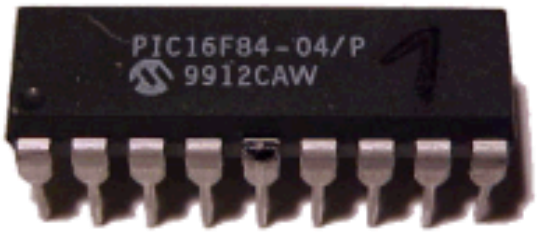
Le Microcontrôleur 16F84 de Microchip



M. EL ALAMI

Sommaire

Introduction

L'année 1971 fut marquée par la fabrication du premier microprocesseur par la société INTEL. Depuis, la plupart des réalisations électroniques dans tous les domaines de l'industrie font appel à ces composants miracles qui peuvent gérer n'importe quel automatisme. L'avantage principal du microprocesseur est que celui-ci travail avec un programme logé dans une mémoire, donc modifiable. C'est le principe de l'automate programmable. La logique câblée, reste un peu figée face à ces nouveaux circuits.
L' avènement des microcontrôleurs, qui associent au microprocesseur de base un programme intégré au circuit, ainsi que des périphériques et de la RAM, a permis de faire évoluer les montages vers plus de simplicité et de rapidité (les périphériques étant intégrés).
Les microcontrôleurs sont aujourd'hui implantés dans la plupart des applications grand public ou professionnelles, il en existe plusieurs familles.
La société Américaine Microchip Technologie a mis au point dans les années 90 un microcontrôleur CMOS: le PIC, ce composant encore très utilisé à l'heure actuelle, est un compromis entre simplicité d'emploi, rapidité et le coût.
Les PIC font partit de la famille des microcontrôleurs, ils possèdent un jeu d'instructions réduit qui caractérise les circuits RISC (Reduced Instruction Set Computer). Les circuits RISC sont caractérisés par leur rapidité d'exécution.

I- Généralité sur les PIC

Les PIC existent dans plusieurs versions:
- les **UVPROM** qui sont effaçable par une source de rayonnements ultraviolets
- les **OTPROM** programmable une seule fois
- les **EEPROM** (**E^2PROM**) et flash **EPROM** qui sont effaçables électriquement et reprogrammable

De nombreux outils de développement sont proposés sur le marché, il est également possible de télécharger des logiciels auprès de la société Microchip.

I-1 Les différentes familles de PIC

La famille des PIC était subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits pour certains PIC (12C508), de 14bits pour d'autres PIC(12F675), la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie les 16F84 et 16F876), et la famille **High-End**, qui utilise des mots de 16 bits. Par la suite, d'autres familles sont apparues, comme la Enhanced family, et les choses ne devraient faire qu'évoluer.
Nous nous limiterons dans ce cours à la famille **Mid-Range**, sachant qu'il n' y a pas de grande différence entre ces familles.

Tableau de quelques familles de Base Line et Mid Range

PIC	ROM	RAM	Commentaires	Timer	E-S
PIC 12C508	512 x 12	25 x 8		1 8bits	6
PIC 12C509	1024 x 12	41 x 8		1 8bits	6
PIC 12CE518	512 x 12	25 x 8	E²PROM : 16 x 8	1 8bits	6
PIC 12CE519	1024 x 12	41 x 8	E²PROM : 16 x 8	1 8bits	6
PIC 12C671	1024 x 14	128 x 8	4 ADC 8 bits	1 8bits	6
PIC 12C672	2048 x 14	128 x 8	4 ADC 8 bits	1 8bits	6
PIC 12C673	1024 x 14	128 x 8	E²PROM : 16 x 8 -4 ADC 8 bits	1 8bits	6
PIC 12C674	2048 x 14	128 x 8	E²PROM : 16 x 8 -4 ADC 8 bits	1 8bits	6
PIC 16C52	384 x 12	25 x 8		1 8bits	12
PIC 16C54	512 x 12	25 x 8		1 8bits	12
PIC 16C55	512 x 12	24 x 8		1 8bits	21
PIC 16C56	1k x 12	32 x 8		1 8bits	13
PIC 16C57	2k x 12	80 x 8		1 8bits	21
PIC 16C58	2k x 12	80 x 8		1 8bits	21
PIC 16C62A	2048 x 14 (flash)	128 x 8		1 16bits 2 8bits	22
PIC 16C63	4096 x 14 (flash)	192 x 8		1 16bits 2 8bits	22
PIC 16C64	2048 x 14	128 x 8		1 16bits 2 8bits	33
PIC 16C65A	4096x 14 (flash)	192 x 8		1 16bits 2 8bits	33
PIC 16C66	8192 x 14	128 x 8		1 16bits 2 8bits	22
PIC 16C67	8192 x 14	128 x 8		1 16bits 2 8bits	33
PIC 16C71	1024 x 14	36 x 8	4 canaux ADC	1 8bits	13
PIC 16C72	2048x 14	128 x 8	5 canaux ADC	1 16bits 2 8bits	22
PIC 16C73A	4096 x 14	192 x 8	5 canaux ADC	1 16bits 2 8bits	22
PIC 16C74A	4096 x 14	192 x 8	8 canaux ADC	1 16bits 2 8bits	33
PIC 16C76	8192 x 14 (flash)	368 x 8	5 canaux ADC	1 16bits 2 8bits	22
PIC 16C77	8192 x 14	368 x 8	8 canaux ADC	1 16bits 2 8bits	33
PIC 16F83	512 x 14 (flash)	36 x 8	E²PROM : 64 x 8		13
PIC 16C84	1024 x 14	68 x 8	E²PROM : 64 x 8		13
PIC 16F84	1024 x 14 (flash)	68 x 8	E²PROM : 64 x 8		13

I-2 Identification d'un PIC

Un PIC est identifié par un numéro de la forme suivant : xxXXyy –zz

- xx : Famille du composant (12, 14, 16, 17, 18)
- XX : Type de mémoire de programme
  - C - EPROM ou EEPROM
  - CR - PROM
  - F - FLASH
- yy : Identification
- zz : Vitesse maximum du quartz

Nous utiliserons un PIC 16F84 –10, soit :

- 16 : Mid-Line
- F : FLASH
- 84 : Type
- 10 : Quartz à 10MHz au maximum

I-3 Exemples d'applications

Les applications à base de PIC sont multiples, ces microcontrôleurs offrent de nombreuses entrées - sorties selon les modèles et ils permettent de faire réagir un programme selon l'état des événements rencontrés.
Les sorties des PIC peuvent délivrer environ 25 mA, on peut au moyen de transistors, amplifier ce courant de sortie pour piloter un relais ou bien tout autre composant électrique ou électronique.
Pour les applications de type analogiques, certain PIC possèdent des convertisseurs intégrés, il est possible également d'adjoindre un circuit annexe (CNA ou CAN) qui dialoguera avec le microcontrôleur et effectuera les conversions analogiques.

II- Etude du PIC 16F84

Il s'agit d'un microcontrôleur 8 bits à 18 pattes. Les principales caractéristiques de ce PIC sont :

- 35 instructions
- Mémoire Programme : une EEPROM de type Flash ROM et de capacité 1K mots de 14 bits
- 2 Mémoires de données : une mémoire vive **SRAM volatile** de 68 octets pour les données temporaires, une mémoire **EEPROM non volatile** de 64 octets pour les données permanentes
- Système de registres : **Un registre de travail** (Work register) non adressable de 8 bits : Accumulateur appelé W.
15 registres adressables de 8 bits pour les besoins de la programmation du PIC
- **Horloge :** La valeur habituelle est de 4 Mhz, il existe des 16F84 acceptants une fréquence maximale de 10 MHz
- 1 cycle machine par instruction, sauf pour les sauts (2 cycles machine)
- 4 sources d'interruption
- 1000 cycles d'effacement/écriture pour la mémoire flash, 10.000.000 pour la mémoire de donnée EEPROM

II.1. Brochage et fonction externe des pattes

La Figure II.1 montre le brochage du circuit de PIC 16F84. Les fonctions des pattes sont les suivantes :

- V_{SS}, V_{DD} : Alimentation
- OSC1, 2 : Horloge
- RA0-4 : Port A
- RB0-7 : Port B
- T0CKL : Entrée de comptage
- INT : Entrée d'interruption
- MCLR : Reset : 0V

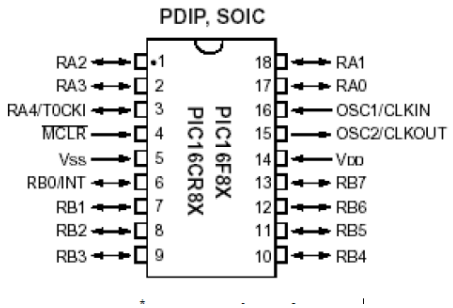


Figure II.1 : Brochage du PIC 16F84

a) Broches d'entrées/Sorties

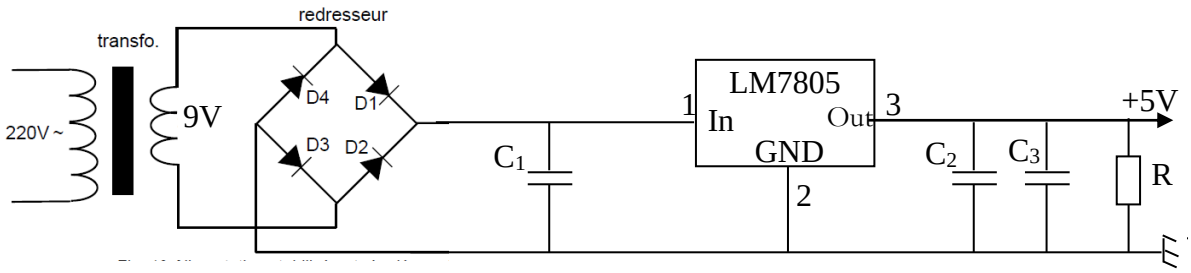
Broches	Nom	Fonction	Sens
1	RA2	Port A (E/S)	Bidirectionnel
2	RA3	Port A (E/S)	Bidirectionnel
3	RA4/TOCK1	Port A (E/S)	Bidirectionnel
6	RB0/INT	Port B (E/S)	Bidirectionnel
7	RB1	Port B (E/S)	Bidirectionnel
8	RB2	Port B (E/S)	Bidirectionnel
9	RB3	Port B (E/S)	Bidirectionnel
10	RB4	Port B (E/S)	Bidirectionnel + interruption par changement d'état
11	RB5	Port B (E/S)	Bidirectionnel + interruption par changement d'état
12	RB6	Port B (E/S)	Bidirectionnel + interruption par changement d'état
13	RB7	Port B (E/S)	Bidirectionnel + interruption par changement d'état
17	RA0	Port A (E/S)	Bidirectionnel
18	RA1	Port A (E/S)	Bidirectionnel

Ces broches sont bidirectionnelles, leurs configurations en entrée ou en sortie est donc programmable (voir description des registres)
L'interruption par changement d'état peut être utilisée pour détecter les interruptions externes (voir interruption)

b) Broches d'alimentation

Broche	Nom	Fonction	Valeur typique	Maximum
14	VDD	Alimentation	+ 5 Volts	+7. 5 Volts
5	VSS	Alimentation	0 Volts	-

Ces deux broches sont connectées à une sourde d'alimentation :



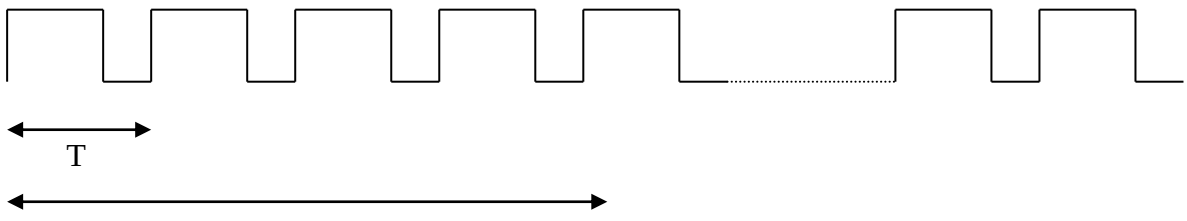
C1 = 22µF, C2 = 100 nF
C3 = 10 µF, R = 1 KΩ

Le **LM7805** est un régulateur de tension permettant de stabiliser le niveau de tension à 5V. Ce système peut être remplacé par une simple pile connectée entre les broches 5 et 14

c) Broches de l'oscillateur

Broche	Nom	Fonction
15	OSC2 **Ou** CLKOUT	Sortie vers le quartz **Ou** Sortie horloge voir oscillateur
16	OSC1 **Ou** CLKIN	Entrée du quartz **Ou** Entrée horloge pour horloge externe

C'est la fréquence de l'oscillateur qui détermine à quelle vitesse seront exécutées les instructions. La fréquence interne est égale au quart de la fréquence horloge fournie par un oscillateur : Fi = Fosc/4
En général, l'exécution d'une instruction simple nécessite un cycle machine. Ce cycle utilise quarte périodes d'horloge. Avec un quartz de 4 MHz, le 16F84 fonctionne à 1 MHz (il peut exécuter globalement un million d'instruction simples par second).



Cycle machine

Remarque : Certaines instructions peuvent nécessiter selon le cas, soit 1 soit 2 cycles d'horloge. Par exemple, les instructions de branchements qui nécessitent 2 cycles machines (8 périodes)
Pour générer le signal d'horloge, plusieurs types d'oscillateurs peuvent être réalisée soit avec un quartz (figure 1) soit avec une horloge extérieure (figure 2) ou bien soit avec un circuit RC (figure 3) , dans ce dernier cas, la stabilité du montage est limité.

