Département GEII

PIC 16F84

Philippe Hoppenot

hoppenot@lsc.univ-evry.fr

http://lsc.univ-evry.fr/~hoppenot/presentationfrancaise.html

Toutes les remarques sur le fond et la forme de ce document sont les bienvenues.

PIC 16F84

I. Qu'est-ce qu'un PIC ?

Un PIC est un microcontrôleur de chez Microchip. Ses caratéristiques principales sont :

Séparation des mémoires de programme et de données (architecture Harvard) : On obtient ainsi une meilleure bande passante et des instructions et des données pas forcément codées sur le même nombre de bits.

Communication avec l'extérieur seulement par des ports : il ne possède pas de bus d'adresses, de bus de données et de bus de contrôle comme la plupart des microprocesseurs.

Utilisation d'un jeu d'instructions réduit, d'où le nom de son architecture : RISC (Reduced Instructions Set Construction). Les instructions sont ainsi codées sur un nombre réduit de bits, ce qui accélère l'exécution (1 cycle machine par instruction sauf pour les sauts qui requirent 2 cycles). En revanche, leur nombre limité oblige à se restreindre à des instructions basiques, contrairement aux systèmes d'architecture CISC (Complex Instructions Set Construction) qui proposent plus d'instructions donc codées sur plus de bits mais réalisant des traitements plus complexes.

Il existe trois familles de PIC :

- Base-Line : Les instructions sont codées sur 12 bits
- Mid-Line : Les instructions sont codées sur 14 bits
- High-End : Les instructions sont codées sur 16 bits

Un PIC est identifié par un numéro de la forme suivant : xx(L)XXyy –zz

- xx : Famille du composant (12, 14, 16, 17, 18)
- L : Tolérance plus importante de la plage de tension
- 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

II. PIC 16F84

Device	Program Memory (words)	Data RAM (bytes)	Data EEPROM (bytes)	Max. Freq (MHz)
PIC16F83	512 Flash	36	64	10
PIC16F84	1 K Flash	68	64	10
PIC16CR83	512 ROM	36	64	10
PIC16CR84	1 K ROM	68	64	10

Figure[*] II.1 : *Liste des composants présentés dans la documentation n°DS30430C.*

[*] Figure prise dans la documentation technique n° DS30430C du PIC

Il s'agit d'un microcontrôleur 8 bits à 18 pattes. La documentation technique n°DS30430C porte sur plusieurs composants (Figure II.1).

Principales caractéristiques :

- 35 instructions
- Instructions codées sur 14 bits
- Données sur 8 bits
- 1 cycle machine par instruction, sauf pour les sauts (2 cycles machine)
- Vitesse maximum 10 MHz soit une instruction en 400 ns (1 cycle machine = 4 cycles d'horloge)
- 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 des pattes

La Figure* II.2 montre le brochage du circuit. 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
  Choix du mode programmation : 12V - 14V
  exécution : 4.5V - 5.5V

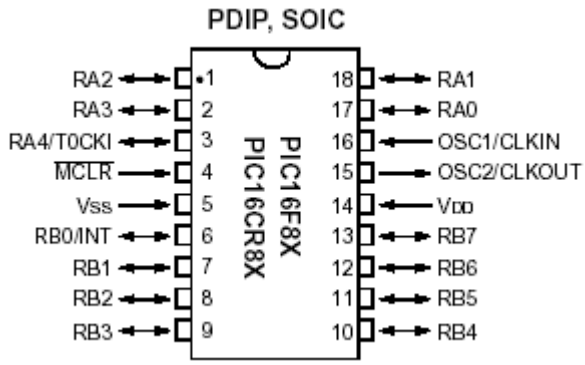


Figure* II.2 : *Brochage du circuit.*

II.2. Architecture générale

La Figure* II.3 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 (III.1 - XII)
- 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 (III.2.2)
  - 68 octets de données(III.2.1)

- une mémoire EEPROM de 64 octets de données (VIII)
- 2 ports d'entrées/sorties (VI)
- un compteur (timer) (VII)
- un chien de garde (watchdog) (X)

