adresse de chaque instruction dans la **ROM** (Figure 4). Dans cette fenêtre, on peut mettre des points d'arrêt pour l'exécution du programme. Un double clic sur une instruction permet de placer un point d'arrêt >> au niveau de cette instruction.

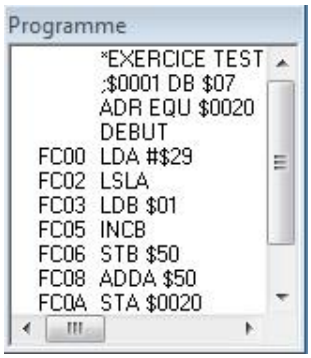


Figure 4

II.3.2 Fenêtres RAM et ROM

Le simulateur dispose d'un espace mémoire de **8Ko** pour la **RAM** adressable à partir de l'adresse **$0000** et d'un espace de **32Ko** pour la **ROM** dont l'adresse haute est impérativement **$FFFF**.

Les tailles des espaces **RAM** et **ROM** sont configurables via le menu ***Options|Configuration***.

Sur la palette **Mémoires** et à l'aide de curseurs (Figure 5), on peut changer les tailles de la **RAM** et de la **ROM**.

Par défaut, la taille de 1Ko est fixée pour la RAM ainsi que pour la ROM. Dans ce cas-là, l'adresse de début de la **ROM** est **$FC00**.

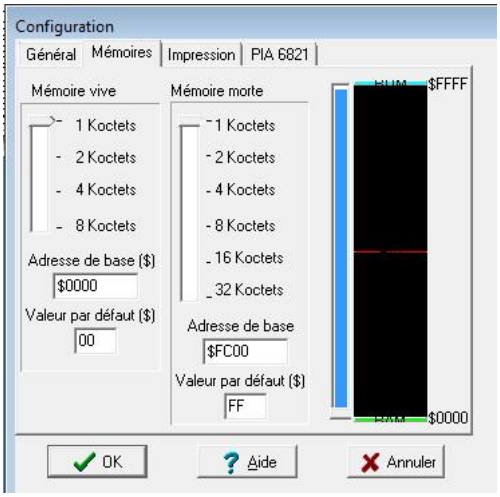


Figure 5

II.3.3 Fenêtre des entrées/sorties (PIA)

Le simulateur ***MOTO6809*** intègre une fenêtre graphique qui simule le circuit d'interface le ***PIA6821*** (Figure 6) qui comporte deux ports d'entrées/sorties (***PortA*** (couleur rouge) et ***PortB*** (couleur verte)). Le PIA***6821*** comporte 06 registres adressables par défaut à partir de l'adresse **$8000**.

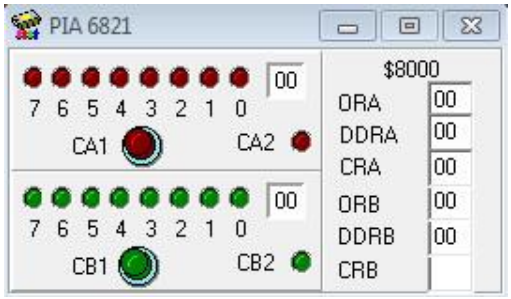


Figure 6

Le fonctionnement de cette fenêtre sera abordé en temps venu lors des TPs correspondants.

II.4 Description des opérations de base du MOTO6809

II.4.1 Créer un nouveau fichier source

Aller au menu ***Fichier***|***Nouveau*** ( ), une nouvelle fenêtre de l'éditeur sera ouverte (Figure 7) dans laquelle on écrit notre programme pour le ***µP*** 6809.