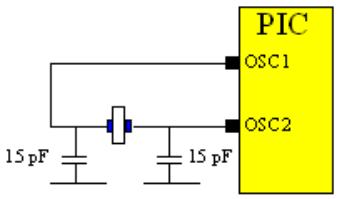


Figure - 1

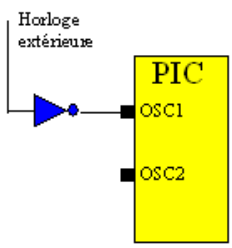


Figure - 2

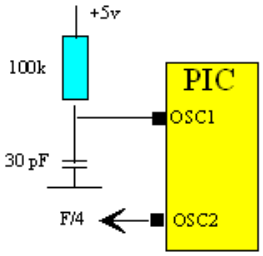


Figure - 3

La fréquence maxi d'utilisation va dépendre du microcontrôleur utilisé. Le suffixe indiqué sur le boîtier donne la nature de l'horloge à utiliser et sa fréquence maxi.
Suffixe :
LP (Low Power) oscillateur à quartz fmax : 200 khz
RC oscillateur RC fmax : 4 Mhz
XT oscillateur à quartz fmax : 4 Mhz
HS (High Speed) oscillateur à quartz rapide fmax : 20 Mhz
Un nombre suit le suffixe, il donne la fréquence d'utilisation pour le circuit spécifié :
02 2 Mhz pour les suffixes XT et RC
04 4 Mhz pour les suffixes XT et RC (le plus répandu)
10 10 Mhz pour le suffixe HS
16 16 Mhz pour le suffixe XT
20 20 Mhz pour le suffixe HS
25 25 Mhz pour le suffixe XT
33 33 Mhz pour le suffixe XT
Pour plus de renseignements se reporter à un DATA BOOK de Microchip ou bien se connecter sur le site à l'adresse: www.microchip.com

d) Broche entrée Timer

Broches	Nom	Fonction	Sens
3	TOCK1	Entrée Timer	Entrée

Cette broche (si la 5eme E/S du port A n'est pas utilisée) représente une entrée d'horloge pour le Timer (voir configuration interne)

e) Broche entrée programmation du PIC

Broches	Nom	Fonction
12	RB6	Programmation Clock

Cette broche configurée en entrée fournie l'horloge de la programmation du PIC par le programmateur (Ecriture du programme de l'application dans la Flash ROM : gravure)

f) Broche Reset

Broches	Nom	Fonction	Sens
4	MCLR	Master Clear Reset	Entrée

Cette broche est active au niveau bas, sa mise à la masse permet d'initialiser le PIC. Le pointeur d'instruction sera chargé par le contenu de la case mémoire 0000H (voir configuration interne). Cette adresse constitue l'emplacement de la 1ére instruction exécutable du programme de l'application.

g) Broches d'interruption :

Il y a plusieurs possibilités d'interruption

- Interruption Externe **RB0/INT** (broche N°6). Dans ce cas, cette broche ne représente plus une E/S du port B. Elle sera connectée à un périphérique externe capable de générer un signal d'interruption pour accéder au PIC.
- Dépassement du Timer : Passage de FFh à 00h du compteur de 8 bits (voir configuration interne)
- Par changement d'état : Les E/S du port B peuvent produire des interruptions par changement d'état : RB4, RB5, RB6, RB7 (broches 10 à 13) sont concernées, il faut qu'ils soient configurés en entrées. L'interruption se produit lorsque le signal sur une de ces entrées change pendant une durée au moins égale à la durée d'un cycle.
- Interruption à la fin du cycle d'écriture d'une donnée dans l4EEPROM (voir configuration interne)

Si l'interruption est autorisée, l'exécution du programme en cours est interrompue et le PIC effectue les opérations suivantes :

- Le pointeur d'instruction est empilé dans la pile. Il sera ensuite chargé par le contenu du vecteur d'interruption logé dans la case mémoire 0004H (voir mémoire)
- Les interruptions sont masquées
- A la fin de la routine d'interruption (retour), le pointeur d'instruction est dépilé. Le PIC reprend l'exécution du programme interrompu
- Les interruptions sont réautorisées

Remarque : Les registres dont les contenues sont nécessaires pour la suite de l'exécution du programme interrompu doivent être sauvegardés par l'utilisateur dans des variables mémoires.

II.2. Configuration interne du PIC 16F84

La figure II.2 présente l'architecture générale du circuit. Il est constitué des éléments suivants:

- un système d'initialisation à la mise sous tension (power-up timer, …)
- un système de génération d'horloge à partir du quartz externe (timing génération)
- une unité arithmétique et logique (ALU)
- une mémoire flash de programme de 1k "mots" de 14 bits
- un compteur de programme (program counter) et une pile (stack)
- un bus spécifique pour le programme (program bus)
- un registre contenant le code de l'instruction à exécuter
- un bus spécifique pour les données (data bus)
- une mémoire RAM contenant
    - les SFR
    - 68 octets de données
- une mémoire EEPROM de 64 octets de données
- 2 ports d'entrées/sorties
- un compteur (Timer)
- un chien de garde (watchdog)

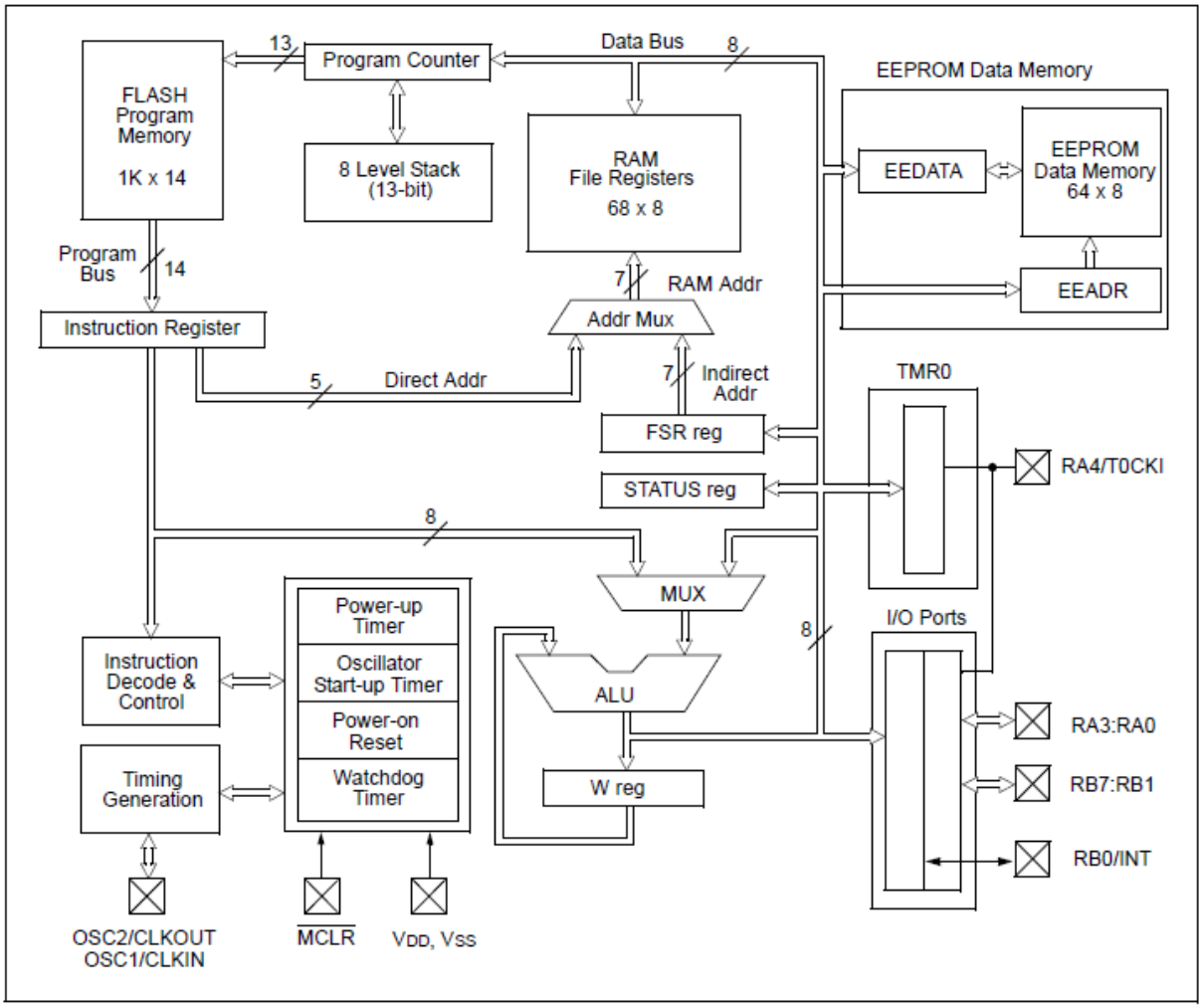


Figure II.2 : architecture générale du PIC 16F84

II.2.1 Organisation de la mémoire centrale

a) Mémoire programme

Cette mémoire est de type FlashROM de 1 K mots de 14 bits chacun. Ce type de mémoire est en réalité une EEPROM technologiquement améliorée de milliers de cycles de la reprogrammation. Elle supporte plusieurs centaines de milliers de cycles de reprogrammations elle est donc non volatile, effaçable et réinscriptible électriquement. Le programme reste donc stocké en l'absence de l'alimentation. Elle a un bus d'adresses de 13 bits et un bus de données interne de 14 bits. Ainsi, les instructions sont codées sur 14 bits.

Adresse en décimal	Adresse en hexadécimal	Contenu sur 14 bits
0	0000h	Reset vector
1	0001h	Instruction programme
2	0002h	Instruction programme
3	0003h	Instruction programme
4	0004h	Interrupt vector
5	0005h	Instruction programme
……	…….	Instruction programme
1023	03FFh	Instruction programme

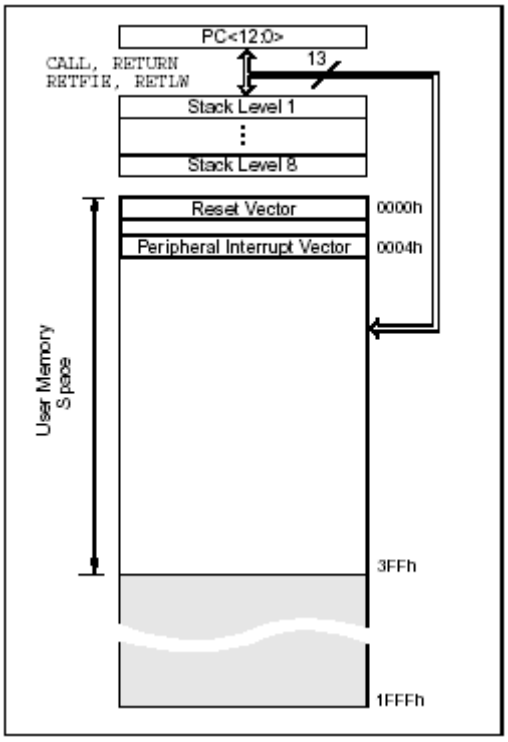


Figure II.3 : Configuration de la mémoire EEPROM

Cette mémoire programme peut être protégée contre la lecture. Elle contient deux cases spécifiques :

- 0000h : le contenu de cette case représente l'adresse de la 1 ére instruction du programme de l'application (Vecteur Reset). Le pointeur d'instruction sera chargé par cette adresse lors d'une remise à zéro de la broche MCLR ou à la suite de la mise sous tension du PIC. Le microcontrôleur commencera l'exécution du programme de l'application
- 0004h : le contenu de cette case représente l'adresse de la 1 ére instruction de la routine de traitement de l'interruption (vecteur d'interruption). Le pointeur d'instructions sera chargé par cette adresse lorsqu'une interruption est détectée au niveau de la broche INT.

b) Mémoire de données

Elle se décompose en deux parties de RAM et une zone EEPROM. La première contient les SFRs (Special Function Registers) qui permettent de contrôler les opérations sur le circuit. La seconde contient des registres généraux, libres pour l'utilisateur. La dernière contient 64 octets.

Comme nous le verrons par la suite, les instructions orientées octets ou bits contiennent une adresse sur 7 bits pour désigner l'octet avec lequel l'instruction doit travailler. D'après la Figure II.4, l'accès au registre TRISA d'adresse 85h, par exemple, est impossible avec une adresse sur 7 bits. C'est pourquoi le constructeur a défini deux banques. Le bit RP0 du registre d'état (STATUS.5) permet de choisir entre les deux. Ainsi, une adresse sur 8 bits est composée de RP0 en poids fort et des 7 bits provenant de l'instruction à exécuter.

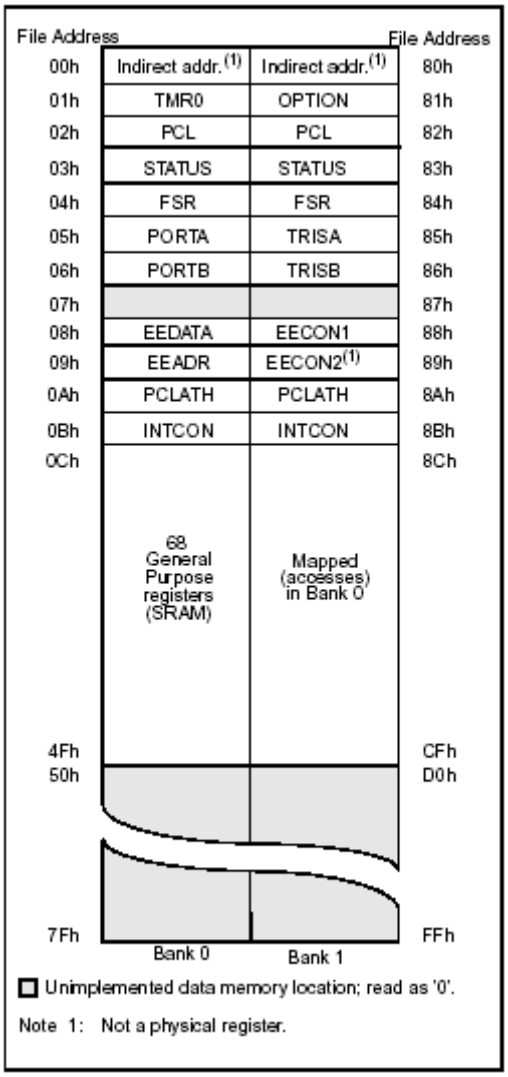


Figure II.4 : Organisation de la mémoire de données

c) Mémoire EEPROM

Le PIC possède une zone EEPROM de 64 octets accessibles en lecture et en écriture par le programme. On peut y sauvegarder des valeurs, qui seront conservées même si l'alimentation est éteinte, et les récupérer lors de la mise sous tension. Leur accès est spécifique et requiert l'utilisation de registres dédiés. La lecture et l'écriture ne peuvent s'exécuter que selon des séquences particulières.

II.3. Registres

a) Registres généraux

Ils sont accessibles soit directement soit indirectement à travers les registres FSR et INDF.

b) Registres spéciaux - SFRs

Ils permettent la gestion du circuit. Certains ont une fonction générale, des autres une fonction spécifique attachée à un périphérique donné. La Figure II.5 donne la fonction de chacun des bits de ces registres. Ils sont situés de l'adresse 00h à l'adresse 0Bh dans la banque 0 et de l'adresse 80h à l'adresse 8Bh dans la banque 1. Les registres 07h et 87h n'existent pas

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on condition.
Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.
2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.
3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

Figure II.5 : Description des SFR.

INDF (00h - 80h) : Utilise le contenu de FSR pour l'accès indirect à la mémoire.
TMR0 (01h) : Registre lié au compteur.

PCL (02h - 82h): Contient les poids faibles du compteur de programmes (PC). Le registre PCLATH (0Ah-8Ah) contient les poids forts.

STATUS (03h - 83h) : Il contient l'état de l'unité arithmétique et logique ainsi que les bits de sélection des banques.

FSR (04h - 84h) : Permet l'adressage indirect

PORTA (05h) : Donne accès en lecture ou écriture au port A, 5 bits. Les sorties sont à drain ouvert. Le bit 4 peut être utilisé en entrée de comptage.

PORTB (06h) : Donne accès en lecture ou écriture au port B. Les sorties sont à drain ouvert. Le bit 0 peut être utilisé en entrée d'interruption.

EEDATA (08h) : Permet l'accès aux données dans la mémoire EEPROM.

EEADR (09h) : Permet l'accès aux adresses de la mémoire EEPROM.

PCLATCH (0Ah - 8Ah) : Donne accès en écriture aux bits de poids forts du compteur de programme.

INTCON (0Bh - 8Bh) : Masque d'interruptions.

OPTION_REG (81h) : Contient des bits de configuration pour divers périphériques.

TRISA (85h) : Indique la direction (entrée ou sortie) du port A.

TRISB (86h) : Indique la direction (entrée ou sortie) du port B.

EECON1 (88h) : Permet le contrôle d'accès à la mémoire EEPROM.
EECON2 (89h) : Permet le contrôle d'accès à la mémoire EEPROM.
(Fin cours séance 1)

II.3.1 Registres PORTA/PORTB

a) Registre PORT A (05h)

Ce registre est de 5 bits, il permet de communiquer directement avec l'extérieur par l'intermédiaire des 5 broches du port A.

Broches	Nom	Fonction	Sens
1	RA2	Port A (E/S)	Bidirectionnel
2	RA3	Port A (E/S)	Bidirectionnel
3	RA4	Port A (E/S)	Bidirectionnel
17	RA0	Port B (E/S)	Bidirectionnel
18	RA1	Port B (E/S)	Bidirectionnel

Ces bits fonctionnent en lecture et écriture (R/W) et elles peuvent accepter ou générer 0V (o logique) ou 5 V (1 logique).

Le registre **TRISA** est de 8 bits et d'adresse 85h dans la banque 1, permet de choisir le sens de chaque patte (entrée ou sortie) : un bit à 1 positionne le port en entrée, un bit à 0 positionne le port en sortie. La patte RA4 peut aussi servir d'entrée de comptage pour le timer0.

Remarque :

1- Il n'y a pas d'instruction permettant d'écrire directement dans TRISA, on écrit d'abord dans l'accumulateur, puis on le transfère dans TRISA. Par défaut, ces ports sont configurés en entrée.

2- Chaque sortie peut fournir 20 mA maximum

3- Chaque entrée peut fournir 25 mA maximum

4- Débit total du port A ne peut pas dépasser 80 mA

b) Registre PORT B (06h)

Ce registre est de 8 bits il fonctionne en lecture et écriture (R/W), il permet de communiquer directement avec l'extérieur par l'intermédiaire des 8 broches du port B et Ces broches peuvent accepter ou générer 0V (o logique) ou 5 V (1 logique).

Broches	**Nom**	**Fonction**	**Sens**
6	RB0	Port B (E/S)	Bidirectionnel
7	RB1	Port B (E/S)	Bidirectionnel
8	RB2	Port B (E/S)	Bidirectionnel
9	RB3	Port B (E/S)	Bidirectionnel
10	RB4	Port B (E/S)	Bidirectionnel + interruption par changement d'état
11	RB5	Port B (E/S)	Bidirectionnel + interruption par changement d'état
12	RB6	Port B (E/S)	Bidirectionnel + interruption par changement d'état
13	RB7	Port B (E/S)	Bidirectionnel + interruption par changement d'état

Il est noter que les broches RBO et RB4 à RB7 doivent être configurées en **entrées** dans le cas ou elles sont utilisées pour détecter des interruptions externes.
Le registre **TRISB** est de 8 bits et d'adresse 86h dans la banque 1, permet de choisir le sens de chaque patte (entrée ou sortie) : un bit à 1 positionne le port en entrée, un bit à 0 positionne le port en sortie.

Remarque :

1- Toutes les entrées du Port B peuvent être connectées à des résistances pour avoir en entrée un 1 logique.

2- Il n'y a pas d'instruction permettant d'écrire directement dans TRISB, on écrit d'abord dans l'accumulateur, puis on le transfère dans TRISB. Par défaut, ces ports sont configurés en entrée.

3- Chaque sortie peut fournir 20 mA maximum

4- Chaque entrée peut fournir 25 mA maximum

5- Débit total du port A ne peut pas dépasser 150 mA

II.3.2 Registre TIMER0 (01h)

Le PIC 16F84 est doté d'un compteur 8 bits. La Figure II.6 présente l'organigramme du TIMER0.

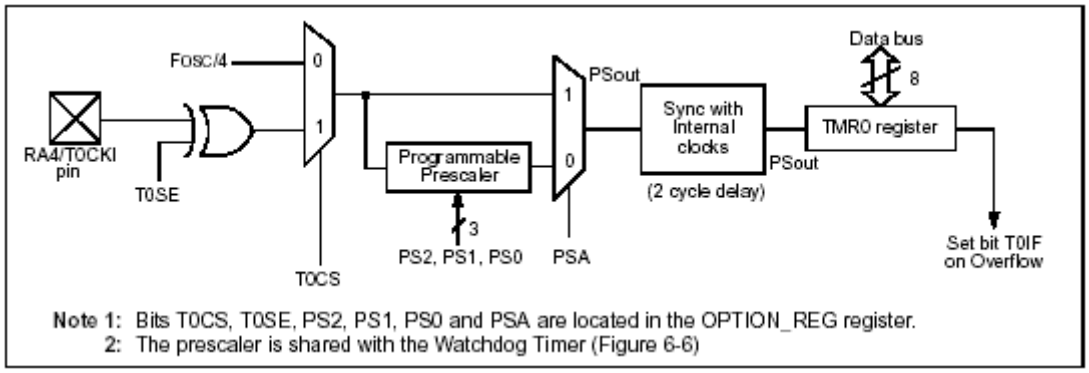


Figure II.6 : Organigramme du Timer0

Le compteur du Timer est un registre de 8 bits, il compte donc sur 8 bits. Tous les bits fonctionnent en R/W. il permet de réaliser plusieurs applications (temporisation, Compteur d'impulsion, fréquencemètre, ou générateur de fonctions) avec une source interne Timer mode et externe (Counter mode).

a) Timer mode (T0CS=0)

Dans ce mode, le mode s'incrémente par l'horloge interne Fosc/4 , à chaque cycle d'instruction. On peut le positionner à une valeur choisie en écrivant dans le registre TMR0. Si on écrit dans le registre TMR0, l'incrément est bloqué pendant les **deux cycles qui suivent** (Exemple temporisation : boucle).
Il peut être multiplié éventuellement par le taux de division du **pré-diviseur** (diviseur de fréquence)

b) Counter mode (T0CS=1)

Dans ce mode**,** le compteur s'incrémente à chaque impulsion appliquée sur RA4/TOCKI avec un signal symétrique, la fréquence du signal appliqué sur RA4/TOCKI doit être inférieur à ¼

de la fréquence horloge interne. Il peut être multiplié éventuellement par le taux de division du **pré-diviseur** (exemple fréquencemètre)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
01h	TMR0	Timer0 module's register								xxxx xxxx	uuuu uuuu
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 0000
81h	OPTION_ REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not associated with Timer0.

Figure II.7 : Registres utiles à la gestion de timer0.

c) Pré-diviseur (Prescaler)

Ce paramètre est utilisé pour réaliser une division supplémentaire de la fréquence d'horloge. En effet, la période de l'horloge d'entrée est divisée par une valeur comprise entre 2 et 256 suivant les bits PS2, PS1 et PS0 (respectivement OPTION_REG.2, .1 et .0) (Figure II.9). Le bit PSA (OPTION_REG.3) permet de choisir entre la pré-division de timer0 (PSA=0) ou du chien de garde (Watchdog) (PSA=1).

PSA	PS2	PS1	PS0	/tmr0	/WD
0	0	0	0	2	1
0	0	0	1	4	1
0	0	1	0	8	1
0	0	1	1	16	1
0	1	0	0	32	1
0	1	0	1	64	1
0	1	1	0	128	1
0	1	1	1	256	1
1	0	0	0	1	1
1	0	0	1	1	2
1	0	1	0	1	4
1	0	1	1	1	8
1	1	0	0	1	16
1	1	0	1	1	32
1	1	1	0	1	64
1	1	1	1	1	128

Figure II.8 : Valeurs du pré-diviseur en fonction de PSA, PS2, PS2 et PS0.

d) Fin de comptage et interruption

Le bit T0IF (INTCON.2) est mis à 1 chaque fois que le registre TMR0 passe de FFh à 00h. On peut donc tester ce bit pour connaître la fin de comptage. Pour compter 50 événements, il faut donc charger TMR0 avec la valeur 256-50=206 et attendre le passage de T0IF à 1. Cette méthode est simple mais bloque le processeur dans une boucle d'attente.

On peut aussi repérer la fin du comptage grâce à l'interruption que peut générer T0IF en passant à 1. Le processeur est ainsi libre de travailler en attendant cet événement.
fin du cours du 23/08/2018

e) Chien de garde (Watchdog)

Principe

C'est un système de protection contre un blocage du programme. Par exemple, si le programme attend le résultat d'un système extérieur (conversion analogique numérique par exemple) et qu'il n'y a pas de réponse, il peut rester bloquer. Pour en sortir on utilise un chien de garde. Il s'agit d'un compteur qui, lorsqu'il arrive en fin de comptage, permet de redémarrer le programme. Il est lancé au début du programme. En fonctionnement normal, il est remis à zéro régulièrement dans une branche du programme qui s'exécute régulièrement. Si le programme est bloqué, il ne passe plus dans la branche de remise à zéro et le comptage va jusqu'au bout, déclenche le chien de garde qui relance le programme.

Mise en service

Elle se décide lors de la programmation physique du PIC. Elle ne peut pas être suspendue pendant l'exécution d'un programme. Elle est définitive jusqu'à une nouvelle programmation de la puce.

La directive de programmation _CONFIG permet de valider (option _WDT_ON) ou non (option _WDT_OFF). La mise en service peut aussi être réalisée directement par le programmateur. L'inconvénient de cette seconde solution est que le code du programme ne contient pas l'information ; la mise en service du chien de grade peut être oubliée lors du téléchargement et générer un fonctionnement incorrect du programme en cas de blocage.

Gestion

Une fois le chien de garde mis en service, il faut remettre le comptage à zéro régulièrement. Cette opération est réalisée par l'instruction clrwdt. Tant que le programme se déroule normalement, cette instruction est exécutée régulièrement et le chien de garde ne s'active pas. Si un blocage apparaît, la remise à zéro n'a pas lieu et le chien de garde est activé. Le PIC redémarre alors à l'adresse 0000h et le bit TO (STATUS.4) est mis 0. Le test de ce bit au début du programme permet de savoir si le système vient d'être mis sous tension (TO=1) ou si le chien de garde vient de s'activer (TO=0).