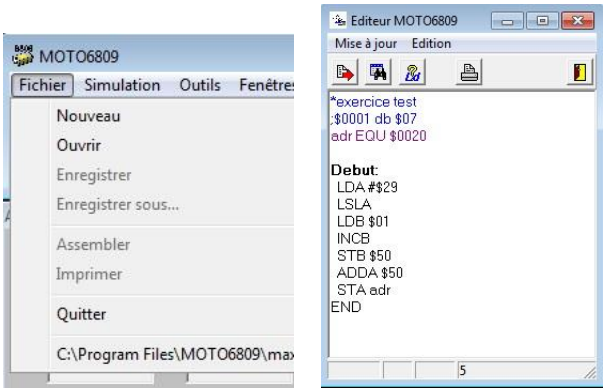


Figure 7

II.4.2 Ouvrir un fichier existant

Aller au menu ***Fichier|Ouvrir*** () et une fenêtre de dialogue apparaît pour choisir le fichier voulu avec l'extension « **.as9** ». Si l'éditeur n'est pas ouvert automatiquement, alors on peut l'ouvrir à l'aide de la commande du menu ***Outils|Editeur*** ou en cliquant sur l'icône  de la barre des icones de la fenêtre principale.

II.4.3 Sauvegarder un fichier

Aller au menu ***Fichier|Enregistrer*** () ou ***Fichier| Enregistrer sous*** pour sauvegarder le fichier actif. L'extension des fichiers est par défaut « ***.as9*** ».

II.4.4 Assemblage du fichier source

Pour lancer la traduction du fichier texte écrit (programme assembleur) dans l'éditeur, il suffit de cliquer sur l'icône  de la barre des icones de l'éditeur.

Suite à cette action, le programme écrit est traduit en langage machine et chargé dans la mémoire **ROM** s'il n'y a pas d'erreur. Sinon un message d'erreur s'affiche.

II.4.5 Exécution d'un programme

Pour exécuter un programme, on peut choisir ***Exécuter*** du menu ***Simulation***, appuyer sur ***Ctrl+F9*** ou cliquer sur le bouton d'exécution  de la barre des icones de la fenêtre principale.

Exécuter : commence l'exécution du programme à partir de l'adresse spécifiée dans le compteur de programme (PC). L'exécution du programme s'arrête lorsque :

- une instruction ***SWI*** est exécutée, la directive ***END*** est remplacée par l'instruction ***SWI*** par le logiciel d'assemblage ;
- un point d'arrêt est rencontré ;
- l'utilisateur interrompre le programme par la commande ***Simulation |Reset*** () ;

II.4.6 Exécution jusqu'au point d'arrêt

Permet l'exécution du programme à partir de l'adresse courante du registre **PC** (compteur de programme) jusqu'à l'instruction pointée par le symbole >> placée par un double clic sur l'instruction dans la fenêtre ***Programme*** (Figure 8).

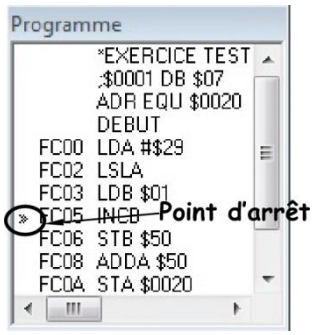


Figure 8

II.4.7 Le mode pas à pas

Le mode pas à pas permet d'exécuter le programme instruction par instruction. On peut choisir ce mode en pressant la touche ***Ctrl+F10*** ou cliquant sur le bouton de commande ().

Remarque : n'hésitez pas à utiliser l'**Aide** du logiciel pour plus d'information sur ***MOTO6809*** !!

TP n° 3

Date : …….

Sous Groupe : ……

Objectifs du TP :

- Comprendre le rôle du registre **PC** (*Program Counter*).
- Comprendre les instructions de chargement (registres, mémoire).
- Comprendre les indicateurs du registre d'états **CCR**
- Comprendre les modes ***d'adressage immédiat***, ***absolu directe*** et ***étendu*** du ***µp*** 6809.

	Nom & prénom	Note	Observation
1			
2			

Exercice 1

Commenter le déroulement des instructions suivantes en complétant les champs libres correspondant au contenu des registres **PC**, **A** et **B** ainsi que l'état des indicateurs **N, Z, V, C** du registre d'état (**CCR**).