Choix de la durée

Le chien de garde possède sa propre horloge. Sa période de base est de 18ms. Le pré-diviseur de fréquence utilisé par le compteur est partagé avec le chien de garde (Figure** VII.3). Si le bit PSA (OPTION_REG.3) est à 1, le pré-diviseur est assigné au chien de garde. 8 valeurs de 1 à 128 sont disponibles, ce qui permet d'aller jusqu'à 128*18ms=2.3s avant le déclenchement du chien de garde.

f) Mode sommeil (Sleep mode)

Principe

Lorsque le PIC n'a rien à faire (par exemple lors de l'attente d'une mesure extérieure), ce mode est utilisé pour limiter sa consommation : le PIC est mis en sommeil (le programme s'arrête) jusqu'à son réveil (le programme repart). Ce mode est principalement utilisé pour les systèmes embarqués fonctionnant sur pile.

Gestion

Mise en sommeil

La mise en sommeil est réalisée grâce à l'instruction sleep. La séquence suivante est exécutée :

- Le chien de garde est remis à 0 (équivalent à clrwdt)
- Le bit TO (STATUS.4) est mis à 1
- Le bit PD (STATUS.3) est mis à 0
- L'oscillateur est arrêté ; le PIC n'exécute plus d'instruction

Réveil

Ce mode n'est intéressant que si l'on peut en sortir pour relancer le programme. Trois événements permettent de sortir le PIC du sommeil :

- Application d'un niveau 0 sur l'entrée MCLR (broche numéro 4). Le PIC effectue alors un reset et relance le programme à partir de l'adresse 0000h. Les bits TO (STATUS.4) et PD (STATUS.3) permettent à l'utilisateur de savoir quel événement à lancer le programme (mise sous tension, reset, chien de garde).

- Activation du chien de garde. Le programme reprend à l'instruction suivant le sleep.

- Apparition d'une interruption (RB0/INT, RB ou EEPROM). Il faut pour cela que les bits de validation spécifique des interruptions concernées soient positionnés. Si le bit de validation générale des interruptions (GIE) est à 0 (pas de validation des interruptions), le programme reprend après l'instruction sleep comme pour le chien de garde. Si le bit de validation générale des interruptions (GIE) est à 1 (validation des interruptions), l'instruction suivant le sleep est exécutée et la fonction d'interruption liée à l'événement qui a réveillé le PIC est exécutée.

II.3.3 Registre OPTION (81 h)

C'est un registre de 8 bits, il permet de fixer des paramètres pour contrôler :

- ❖ Le Timer/Compteur
- ❖ Le pré-diviseur
- ❖ L'interruption externe INT
- ❖ Le tirage au plus du port B (Weak Pull UP)
- ❖ Tous les bits fonctionnent en R/W

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
bit 7							bit 0

Bit 7 : NOT_RBPU : Registre B Pull Up: Autorise ou non le tirage au + du port B par des résistances
Bit=1 : il n'y a pas de tirage au +
Bit=0 : Autorise le tirage au +

Bit 6 : INTEDG : interrompe Edge : il permet de choisir le front actif de l'interruption externe sur la broche 6 RB0/INT
Bit=1 : front montant est actif
Bit=0 : front descendant est actif

Bit 5 : TOCS : Timer Clock Source : Permet de choisir du signal d'horloge qui incrémente le Timer
Bit=1 : le Timer s'incrémente à chaque impulsion appliqué sur RA4/T0CKI
Bit=0 : le Timer s'incrémente à chaque instruction (sans pré-diviseur)

Bit 4 : TOSE : Timer 0 Source Edge Select : Lorsque le Timer s'incrémente sur chaque impulsion appliqué sur RA4/T0CKI ; ce bit permet de choisir si cela se fait sur le front montant ou sur le front descendant
Bit=1 : le Timer module s'incrémente sur les fronts descendants de RA4/T0CKI
Bit=0 : le Timer module s'incrémente sur les fronts montants de RA4/T0CKI

Bit 3 : PSA : Pre scaler Assignement : Détermine si le pré-diviseur sera associé au Timer ou au Watchdog

Bit=1 : le pré-diviseur fonctionne sur watchdog
Bit=0 : le pré-diviseur fonctionne sur le Timer

Bit 2, 1, 0: PS Pre scaler rate Select : Ils déterminent le taux de division du prés-diviseur (voir figure II.9)

II.3.4 Registre INTCON (0Bh ou 8Bh)

Contrôle des interruptions (Interrupt control) il gère les interruptions sa taille est de 8 bits.
Tous les bits fonctionnent en R/W

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	EEIE	T0IE	INTE	RBIE	TOIF	INTF	RBIF
bit 7							bit 0

Bit 7 : Global Interrupt Enable (GIE) : il autorise ou non tous types d'interruptions
Bit=1 : Toutes les interruptions sont autorisées
Bit=0 : les interruptions ne sont pas prises en compte par le PIC
Il passe à 0 pendant le traitement d'une interruption, il est remis à 1 par l'instruction de retour de la routine de traitement de l'interruption

Bit 6 : EEprom Interrupt Enable (EEIE) : Autorise ou non tous types les interruptions en fin d'écriture dans l'EEPROM
Bit=1 : Autorise l'interruption en fin d'écriture
Bit=0 : Interdit l'interruption en fin d'écriture

Bit 5 : Timer Overflow Interrupt Enable (T0IE) : Autorise ou non tous types les interruptions générés par un dépassement Timer (passage de FFh à 00h)
Bit = 1 : Autorise l'interruption lors du passage de FFh à 00h du Timer
Bit = 0 : Interdit l'interruption lors du passage de FFh à 00h du Timer

Bit 4 : Interrupt Enable (INTE) : Autorise ou non les interruptions par l'entrée RB0/INT broche 6
Bit = 1 : Autorise l'interruption par l'entrée RB0/INT broche 6
Bit = 0 : Interdit l'interruption par l'entrée RB0/INT broche 6

Bit 3 : Register B Interrupt Enable (RBIE) : Autorise ou non les interruptions par changement d'état sur le port B, entrées RB4 à RB7
Bit = 1 : Autorise l'interruption par changement d'état sur le port B
Bit = 0 : Interdit l'interruption par changement d'état sur le port B

Bit 2 : Timer 0 Overflow Interrupt Flag (T0IF) : fonctionne lorsque le bit 5 (**T0IE)** autorise les interruptions dues au dépassement Timer
Bit = 1 : Le compteur timer TMR0 a effectué un dépassement
Bit = 0 : Le compteur timer TMR0 n'a pas effectué le dépassement

Bit 1 : Interrupt Flag (INTF) : fonctionne lorsque le bit 4 (**INTE)** autorise les interruptions par l'entrée RB0/INT broche 6 et signale alors une interruption valide sur l'entrée RB0/INT broche 6
Bit = 1 : Une interruption est apparue sur RB0/INT broche 6
Bit = 0 : Pas d'interruption sur RB0/INT broche 6

Bit 0 : Register B Interrupt Flag (RBIF): fonctionne lorsque le bit 3 (**RBIE**) autorise les interruptions par changement d'état sur le port B et signale une interruption valide sur le port B.
Bit = 1 : l'un des ports RB4, RB5, RB6 ou RB7 a changé d'état
Bit = 0 : Aucun des ports RB4, RB5, RB6 ou RB7 n'a changé d'état
fin du cours 28/04/2017

II.3.5 Registre STATUS (03h ou 83h)

C'est un registre de 8 bits, il permet de d'indiquer une ensemble d'informations relatives aux :

- Opérations arithmétiques
- Choix de la page mémoire SRAM de données
- Débordement du chien de garde
- Exécution d'un sleep
- Tous les bits fonctionnent en R/W

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

Bit 7 : IRP : Sélectionne la page de registre utilisé. Ce bit n'est pas utilisé avec le PIC 16F84 et doit être maintenu à 0

Bit 6 : Register Page 1 (RP1) : Sélectionne la page de registre utilisé. Ce bit n'est pas utilisé avec le PIC 16F84 et doit être maintenu à 0

Bit 5 : Register Page 0 (RP0) : Sélectionne la page de registre utilisé.

RP0 = 1 : Sélectionne la page de registre de 80h à 8Bh (bsf STATUS, RP0)
RP0 = 0 : Sélectionne la page de registre de 00h à 0Bh (bcf STATUS, RP0)

Bit 4 : Not Time Out (NOT_TO) : Repère le dépassement de délai du chien de garde (watchdog)

NOT_TO = 1: Après l'exécution de Sleep
NOT_TO = 0: Après un Power UP ou une instruction CLRWDT

Bit 3 : Not Power Down (NOT_PD) : Fonctionne en lecture seulement

NOT_PD = 1: Après un Power On Reset ou une instruction CLRWDT
NOT_TO = 0: Après l'exécution de Sleep

Bit 2 : Zero (Z)
Z = 1 : si le résultat de l'opération arithmétique est 0
Z = 0 : si le résultat de l'opération arithmétique est différent de 0

Bit 1 : Digit Carry (DC) : Utilisé en BCD
DC = 1 : si le retenue d'un résultat est entre les bits 3 et 4
DC = 0 : si le retenue d'un résultat n'est pas entre les bits 3 et 4
Bit 0 : Carry (C)
C = 1 : s'il y a un retenu externe d'un résultat
C = 0 : s'il n'y a pas un retenu externe d'un résultat

II.3.6 Registre de Travail (Accumulateur W)

C'est un registre de 8 bits, il est impliqué pratiquement dans toutes les instructions. Il est utilisé avec UAL pour effectuer des opérations arithmétiques et logiques. Il est également associé aux opérations d'Entrées/Sorties, de transfert de données,…. Etc.

II.3.7 Registre d'instruction

La taille de ce registre est de 14 bits. Il permet de stocker temporairement, de décoder et d'exécuter l'instruction

II.4. Jeu d'instructions

Les PIC 16F84 est conçu selon une architecture RISC. Programmer avec 35 instructions permettant de limiter la taille du codage et donc de la place mémoire et du temps d'exécution.

II.4.1 Format général

Le format général d'une instruction est présenté dans la Figure II.9.

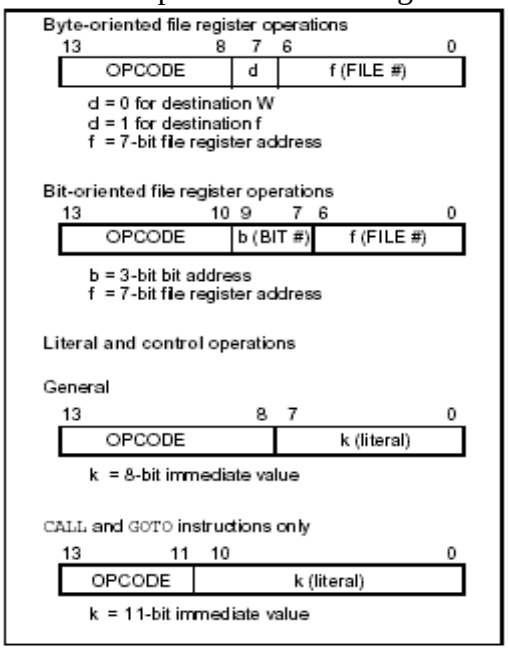


Figure II.9 : format général d'une instruction

Toutes les instructions sont codées sur 14 bits. Elles sont regroupées en trois grands types (Figure II.4) :

- Instructions orientées octets
- Instructions orientées bits
- Instructions de contrôle

Le registre de travail W joue un rôle particulier dans un grand nombre d'instructions.

II.4.2 Liste des instructions

INSTRUCTIONS OPERANT SUR REGISTRE (direct)			indicateurs	Cycles
ADDWF	**F,d**	W+F → {W,F ? d}	C,DC,Z	1
ANDWF	**F,d**	W and F → {W,F ? d}	Z	1
CLRF	**F**	Clear F	Z	1
CLRW		Clear W	Z	1
CLRWDT		Clear Watchdoc timer	TO', PD'	1
COMF	**F,d**	Complémente F → {W,F ? d}	Z	1
DECF	**F,d**	décrémente F → {W,F ? d}	Z	1
DECFSZ	**F,d**	décrémente F → {W,F ? d} skip if 0		1(2)
INCF	**F,d**	incrémente F → {W,F ? d}	Z	1
INCFSZ	**F,d**	incrémente F → {W,F ? d} skip if 0		1(2)
IORWF	**F,d**	W or F → {W,F ? d}	Z	1
MOVF	**F,d**	F → {W,F ? d}	Z	1
MOVWF	**F**	W → F		1
RLF	**F,d**	rotation à gauche de F a travers C → {W,F ? d}	C	1
RRF	**F,d**	rotation à droite de F a travers C → {W,F ? d}		1
SUBWF	**F,d**	F – W → {W,F ? d}	C,DC,Z	1
SWAPF	**F,d**	permute les 2 quartets de F → {W,F ? d}		1
XORWF	**F,d**	W xor F → {W,F ? d}	Z	1

INSTRUCTIONS OPERANT SUR BIT				
BCF	**F,b**	RAZ du bit b du registre F		1
BSF	**F,b**	RAU du bit b du registre F		1
BTFSC	**F,b**	teste le bit b de F, si 0 saute une instruction		1(2)
BTFSS	**F,b**	teste le bit b de F, si 1 saute une instruction		1(2)

INSTRUCTIONS OPERANT SUR DONNEE (Immediat)				
ADDLW	**K**	W + K → W	C,DC,Z	1
ANDLW	**K**	W and K → W	Z	1
IORLW	**K**	W or K → W	Z	1
MOVLW	**K**	K → W		1
SUBLW	**K**	K – W → W	C,DC,Z	1
XORLW	**K**	W xor K → W	Z	1

INSTRUCTIONS GENERALES				
CALL	**L**	Branchement à un sous programme de label L		2
GOTO	**L**	branchement à la ligne de label L		2
NOP		No operation		1
RETURN		retourne d'un sous programme		2
RETFIE		Retour d'interruption		2
RETLW	**K**	retourne d'un sous programme avec K dans W		2
SLEEP		se met en mode standby	TO', PD'	1

{W,F ? d} signifie que le résultat va soit dans W si d=0 ou w, soit dans F si d= 1 ou f

Figure II.10 : liste des 35 instructions du PIC 16F84

II.5. Modes d'adressages

On ne peut pas concevoir un programme qui ne manipule pas de données. Il existe trois grands types d'accès à une donnée ou modes d'adressage :

- Adressage immédiat : La donnée est contenue dans l'instruction.
- Adressage direct: La donnée est contenue dans un registre.
- Adressage indirect : L'adresse de la donnée est contenue dans un pointeur.

II.5.1 Adressage immédiat

La donnée est contenue dans l'instruction.

Exemple : movlw 0xC4 ; **Transfert la valeur 0xC4** dans W

II.5.2 Adressage direct

La donnée est contenue dans un registre. Ce dernier peut être par un nom (par exemple W) ou une adresse mémoire.

Exemple : movf 0x2B, 0 ; **Transfert dans W** la valeur **contenue à l'adresse 0x2B.**
Remarque : L'adresse 0x2B peut correspondre à 2 registres en fonction de la banque choisie (Figure II.11). Le bit RP0 permet ce choix, le bit RP1 étant réservé pour les futurs systèmes à 4 banques.

II.5.3 Adressage indirect

L'adresse de la donnée est contenue dans un pointeur. Dans les PIC, un seul pointeur est disponible pour l'adressage indirect : FSR. Contenu à l'adresse 04h dans les deux banques, il est donc accessible indépendamment du numéro de banque. En utilisant l'adressage direct, on peut écrire dans FSR l'adresse du registre à atteindre. FSR contenant 8 bits, on peut atteindre les deux banques du PIC 16F84. Pour les PIC contenant quatre banques, il faut positionner le bit IRP du registre d'état qui sert alors de 9ème bit d'adresse (Figure II.11).

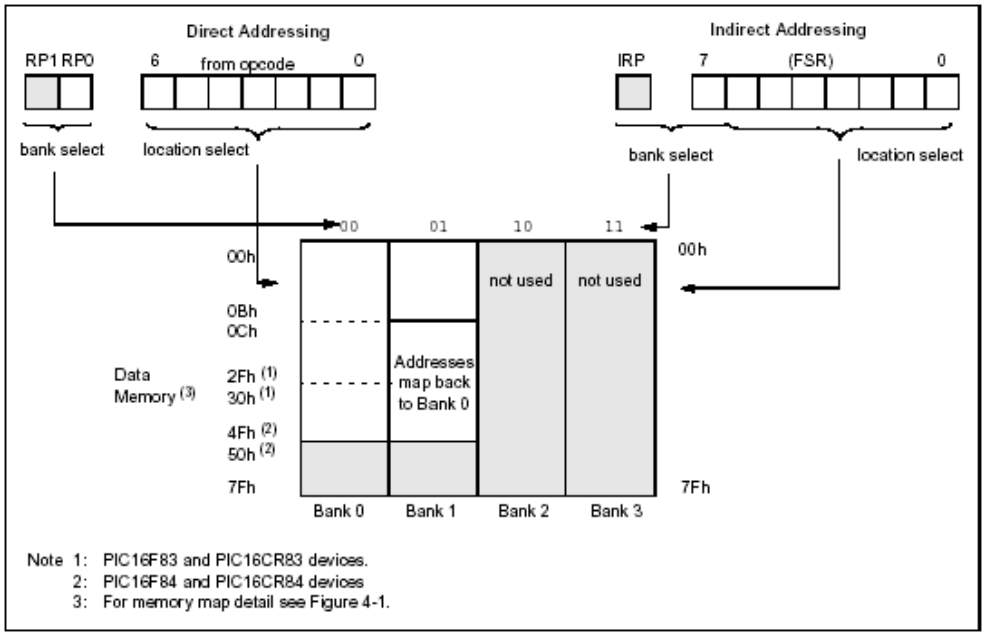


Figure II.11 : Adressages direct et indirect à la mémoire de données.

L'accès au registre d'adresse contenue dans FSR se fait en utilisant le registre INDF. Il se trouve à l'adresse 0 dans les deux banques. Il ne s'agit pas d'un registre physique. On peut le voir comme un autre nom de FSR, utilisé pour accéder à la donnée elle-même, FSR servant à choisir l'adresse.

Exemple : movlw 0x1A ; Charger 1Ah dans W
movwf FSR ; Charger W, contenant 1Ah, dans FSR
movf INDF, 0 ; Charger la valeur contenue à l'adresse 1Ah dans W
fin du cours 05/04/2018

II.6. Interruptions

L'interruption est un mécanisme fondamental de tout processeur. Il permet de prendre en compte des événements extérieurs au processeur et de leur associer un traitement spécifique. La Figure II.12 donne le déroulement du programme lors d'une interruption. Il faut noter que l'exécution d'une instruction n'est jamais interrompue ; c'est à la fin de l'instruction en cours lors de l'arrivée de l'événement que le sous-programme d'interruption est exécuté.

La séquence classique de fonctionnement d'une interruption est la suivante :

1- Détection de l'événement déclencheur
2- Fin de l'instruction en cours
3- Sauvegarde de l'adresse de retour
4- Déroutement vers la routine d'interruption
5- Sauvegarde du contexte
6- Identification de l'événement survenu
7- Traitement de l'interruption correspondante
8- Restauration du contexte
9- Retour au programme initial

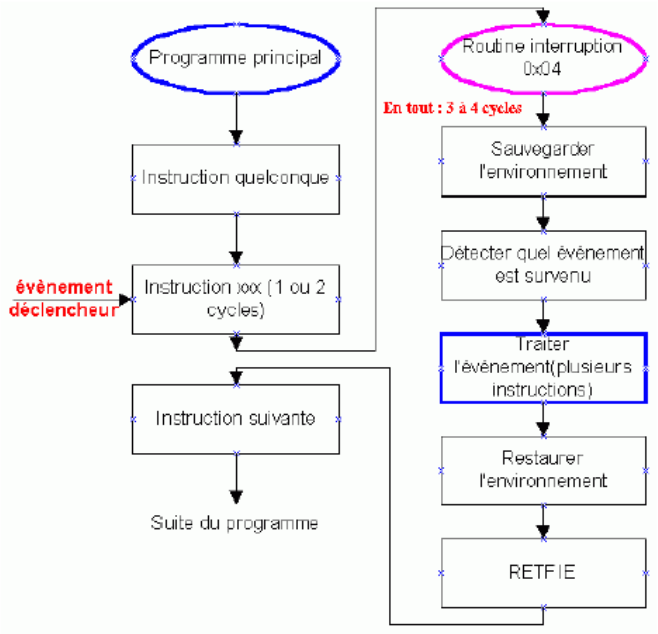


Figure II.12 : Déroulement d'un programme lors d'une interruption.

II.6.1 Différentes sources d'interruption

Dans le cas du PIC 16F84, il existe 4 sources d'interruption :

- INT : Interruption externe, broche RB0/INT
- TMR0 : Fin de comptage
- PORTB : Changement d'état du port B (RB7-RB4)
- EEPROM : Fin d'écriture en EEPROM

II.6.2 Séquence de détournement vers le sous-programme d'interruption

Par construction, l'interruption survient n'importe quand pendant l'exécution du programme. Avant l'exécution du sous-programme d'interruption, il faut donc sauvegarder l'adresse de l'instruction suivant celle en cours pour l'exécuter après le sous-programme d'interruption. L'adresse de retour est stockée dans la pile. Cette opération est gérée automatiquement par le processeur.

Une fois l'adresse de retour sauvegardée, le compteur de programme peut être chargé avec l'adresse du sous-programme à exécuter, ici 0004h.

Dans le cas du PIC, à cause de la faible taille de la pile, une interruption n'est pas interruptible. Le bit GIE de validation générale est donc mis à 0 au début du sous-programme d'interruption. Cette opération est gérée automatiquement par le processeur.

La Figure II.13 montre l'enchaînement des ces opérations. Cinq étapes sont alors utiles pour commencer l'interruption :

- Apparition d'un événement, sans perturber le déroulement normal des instructions
- Prise en compte de l'événement, exécution de l'instruction en cours (PC)
- Cycle d'attente, sauvegarde de l'adresse PC+1 dans la pile
- Chargement de l'adresse 0004h dans le PC
- Exécution de l'instruction d'adresse 0004h et chargement de l'instruction suivante

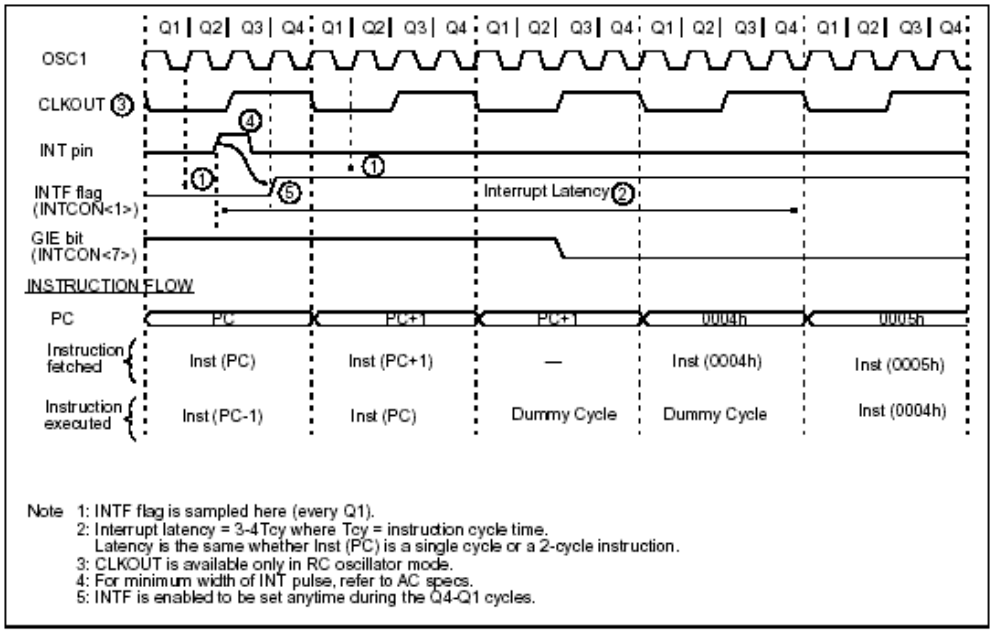


Figure II.13 : Déroulement de l'appel à un sous-programme d'interruption.

Deux cycles machine sont donc perdu à chaque interruption, sans compter la sauvegarde et la restitution du contexte et le retour au programme initial.

II.6.3 Sauvegarde et restitution du contexte

C'est un point important pour tous les sous-programmes qui devient capital pour les sous-programmes d'interruption. En effet, beaucoup d'instructions modifient le registre STATUS et/ou utilisent le registre W. Afin de les rendre dans le même état à la fin du sous-programme d'interruption qu'au début, il faut les sauvegarder au début et les recopier à la fin. Si d'autres

a) Où sauvegarder ces registres ?

Classiquement dans la pile. Dans le cas des PICs, elle est très petite et non accessible pour l'utilisateur. Il faut donc définir une zone de sauvegarde dans la RAM. Pour définir une variable, on peut utiliser les directives CBLOCK et ENDC :

```
CBLOCK 0x0C ; début de la zone de stockage
    Sauve_W : 1 ; 1 octet réservé pour la sauvegarde de W
```

```
        Sauve_Status :1 ; 1 octet réservé pour la sauvegarde de STAUTS
    ENDC
```

b) Comment sauvegarder ces registres ?

Pour W, c'est simple : il suffit d'utiliser l'instruction :

```
movwf Sauve_W ; Sauvegarde de W
```

Pour STATUS, c'est plus compliqué. En effet, il n'existe pas d'instruction de transfert d'un registre vers un autre. Il faut donc passer par W. Première difficulté : il faut penser à sauvegarder W avant de l'utiliser pour sauvegarder STATUS. On pourrait alors utiliser la séquence suivante :

```
movf STATUS, 0 ; Ecrit la valeur de STATUS dans W
movwf Sauve_STATUS; ; Sauvegarde de STATUS
```

Malheureusement, cette séquence ne fonctionne pas. En effet, l'instruction movf modifie le bit Z du registre STATUS. L'astuce consiste à utiliser l'instruction swapf qui intervertit les digits de poids fort et de poids faible d'un registre, sans modifier le registre STATUS. La séquence suivante permet de sauvegarder STATUS "swapé" :

```
swapf STATUS, 0 ; Ecrit STATUS "swapé" dans W
movwf Sauve_STATUS ; Sauvegarde de STATUS "swapé"
```

c) Comment restituer ces registres ?