Conditions initiales : le registre **A** contient la valeur **$FF ;**

Etiquettes	Op.	Opérandes	Registre PC	Registres A et B	Indicateurs N	Z	V	C	Commentaires
			PC =	A=$**FF** , B=$00					
Debut:									
	LDA	#$48	PC =	A= , B=					
	ADDA	#$21							
	LDB	#$C1							
	ADDB	#$FF							
	LDB	#$C1							
	SUBB	#$01							
	LDD	#$A03C							
	ADDD	#$80D2							
	END								

1- Quel mode d'adressage est utilisé pour les instructions du programme précédent ?

……..

2- Quel est le rôle du registre PC ?

……

……

3- Expliquer les valeurs des indicateurs après l'exécution de la dernière instruction (ADDD) ?

……

……

……

……

……

……

Exercice 2

Exécuter le programme ci dessous en notant les résultats pour chaque instruction et répondre aux questions suivantes :

1- Quelle est la différence entre les deux instructions suivantes :
   - STA $0002
   - STA $04

……

……

……

……

2- Quelle est l'opération effectuée par l'instruction « ***STD $02*** » et quel est son résultat ?

……

……

……

……

3- Pour quoi a-t'on utilisé l'instruction ***TFR A, DP*** ?

……

……

……

Etiquettes	**Op.**	**Opérandes**	**Résultats**	**Indicateurs** N	Z	V	C
Debut:	**LDA**	#$08	A= , B=				
	STA	$0002	Mémoire à partir de l'adresse $0000				
	STA	$04	Mémoire à partir de l'adresse $0000				
	INC	$0002	Mémoire à partir de l'adresse $0000				
	LDB	#$F7	A= , B=				
	ADDB	$02	A= , B=				
	LDA	#$01					
	TFR	A, DP					
	STD	$02					
	END						

Exercice 3

On veut réaliser l'addition entre le contenu du registre **X** avec le contenu du registre **Y** et le résultat doit être sauvegardé à partir de la case mémoire [$0020].

Compléter le programme ci dessous et tester le dans le simulateur MOTO6809.

Debut :

LDX #$03FF
LDY #$0D01

END

TP n° 4

Date : …….

Sous Groupe : ……

Objectifs du TP :

- Comprendre les ***directives d'assemblage***.
- Comprendre les instructions de calcul.

	Nom & prénom	Note	Observation
1			
2			

Exercice 1

Editer le programme suivant et répondre aux questions :

** Après assemblage et chargement du programme, quelle est le résultat d'exécution des directives suivantes :*

;$0010 **DB** $4C	……
;$0011 **DB** $B2	……
;$0012 **DB** $35	……
	……
	……
val1 **EQU** $F1	……
val2 **EQU** $2E51	……
	……
;$FFFE **DB** $FC	……
;$FFFF **DB** $20	……
	……

** Exécuter le programme suivant et commenter le résultat de chaque instruction :*

ORG $FC20	……
LDA #val1	……
ANDA #%00001111	……
STA $20	……
LDX #$0010	……
LEAY 2,X	……
INCA	……
STA ,X	……
LDD #val2	……
STB ,Y	……
END	……

Exercice 2

Ecrire un programme en assembleur qui effectue la somme suivante : $S = a + b + c$ et vérifier l'exécution dans ***MOTO6809***. Les nombres ***a***, ***b*** et ***c*** sont ***des nombres signés,*** codés sur 8 bits et stockée en mémoire aux adresses ***$0080***, ***$0081*** et ***$0082*** respectivement. Le résultat ***S*** doit être stocké en mémoire à partir de l'adresse ***$0010***.

Exercice 3

Ecrire un programme en assembleur ***6809*** qui effectue le calcul de : $p = \frac{a \times b}{2}$.

où ***a*** et ***b*** (stockés en mémoire à l'adresse ***ada*** et ***adb***) sont ***des nombres non signés*** sur 8 bits et ***p*** est le résultat sur 16 bits qui doit être stocker en mémoire à partir de l'adresse ***adp***.

Vérifier avec l'exemple suivant : ***a*** = 9 et ***b*** = 4.

Université de Mostaganem
Faculté des Sciences et de la Technologie
Département de Génie Electrique

TP n° 5

3éme année **licence ELN**
Systèmes à Microprocesseurs
(SAM)

Date : …….