Il faut commencer par restituer STATUS sans le modifier. En effet, on doit pour cela utiliser W qu'il est donc inutile de restituer avant. Comme STATUS a été sauvegardé "swapé", la séquence suivante convient :

```
swapf Sauve_STATUS, 0 ; Ecrit Sauve_Status "sawpé" dans W
movwf STATUS ; Restitue STATUS, "swapé" deux fois
```

Pour restituer W, on pourrait tout simplement utiliser la séquence suivante :

```
movf Sauve_W, 0 ; Ecrit Sauve_W dans W
```

Malheureusement, l'instrction movf modifie le bit Z du registre STATUS, déjà restitué. Il faut donc encore une fois passer par l'instruction swap, à exécuter deux fois :

```
swapf Sauve_W, 1 ; Ecrit Sauve_W "swapé" dans lui-même
```

swapf Sauve_W, 0 ; Restitue W, "swapé" deux fois.

II.6.4 Reconnaissance de l'interruption active

En revanche, il n'existe qu'une adresse d'interruption, 0004h, pour les différentes sources. Les bits 0 à 2 du registre INTCON (Figure* IX.3) et le bit 4 du registre EECON1 (Figure* VIII.1) permettent de savoir quel événement extérieur a déclenché une interruption. Ainsi, au début du programme d'interruption, si plusieurs sources ont été validées, il faut impérativement aller tester ces différents bits pour connaître la source active et dérouler le programme correspondant. On utilise pour cela l'instruction btfsc qui exécute l'instruction suivante si le bit testé vaut 1, la saute sinon. On peut donc écrire la séquence suivante après la sauvegarde du contexte où Int_xxx correspond aux différents sous-programmes de gestion des divers événements :

```
btfsc INTCON, 0 ; Test du bit RBIF
call Int_PB ; Appel sous-programme si RBIF=1
btfsc INTCON, 1 ; Test du bit INTF
call Int_Ext ; Appel sous-programme si INTF=1
btfsc INTCON, 2 ; Test de bit T0IF
call Int_Timer ; Appel sous-programme si T0IF=1
btfsc EECON1, 4 ; Test de bit EEIF
```

call Int_Timer ; Appel sous-programme si EEIF=1

II.6.5 Retour au programme initial

Une fois le sous-programme d'interruption terminé, après la restitution du contexte, il faut revenir au programme initial. C'est l'instruction retfie qui le permet. Elle commence par revalider les interruptions (GIE=1) puis elle revient au programme initial grâce à la valeur du compteur de programme empilée.

II.7 Accès à la mémoire EEPORM

Le PIC possède une zone EEPROM de 64 octets accessibles en lecture et en écriture par le programme. On peut y sauvegarder des valeurs, qui seront conservées même si l'alimentation est éteinte, et les récupérer lors de la mise sous tension. Leur accès est spécifique et requiert l'utilisation de registres dédiés. La lecture et l'écriture ne peuvent s'exécuter que selon des séquences particulières.

II.7.1 Registres utilisés

Quatre registres sont utilisés pour l'accès à la mémoire eeprom du PIC :

- EEDATA contient la donnée.
- EEADR contient l'adresse.
- EECON1 (Figure II.14) est le registre de contrôle de l'accès à l'eeprom. Cinq bits permettent un cet accès :
  - RD et WR initient la lecture ou l'écriture. Ils sont mis à 1 par le programme pour initier l'accès et mis à zéro par le système à la fin de l'accès.
  - WREN autorise (1) ou non (0) l'accès en écriture.
  - WRERR est mis à 1 par le système quand une opération d'écriture est interrompue par MCLR, reset ou le chien de garde.

- EEIF est un drapeau d'interruption signalant la fin de l'écriture physique dans la mémoire eeprom. Il doit être mis à 0 par programme.

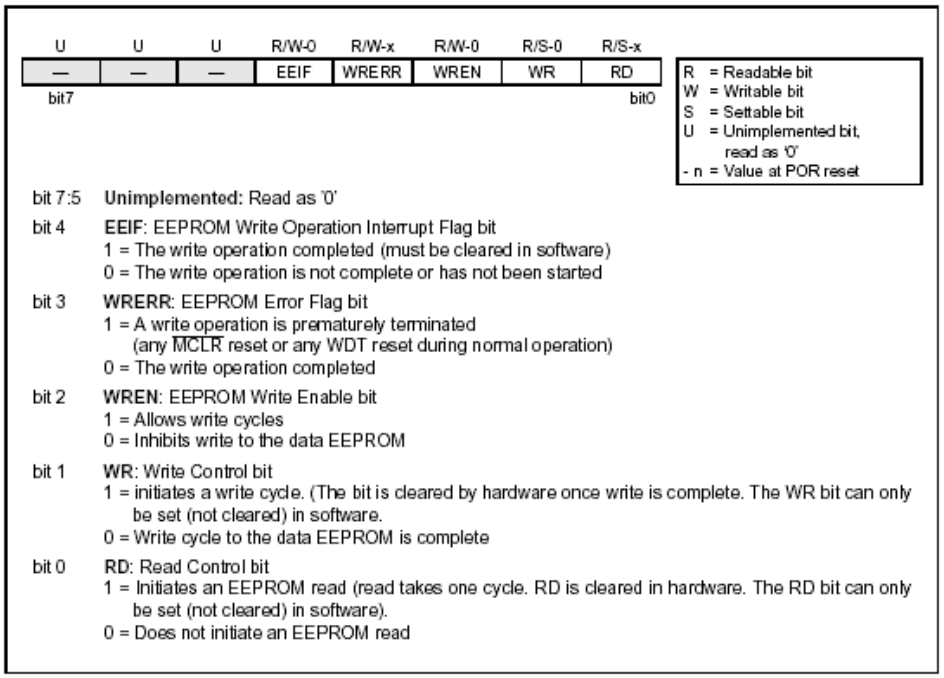


Figure II.14 : Registre EECON1.

- EECON2 joue un rôle spécifique lors de l'écriture.

II.7.2 Lecture

Pour lire une donnée dans la mémoire eeprom, il faut mettre l'adresse dans EEADR et positionner RD à 1. La valeur lue est alors disponible dans EEDATA au cycle machine suivant. Le programme ci-dessous donne un exemple de lecture dans la mémoire eeprom.

```
BCF     STATUS, RP0  ; Bank 0
MOVLW   CONFIG_ADDR  ;
MOVWF   EEADR        ; Address to read
BSF     STATUS, RP0  ; Bank 1
BSF     EECON1, RD   ; EE Read
BCF     STATUS, RP0  ; Bank 0
MOVF    EEDATA, W    ; W = EEDATA
```

II.7.3 Ecriture

Pour écrire une donnée dans la mémoire eeprom, il faut d'abord mettre l'adresse dans EEADR et la donnée dans EEDATA. Un cycle bien spécifique doit ensuite être respecté pour que l'écriture ait lieu. L'exemple suivant donne le cycle :

```
        BSF     STATUS, RP0  ; Bank 1
        BCF     INTCON, GIE  ; Disable INTs.
        BSF     EECON1, WREN ; Enable Write
        MOVLW   55h          ;
Required
Sequence
        MOVWF   EECON2       ; Write 55h
        MOVLW   AAh          ;
        MOVWF   EECON2       ; Write AAh
        BSF     EECON1,WR    ; Set WR bit
                             ;   begin write
        BSF     INTCON, GIE  ; Enable INTs.
```

II.8 Origine de Reset

Le reset représente une interruption spéciale qui initialise le PIC. Son vecteur d'interruption est stocké dans la case mémoire 0000H. Plusieurs façons sont possibles pour générer un reset.

- Mise à zéro de la broche 4 : MCLR qui assure une initialisation générale du PIC en mode normale ou en mode sleep
- Mise sous tension du PIC : Power On reset (POR) qui assure également une initialisation du PIC
- Débordement du timer du Watchdog
  - En mode normale : Initialisation du PIC
  - En mode sleep : réveil du PIC et incrémentation du PIC, le programme continue à l'instruction suivante.

Le tableau suivant résume les différentes possibilités du reset et les bits Not_TO et Not_PD du registre STATUS correspondants :

Bit 4 Not_TO	Bit 3 Not_PD	Compteur d'instruction PC	Etat du PIC	Origine
1	1	0000H		Power on Reset : POR
0	1	0000H	Normal	Watchdog
0	0	PC = PC +1	Sleep	
1	1	0000H	Normal	MCLR
1	0	0000H	Sleep	

III- Les outils de développement

Les étapes nécessaires permettant de voir un programme s'exécuter sur un PIC sont :

- Ecrire un programme en langage assembleur dans un fichier texte et le sauvegarder avec l'extension **.asm** (la syntaxe de ce langage se trouve en Annexe)
- Compiler ce programme avec l'assembleur MPASM fourni par Microchip. Le résultat est un fichier avec l'extension **.hex** contenant une suite d'instruction compréhensible par le pic.
- Transplanter le fichier **.hex** dans la mémoire programme du PIC (mémoire flash) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de Microchip ou tout autre programmateur acheté ou réalisé par soit même.
- Mettre le PIC dans son montage final, mettre sous tension et admirer le travail.

Microchip propose gratuitement l'outil de développement **MPLAB** qui regroupe l'éditeur de texte (voir Annexe), le compilateur MPASM, un outil de simulation et le logiciel de programmation.
Pour ce qui nous concerne, nous utiliseront MPLAB pour écrire, compiler et éventuellement simuler nos programmes, ensuite nous utiliserons un programmateur pour implanter les programmes dans la mémoire flash du PIC.

Remarque :
On ce qui concerne la simulation des circuits électroniques utilisant des PIC, nous utiliserons le logiciel ISIS Proteus qui est très réputé pour les simulations interactives à base des PICs et microprocesseurs.

Annexe

3- MPLAB

MPLAB est une plate-forme contenant :

- un éditeur
- un assembleur
- un simulateur

Ces outils permettent:

- L'édition du fichier source en langage assembleur (Nom_fichier.ASM)
- La traduction de ce fichier en un fichier.HEX (Nom_fichier.HEX) qui peut être chargé dans la mémoire programme du microcontrôleur (EEPROM)
- L'ensemble des fichiers nécessaires à ces opérations est regroupé dans un espace " projet " nommé par MPLAB : Nom_fichier.PJT

Tous les fichiers d'un même projet doivent porter exactement le même nom que le projet lui même, seules diffèrent les extensions

Ecriture du programme source

✓ Cliquer sur File - New

- Saisir le programme
- Cliquer sur File - Save As
- Indiquer le dossier où devra se sauvegarder le fichier
- le nom du fichier, le même que celui du projet (TP1.asm)
- Cliquer sur OK

Création d'un nouveau projet

✓ Cliquer sur Project - New Project

o Dans la fenêtre " New Project " qui s'ouvre, indiquer

- le dossier où devra se sauvegarder le projet
- le nom du projet (TP1.pjt par exemple)
- Cliquer sur *OK*

o La fenêtre " Edit Project " s'ouvre, elle confirme

- le nom du fichier objet qui sera créé (tp1.hex)
- l'éditeur concerné (Editor 16F84)
- l'environnement (Microchip)

o Mettre le nom du fichier (TP1.hex) en surbrillance, cliquer sur Add Node

o Dans la fenêtre " Add Node ", indiquer

- le dossier du projet
- le nom du fichier (TP1.asm)
- Cliquer sur *OK*

o Puis cliquer sur OK de la fenêtre " Edit Project" revient

Création du programme objet

✓ Traduction du fichier .asm en fichier .hex exploitable par le programmateur de PIC

- Cliquer sur Project - Make Project
- La traduction de TP1.asm en TP1.hex commence
- Si erreurs (message " *build failed*")
- Si pas d'erreurs (message " Build completed succesfully ")

Correction des erreurs

- Revenir au fichier source (TP1.asm)
- Corriger les erreurs
- Recompiler en cliquant sur Project - Make Project

Simulation

✓ Configuration de MPLAB en mode simulation

- *Options - Development Mode*
- Cliquer sur MPLAB-SIM Simulator

✓ Visualisation des contenus registres

- Window – Special Functions Registers

✓ Visualisation du contenu de la mémoire de données (SRAM)

- Window – File Registers
- Modification du contenu d'une mémoire : Window - Modify

✓ Visualisation du contenu de la mémoire programme (EEPROM)

- Window – Program memory

✓ Lacement de la simulation

- L'instruction à exécuter est mise en surbrillance,
- Avancement de l'exécution du programme : touche F7 ou touche F8
- Debug-Run-Animate (touches Ctrl F9) provoque l'exécution complète du programme

Ouverture d'un projet existant

- Cliquer sur Project - Open Project
- Indiquer le dossier où se trouve le projet
- Indiquer le nom du projet (TP1.pjt par exemple)
- Cliquer sur *OK*

Ouverture d'un fichier existant

- Cliquer sur File - Open
- Indiquer le dossier où se trouve le fichier
- Indiquer le nom du fichier (TP1.asm par exemple)
- Cliquer sur OK

Fichiers créés

Fichier Source (.asm) : fichier source
Fichier liste (.lst) : contient le source et le code machine correspondant.
Fichier objet (.cod) : contient le travail d'assemblage.
Fichier hexadécimal (.hex) : contient le code assembleur à graver dans le microcontrôleur
Fichier des erreurs (.err) : contient, les erreurs éventuelles

2- Assembleur MPASM

2.1 Introduction

Comme tous les langages, MPASM est composé d'un ensemble de directives, d'opérateurs et d'un jeu d'instructions.

2.2 Directives

Les directives sont des commandes destinées à l'assembleur (MPASM) pour lui indiquer un ensemble d'informations telles que: l'origine du programme, les différentes assignations et définitions, bloc de données,....Parmi les directives les plus utilisées on peut citer :

- **Directive d'assignation**

Syntaxe :
Label **EQU** expression
Description: Permet d'affecter une valeur à un label

Exemple

STATUS **EQU** H'0003' : Assigne l'adresse 03H au label STATUS (registre d'état)

Masque **EQU** 0x04 : Assigne la valeur 04H au label masque

Dans le fichier source, il suffira d'écrire STATUS ou Masque, l'assembleur comprendra qu'il s'agit respectivement de 03H et de 04H.

- **Directive de définition**

Syntaxe :
#define <name> [<string>]
Description: Permet de remplacer un texte difficile à mémoriser par un texte plus simple

Exemple

#define diode-rouge PORTA , 1

Dans le fichier source, il suffira d'écrire diode-rouge, l'assembleur comprendra qu'il s'agit de la sortie (ou entrée) n° 1 du PORTA.

La différence entre EQU et # define est qu'on réserve la première pour remplacer des valeurs et la seconde pour remplacer des textes.

- **Directive d'envoie de liste**

Syntaxe :

list [<list_option>, ..., <list_option>]

Description : Envoie une liste de directives à l'assembleur.

Option	Par défaut	Description
b = nnn	8	Espaces de tabulation
c = nnn	132	Largeur des colonnes
f=<format>	INHX8M	Format du fichier .hex de sortie : INHX32, INHX8M, ou INHX8S
free	FIXED	Utilise free-format parser.
fixed	FIXED	Utilise fixed-format parser
mm = { ON|OFF}	On	Sort la memory map dans le fichier .lst
n = nnn	60 (décimal)	Nombre de lignes par page
p = <type>	**None**	**Type de PIC**
r = <radix>	hex	Base de numération par défaut : hex, dec, oct
st = {ON|OFF}	On	Sort la table des symboles dans le fichier .lst
t = {ON|OFF}	Off	Coupe les lignes du listing
w = {0|1|2}	0	Détermine le type de message d'erreur
x = {ON|OFF}	On	macro expansion on ou off

Exemple

list p = 16F84, f = INHX32, r = dec

Type	Syntaxe	Exemple
Decimal	D'<digits>'	D'100'
	.<digits> ou <digit>	.100 ou 100
Hexadecimal	H'<hex_digits>	' H'9f ou 9fH
	0x<hex_digits>	0x9f
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001' ou 00111001B
ASCII	A'<character>'	A'C '
	'<character>'	'C'

- **Directive d'inclusion**

Syntaxe :

 # include <include_file> ou

 # include "<include_file>"

Description: Afin d'éviter l'écriture d'une multitude de lignes d'assignation au début de chaque programme, on peut les regrouper dans un seul fichier et l'appeler par cette directive.

Exemple

#include "p16F84.inc" ou **#include** <p16F84.inc>

Le fichier p16F84 est un fichier standard contenant toutes les assignations pour faciliter la mémorisation des noms des différents registres.

```
Register Definitions
W               EQU     H'0000'
F               EQU     H'0001'
```

```
Register Files
INDF            EQU     H'0000'
TMR0            EQU     H'0001'
PCL             EQU     H'0002'
STATUS          EQU     H'0003'
FSR             EQU     H'0004'
PORTA           EQU     H'0005'
PORTB           EQU     H'0006'
EEDATA          EQU     H'0008'
EEADR           EQU     H'0009'
PCLATH          EQU     H'000A'
INTCON          EQU     H'000B'
OPTION_REG      EQU     H'0081'
TRISA           EQU     H'0085'
TRISB           EQU     H'0086'
EECON1          EQU     H'0088'
EECON2          EQU     H'0089'
```

```
STATUS Bits
IRP             EQU     H'0007'
RP1             EQU     H'0006'
RP0             EQU     H'0005'
NOT_TO          EQU     H'0004'
NOT_PD          EQU     H'0003'
Z               EQU     H'0002'
DC              EQU     H'0001'
C               EQU     H'0000'
```

```
INTCON Bits
GIE             EQU     H'0007'
EEIE            EQU     H'0006'
T0IE            EQU     H'0005'
INTE            EQU     H'0004'
RBIE            EQU     H'0003'
T0IF            EQU     H'0002'
INTF            EQU     H'0001'
RBIF            EQU     H'0000'
```

```
OPTION Bits
NOT_RBPU        EQU     H'0007'
INTEDG          EQU     H'0006'
T0CS            EQU     H'0005'
T0SE            EQU     H'0004'
PSA             EQU     H'0003'
PS2             EQU     H'0002'
PS1             EQU     H'0001'
PS0             EQU     H'0000'
```

```
EECON1 Bits
EEIF            EQU     H'0004'
WRERR           EQU     H'0003'
WREN            EQU     H'0002'
WR              EQU     H'0001'
RD              EQU     H'0000'
```

```
RAM Definition
    __MAXRAM H'CF'
    __BADRAM H'07', H'50'-H'7F', H'87'
```

```
Configuration Bits
_CP_ON          EQU     H'000F'
_CP_OFF         EQU     H'3FFF'
_PWRTE_ON       EQU     H'3FF7'
_PWRTE_OFF      EQU     H'3FFF'
_WDT_ON         EQU     H'3FFF'
_WDT_OFF        EQU     H'3FFB'
_LP_OSC         EQU     H'3FFC'
_XT_OSC         EQU     H'3FFD'
_HS_OSC         EQU     H'3FFE'
_RC_OSC         EQU     H'3FFF'
```

Il peut y avoir des includes dans des includes, avec une hiérarchie de 6 niveaux au maximum.

- **Directive de l'adresse d'origine du programme d'application**

Syntaxe :

[<label>] **org** <expr>

Description : Cette directive indique à l'assembleur à quelle adresse (expr) doivent commencer les instructions du programme.

Remarques :

- Après un reset ou une mise sous tension, le PIC démarre toujours à l'adresse 0x00. Le début du programme doit donc se situer là. Il faut donc utiliser les adresses 0000 0001H 0002H et 0003H pour informer le PIC où aller à l'allumage ou après un Reset

- L'adresse 0x04 est l'adresse utilisée par les interruptions.

Exemple

```
org 0x000          ; Adresse de départ après reset
goto  init         ; Adresse 0: saut l'étiquette init
```

- **Directive de configuration**

Syntaxe :

__config <expr>

Description : Elle permet de configurer les paramètres (case mémoire de 14 bits dont l adresse = 2007h de l'EEPROM programme) de fonctionnement du PIC au moment de sa programmation. Leurs définitions sont dans le fichier include.

Code Protect	Power Up Timer	Watchdog	Oscillateur
bit 4	bit 3	bit 2	bit 1, bit 0
0->Oui	0->Oui	0->Non	00->LP, 01->XT
1->Non, par défaut	1->Non, par défaut	1->Oui, par défaut	10->HS, 11->RC, par défaut

Les 5 bits sont à 1 par défaut.

Exemple

__CONFIG _CP_OFF & _WDT_ON & _PWRTE_ON & _HS_OSC

_CP_OFF	Code protection OFF
_PWRTE_ON	Timer reset sur power on en service
_WDT_ON	Watch-dog en service
_HS_OSC	Oscillateur quartz grande vitesse

on peut exprimer la configuration désirée en binaire sur 14 bits.

__CONFIG b11111000111

- **Directive de déclaration d'un block de variables**

Syntaxe :

cblock [<expr>]

<Nom de la variable> [:<Nombre d'octets >]

< Nom de la variable > [:<Nombre d'octets >]

etc.....

endc

Description : Elle permet de déclarer les variables du programme toute en définissant une zone en mémoire SRAM où seront stockées les variables de l'application.

- **Cblock** et **endc** délimitent les frontières en mémoire.
- <expr> indique l'adresse de début de la zone mémoire (dans notre cas expr = 0xC0)
- Capacité maximale = 68octets.

Exemple

```
cblock h'20'
V1: 1          ; V1 sera codée sur 1 octet
V2             ; V1 sera codée sur 1 octet par défaut
V3 :5          ; V3 sera codée sur 5 octets
V4             ; 4 sera codée sur 1 octet par défaut
endc
```

ou une déclaration abrégée

```
cblock h'20'
V1 :1, V2, V3 :5, V4
Endc
```

V1 sera stockée l'adresse 20h
V2 sera stockée l'adresse 21h
V3 sera stockée l'adresse 22h
V4 sera stockée l'adresse **27**h

- **Directive de déclaration de variables**

Syntaxe :

variable <label> = <expr> [...,<label>=<expr>]

Description : Crée le symbole *label* et lui associe l'adresse *expr*. L'adresse peut être modifiée par la suite par **Label = autre_expr**

Exemple

variable X = h'0a'
la valeur de X sera stockée dans la case mémoire h'0a'

- **Macro**

Syntaxe :

<label> **macro** [<arg>, ..., <arg>]
séquence d'instructions
endm
Description: Elle remplace une partie du code qui sera fréquemment utilisé dans l'application. L'appel se fait simplement par le nom de la macro.

Exemple

- Sans passage de paramètres

```
Page0 macro
bcf STATUS,RP0
endm
```

l'instruction bcf permet de mettre zéro un bit. Cette macro annule le bit RP0 du registre d'état pour passer la page 0.

- Avec passage de paramètres

```
Read macro v1, v2, v3
movlw v1
movlw v2
movlw v3
endm
```

Remarque:

- Une macro peut appeler une autre macro. Elle peut également s'auto appeler. Le nombre maximum d'imbrication = 16.
- Les variables locales d'une macro sont defnies par la directive **local** :

local <label>[,<label>...]

- Les variables déclarées dans le programme principales sont des variables globales.

- **Directive de fin du programme source**

Syntaxe :

END

Description : Elle doit être placée en fin de programme

- **Directive de la valeur maximale de la RAM**

Syntaxe :

__maxram <expr>

Description : Indique la valeur maximale d'une adresse RAM valide

Expr doit être supérieur ou égal à la plus grande valeur de la page 0.

- **Directive des cases mémoire invalides de la RAM**

Syntaxe :

__badram <expr>[-<expr>][, <expr>[-<expr>]]

Description : Indique les adresses ou les plages d'adresses RAM invalides

Expr doit être inférieur ou égal à la valeur donnée par __maxram

Il faur définir __maxram avant d'utiliser __badram.

Exemple

```
__MAXRAM H'CF'
__BADRAM H'07', H'50'-H'7F', H'87'
```

2.3 Opérateurs

Operateur		Exemple
$	Current/Return program counter	goto $ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags
high	Return high byte	movlw high CTR_Table
low	Return low byte	movlw low CTR_Table
upper	Return upper byte	movlw upper CTR_Table
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
|	Bitwise inclusive OR	flags = flags | ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
||	Logical OR	if (len == 512) || (b == c)
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
|=	Inclusive OR, set equal	flags |= ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment i ++	
--	Decrement i --	

2.4 Modes d'adressage

L'accès aux données peut se faire à l'aide de 3 modes d'adressage :

- *Mode d'adressage immédiat* : La donnée est immédiatement manipulée par l'instruction.

Mnemonique constante

- *Mode d'adressage direct* : Dans ce cas, on indique directement l'adresse de la donnée à traiter.

Mnemonique adresse, registre de travail

Ou

Mnemonique registre de travail , adresse

- *Mode d'adressage indirect* : L'accès à la donnée se fait indirectement en utilisant les registres suivants : FSR indiquant l'adresse de la case mémoire de la donnée à traiter (pointeur) et INDF contiendra le contenu de la case adressée.

2.5 Jeu d'instructions

Le jeu d'instructions du PIC 16F84 appartient à la famille RISC (Reduced Instruction Set Computer). Il est composé de 35 instructions réparties selon les types suivants : Instructions orientées octets, instructions orientées bits, les instructions générales et les instructions de sauts et de branchement.

La syntaxe d'une instruction de l'assembleur **MPASM** intégré dans **MPLAB version 5.31** que nous utiliserons dans ce cours, est représentée par une ligne (contenant au maximum 255 caractères)et peut être répartie en cinq colonnes dans l'ordre comme suit :

- **Etiquette (facultative)** : Les caractères utilisables sont lettres alphabétiques, chiffres, sous ligné et point d'interrogation. Leur longueur maximale est de 32 caractères
- Espace(s) ou tabulation(s), ou deux point « : »
- **Mnémonique** (en majuscules ou minuscules),
- Tabulation ou Espace(s)
- **Opérande ou la valeur**
- Virgule éventuelle de séparation
- **Bit de destination W ou F** ou éventuellement numéro du bit de 0 à 7 si nécessaire
- Espace(s) ou tabulation(s)
- point-virgule. (facultatif si pas de commentaire)
- **Commentaire** (facultative)

Exemple

```
Etq1 :  MOVLW  k    ; Stocke la valeur k dans le registre de travail W
```

2.5.1 Format d'une instruction orientée octet

Chaque instruction est codée sur 14 bits répartis comme suit :

b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
OPCODE						**d**			**Opérandes : f**				

- Le code opération (OPCODE) indique la nature de l'opération à effectuer, il est codé sur 6 bits (b8 à b13).
- Le bit d (destination) indique l'endroit où sera stocké le résultat de l'exécution de l'instruction, il est représenté par le bit d (b7).
  - Pour d = 0, la destination sera l'accumulateur W
  - Pour d = 1, la destination sera une case mémoire (***utilisé par défaut***)
- Les opérandes représentent les données à traiter, ce champ est nommé par Microchip registre fichier (file) qui indique un emplacement mémoire de la RAM (00h à 7Fh). Elles sont représentées par les bits b0 à b6.