Sous Groupe : ……

Objectifs du TP :

- Comprendre le fonctionnement des instructions de comparaison et celles de test et de branchement ;
- Utiliser l'adressage indexé post-incrémenté et pré-décrémenté ;
- Comprendre le fonctionnement des instructions d'appel et de retour de sous programme ;

	Nom & prénom	Note	Observation
1	……		
2	……..		

Exercice 1

Ecrire un programme en assembleur 6809 qui effectue l'initialisation avec la valeur $**C33C** d'un tableau de ***N*** mots (16 bits) situé en mémoire à l'adresse ***adtab***.

Application : ***adtab*** = $0300 ; ***N*** = 50.

Exercice 2

On considère qu'une liste de 10 ***données non signées*** codées sur **8** bits est rangée en mémoire à partir de l'adresse **$0020**. Développer un programme qui recherche la valeur maximale et la valeur minimale dans cette liste et ranger ces deux résultats respectivement dans les adresses **$0010** et **$0012**.

Rem. : *Vérifier le programme en initialisant la liste au préalable.*

Exercice 3

Ecrire un programme en assembleur qui effectue le calcul suivant :

$Sum = \left(a^2 - 4a + 1\right) + \left(b^2 - 4b + 1\right) + \left(c^2 - 4c + 1\right)$; où ***a***, ***b*** et ***c*** sont des nombres non signés sur 8 bits.

Utiliser ***un sous programme*** (S.P) pour optimiser la taille du programme global.

Rem. : *attention au format de Sum (> 16 bits).*

References

1. Letocha , « Introduction aux circuits logiques », Mc-Graw Hill.
2. J.M. Bernard, J. Hugon, « De la logique câblée aux microprocesseurs, Tomes 1 à 4 » Eyrolles.
3. R. Delsol , « Electronique numérique, Tomes 1 et 2 » Edition Berti.
4. P. Cabanis, « Electronique digitale » Edition Dunod.
5. M. Gindre, « Logique séquentielle » Edition Ediscience.
6. J. P. Vabre et J. C. Lafont , « Cours et problèmes d'électronique numérique » Ellipses, 1998.
7. R. Katz, "Contemporary Logic Design", 2nd ed. , Prentice Hall, 2005.
8. M. Aumiaux, « L'emploi des microprocesseurs » Masson, Paris, 1982.
9. M. Aumiaux, « Les systèmes à microprocesseurs », Masson, Paris, 1982.
10. R.L. Tokheim, « Les microprocesseurs, Tomes 1 et 2 » série Schaum, McGraw Hill.
11. J.C. Buisson, « Concevoir son microprocesseur, structure des systèmes logiques » Ellipses, 2006.
12. A. Tanenbaum, « Architecture de l'ordinateur » Dunod.
13. P. Zanella, Y. Ligier, E. Lazard, «Architecture et technologie des ordinateurs » Dunod.
14. J.M. Trio, « Microprocesseurs 8086-8088 : Architecture et programmation, Coprocesseur de calcul 8087 », Eyrolles.
15. H. Lilen, « Cours fondamental des microprocesseurs » Dunod, 1993.
16. J.C. Buisson, « Concevoir son microprocesseur : Structure des systèmes logiques » Ellipses, 2006.
17. H. DRIAS-ZERKAOUI, Introduction à l'architecture des ordinateurs, 4ème édition, 2010, OPU - Algérie.
18. Andrew TANENBAUM, Architecture de l'ordinateur, 3ème édition, 1991, InterEditions, Paris.
19. Pierre Alain Groupille, Technologie des ordinateurs et des réseaux.
20. Draoua Abdelkader, Notes de cours Microprocesseur 68000, USTOran - Algérie.

Résumé — Ceci est le cours de la matière SAM du niveau licence L3 Electronique.

Mots clés : Microprocesseur, 6809, Assembleur, programme, codage binaire, interfaces d'entrées/sorties.

Département de Génie Électrique
Faculté des Sciences et de la Technologie
Université Abdelhamid Ibn Badis de Mostaganem
Site 1 - Route Belhacel
27000 Mostaganem, Algérie