2.5.1.1 Description des instructions orientées octet

a. Instructions de transfert de données

Syntaxe :

MOVF f , d

(f) ⇒ (d)

Description : (Move f). Elle permet de copier le contenu de la case mémoire d'un registre (f) de la RAM dans (d)

- si d= 0 le résultat va dans le registre W
- si d= 1 le résultat va dans le registre f
- Indicateurs du STATUS affectés : Z
- Nombre de cycle d'horloge : 1

Exemple

Movf 0Ah, 0 ; le contenu du registre de la RAM 0Ah est copié dans W

Syntaxe :

MOVLW k

(k) ⇒ (W)

k: (0 à 255)

Description : (Move Literal to W). Stocke la valeur k dans le registre de travail W

- Indicateurs du STATUS affectés : Z
- Nombre de cycle d'horloge : 1

Exemple

Movlw 01010101b ; le registre W est chargé par 55h

Syntaxe :

MOVWF f

(W) ⇒ (f)

Description : ((Move W to f). Elle permet de copier le contenu du registre de travail W dans le registre (f) de la RAM.

- Indicateurs du STATUS affectés : aucun
- Nombre de cycle d'horloge : 1

Exemple

Movwf 0Ah ; le contenu de W est copié dans le registre 0Ah de la RAM .

Syntaxe :

SWAPF f,d

(f bit 0 à bit 3) ⇒ (d bit 4 à bit 7)

(f bit 4 à bit 7) ⇒ (d bit 0 à bit 3)

Description : (Swap halves f): Échange de quartets entre le registre (f) et (d)

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés : C, DC, Z

b. Instructions de traitement de données

- **Addition**

Syntaxe :

ADDLW k

k + (W) ⇒ (W)

k: (0 à 255)

Description (Add literal to W) : Ajoute la valeur k (valeur immédiate) au contenu du registre de travail W et met le résultat dans W.

- Indicateurs du STATUS affectés : C, DC, Z
- Nombre de cycle d'horloge : 1

Exemple

Avant l'instruction W contient d'05'

addlw d'07' ; après l'instruction W contient h'12'

Syntaxe :

ADDWF f , d

(W) + (f) ⇒ (d)

f: (00 à 4F); d: (0 ou 1)

Description (Add W and f) : Ajoute le contenu du registre de travail W à celui du registre f, et stocke le résultat dans W si d=0 ou dans f si d=1

- Indicateurs du STATUS affectés : C, DC, Z
- Nombre de cycle d'horloge : 1

Exemple

```
#define    W      0          ; on affecte 0 à la variable W utilisée ci – après
Movlw      12                ; charger 12 dans W
Movwf      valeur1           ; valeur1 vaut maintenant 12
Movlw      30                ; charger 30 dans W
Addwf      valeur1, W        ; 12 + 30   stocké dans W
```

- **Soustraction**

Syntaxe :

SUBLW k

k - (W) ⇒ (W)

k: (0 à 255)

Description (Subtract W from literal) : Soustraction entre une valeur K (valeur immédiate) et le registre W (complément à 2).

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés : C, DC, Z

Exemple

```
SUBLW 0x01        ; exécute la soustraction 01 - contenu de W résultat dans W
```

Syntaxe :

SUBWF f,d

(f) - (W) ⇒ (d)

k: (0 à 255)

Description (Subtract W from literal) : Retranche le contenu de W au contenu du registre f (complément à 2), et stocke le résultat dans W si d=0 ou dans f si d=1.

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés : C, DC, Z

Exemple

Dans cet exemple on charge une valeur dans le registre W puis on la soustraie du contenu d'un registre temporaire, le résultat est dirigé vers le registre temporaire..

(complément à 2)

```
Reg_temp  equ  0Ah     ;0Ah correspond à l'adresse d' un registre temporaire
#define f 1            on affecte la valeur 1 à la variable f
Movlw 00001010b        ; 0Ah dans le registre W
Movwf Reg_temp         ;  W dans le registre temporaire
Movlw 00000001b        ;  01h dans le registre W
Subwf Reg_temp, f      ; on soustraie  de W le contenu du registre temporaire= 00001001
```

- **Remise à zéro**

Syntaxe :

CLRF registre

$0 \Rightarrow f$

Description: (Clear f): Mettre zéro la case mémoire spécifiée (registre)

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Z

Exemple

```
CLRF INTCON          ; interdit toutes les interruptions
```

Syntaxe :

CLRW

$0 \Rightarrow W$

Description: (Clear W): Mettre zéro le contenu de l'accumulateur W.

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Z

- **Incrémentation / décrémentation**

Syntaxe :

INCF f , d

(f) + 1 $\Rightarrow$ (d)

Description : (Increment f): Incrémente f et range le résultat dans d. f est l'emplacement mémoire d 'un registre. f: (00 à 4F); d: (0 ou 1)

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Z

Exemple

```
Reg_temp   equ   0Ah          ; OAh  l'adresse d' un registre temporaire
#define f      1              ; Affectation de  la valeur 1 à la variable f
Movlw 01010101b               ; 55h dans le registre W
Movwf  Reg_temp               ; W dans le registre temporaire
Incf Reg_temp, f              ; le contenu du registre temporaire est incrémenté = 56h
```

Syntaxe :

DECF f , d

(f) - 1 ⇒ (d)

Description : (Decrement f): Décrémente f et range le résultat dans d, f est l'emplacement mémoire d 'un registre. f: (00 à 4F); d: (0 ou 1)

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Z

Exemple

```
Reg_temp   equ   0Ah          ; 0Ah  l'adresse d'un registre temporaire
#define f      1              ; Affectation de  la valeur 1 à la variable f
Movlw 01010101b               ; 55h dans le registre W
Movwf  Reg_temp               ; W dans le registre temporaire
Decf Reg_temp, f              ; le contenu du registre temporaire est décrémenté = 54h
```

Syntaxe :

INCFSZ f , d

(f) - 1 ⇒ (d) ⇒ skip if 0

Description : (Increment f, skip if zero): Incrémente f et range le résultat dans d. f est l'emplacement mémoire d 'un registre. **f: (00 à 4F); d: (0 ou 1)**

Si le résultat = 0, on saute l'instruction qui suit INCFSZ pour exécuter celle qui vient après

Si le résultat est ≠ 0, on exécute l'instruction qui suit

- Nombre de cycle d'horloge : 1 ou 2
- Indicateurs du STATUS affectés: Aucun

Exemple

```
Reg_temp   equ   0Ah          ; OAh l'adresse d' un registre temporaire
#define f 1                   ; la valeur 1 dans la variable f
Movlw 01h                     ; 01h dans le registre W
Movwf  Reg_temp               ; W dans le registre temporaire
etq :
INCFSZ  Reg_temp  , f         ; Incrémente f et saute l 'instruction suivante si f = 0
goto    etq                   ; exécuté si le résultat ≠ 0
movlw 50h                     ; exécuté si le résultat=0
```

Syntaxe :

DECFSZ f , d

(f) - 1 ⇒ (d) ⇒ skip if 0

Description : (Decrement f, skip if zero): Décrémente f et range le résultat dans d, f est l'emplacement mémoire d 'un registre. f: (00 à 4F); d: (0 ou 1)

Si le résultat=0, on saute l'instruction qui suit DECFSZ pour exécuter celle qui vient après

Si le résultat est ≠ de 0, on exécute l'instruction qui suit

- Nombre de cycle d'horloge : 1 ou 2
- Indicateurs du STATUS affectés: Aucun

Exemple

```
Reg_temp   equ   0Ah          ; 0Ah l'adresse d' un registre temporaire
#define f 1                   ; la valeur 1 dans la variable f
Movlw 01h                     ; 01h dans le registre W
Movwf  Reg_temp               ; W dans le registre temporaire
etq :
DECFSZ  Reg_temp  , f         ; Décrémente f et saute l 'instruction suivante si f = 0
goto    etq                   ; exécuté si le résultat ≠ 0
movlw 50h                     ; exécuté si le résultat=0
```

- **Opérations logiques**

Syntaxe :

ANDWF f,d

W AND f ⇒ d

Description : (AND W and f) : ET logique bit à bit entre les contenus de W et de f, et range le résultat dans d. f est l'emplacement mémoire d 'un registre. f: (00 à 4F); d: (0 ou 1)

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Z

Exemple

```
#define f 1                    ; la valeur 1 dans la variable f
R_etat  equ  03h               ; on affecte 03 à la variable R_etat utilisée ci - après
Movlw 00000111b                ; on met le contenu 00000111 dans le registre W
Andwf  R_etat, f               ; on masque et on configure le registre d' état
```

ANDLW k	(AND literal and W)	k AND W ⇒ W
COMF f,d	(Complement f)	NOT f ⇒ d
IORLW k	(Incl. OR literal and W)	k OR. W ⇒ W
IORWF f,d	(Inclusive OR W and f)	W OR f ⇒ d
RLF f,d	(Rotate left f)	f(n) ⇒ dest(n+1), f(7) ⇒ C, C ⇒ dest(0)
RRF f,d	(Rotate right)	f f(n) ⇒ dest(n-1), f(0) ⇒ C, C ⇒ dest(7)
XORLW k	(Exclusive OR literal and W)	k XOR W ⇒ W
XORWF f,d	(Exclusive OR W and f)	W XOR f->d

- Nombre de cycle d'horloge : 1
- Indicateurs positionnés: STATUS C ou Z

2.5.2 Instructions « orientées bits »

Ce sont des instructions destinées à manipuler directement des bits. Elles sont codeés sur 14 bits comme suit :

b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
OPCODE				**b**			**Opérandes : f**						

- 4 bits pour OPCODE l'instruction (b10 à b13) ,
- 3 bits pour indiquer le numéro du bit à manipuler (bit 0 à 7): (b7 à b9)
- 7 bits pour indiquer l'opérande **f** : (b0 à b6)

2.5.2.1 Description des instructions orientées bits

Syntaxe :

BCF f , b

(0 ⇒ (b)

Description: (Bit clear f):: (Decrement f, skip if zero): Met à zéro le bit b du registre f

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Aucun

Exemple

- bcf STATUS , RP0 ; 0 dans le bit 5 RP0 du registre STATUS, page mémoire 0

BSF f , b	(Bit set f)	1 ⇒ f(b)
BTFSC f ,b	(Bit test, skip if clear)	skip if f(b) = 0
BTFSS f ,b	(Bit test, skip if set)	skip if f(b) = 1

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Aucun

2.5.3 Description des instructions d'appel et de retour

Syntaxe :

CALL k

PC + 1⇒ pile et (k) ⇒ PC

k: (étiquette)

Description: Call : Saut à l'adresse k où se trouve un sous programme ; exécution des instructions jusqu'à une instruction de retour. La pile ne contenant que 8 mots, 8 imbrications sont possibles au maximum.

- Nombre de cycle d'horloge : 2
- Indicateurs du STATUS affectés: Aucun

Exemple

Call sous_P

Syntaxe :

GOTO k

(k) ⇒ PC (9 bits)

k: (étiquette)

Description: Goto address k: Saut inconditionnel, le programme saute à l'adresse donnée par la valeur de k (ou par une étiquette).

- Nombre de cycle d'horloge : 2
- Indicateurs du STATUS affectés: Aucun

Exemple

Goto fin

-

-

-

fin :

Syntaxe :

RETURN

Pile ⇒ PC

Description: (Return from subroutine): Retour au programme principal à la fin d'exécution d' un sous programme lancé par une instruction CALL.

- Nombre de cycle d'horloge : 2
- Indicateurs du STATUS affectés: Aucun

Syntaxe :

RETLW c

Pile ⇒ PC

Description: (Return with literal in W): Retour d'un sous programme, chargement d ' une valeur de retour c dans le registre W.

- Nombre de cycle d'horloge : 2
- Indicateurs du STATUS affectés: Aucun

Exemple RETLW 0afh ; retour du sous programme et on charge 0afh dans W

Syntaxe :

RETFIE

Pile ⇒ PC

Description: (Return from Interrupt) : Retour d un sous programme d'interruption.

- Nombre de cycle d'horloge : 2
- Indicateurs du STATUS affectés: INTCON, GIE

2.5.4 Autres instructions

Syntaxe :

NOP

(PC) + 1 ⇒ (PC)

Description: (No operation): Aucune opération.

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: Aucun

Syntaxe :

CLRWDT

Description: (Clear watchdog timer): Réinitialise le temporisateur du chien de garde

00 ⇒ WDT
00 ⇒ prédiviseur de WDT

- Nombre de cycle d'horloge : 1
- Indicateurs du STATUS affectés: TO = 1 PD = 1

Ne pas utiliser dans une routine d'interruption

Syntaxe :

SLEEP
Description: Mise en veille du microcontrôleur (Power Down mode). La consommation de courant est minimale. 0 ⇒ PD 1 ⇒ TO 00 ⇒ WDT 00 ⇒ Prédiviseur de WDT • Nombre de cycle d'horloge : 1 • Indicateurs du STATUS affectés: PD=0 et TO