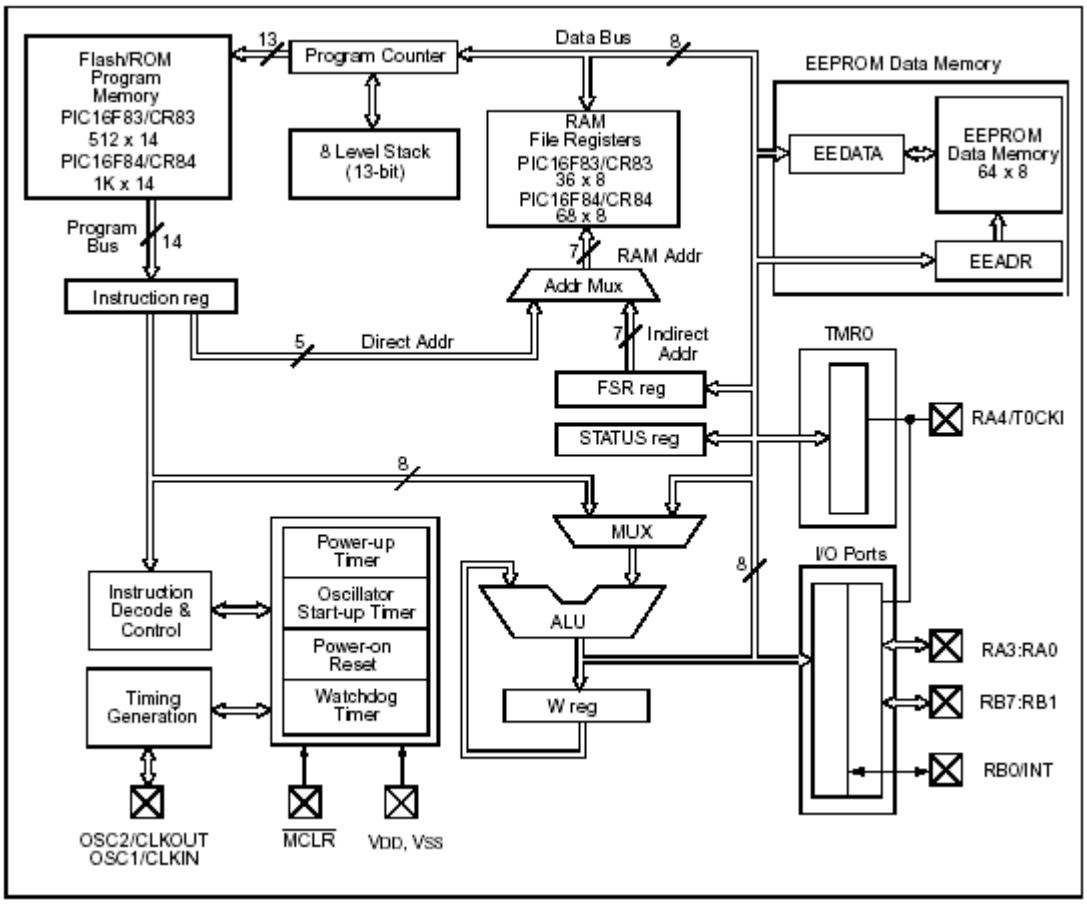


Figure* II.3 : *Architecture générale du PIC 16F8X.*

III. Organisation de la mémoire

Le PIC contient de la mémoire de programme et de la mémoire de données. La structure Harvard des PICs fournit un accès séparé à chacune. Ainsi, un accès aux deux est possible pendant le même cycle machine.

III.1. Mémoire de programme

C'est elle qui contient le programme à exécuter. Ce dernier est téléchargé par liaison série (voir § XII). La Figure* III.1 montre l'organisation de cette mémoire. Elle contient 1k "mots" de 14 bits dans le cas du PIC 16F84, même si le compteur de programme (PC) de 13 bits peut en adresser 8k. Il faut se méfier des adresses images ! L'adresse 0000h contient le vecteur du reset, l'adresse 0004h l'unique vecteur d'interruption du PIC.

La pile contient 8 valeurs. Comme le compteur de programme, elle n'a pas d'adresse dans la plage de mémoire. Ce sont des zones réservées par le système.

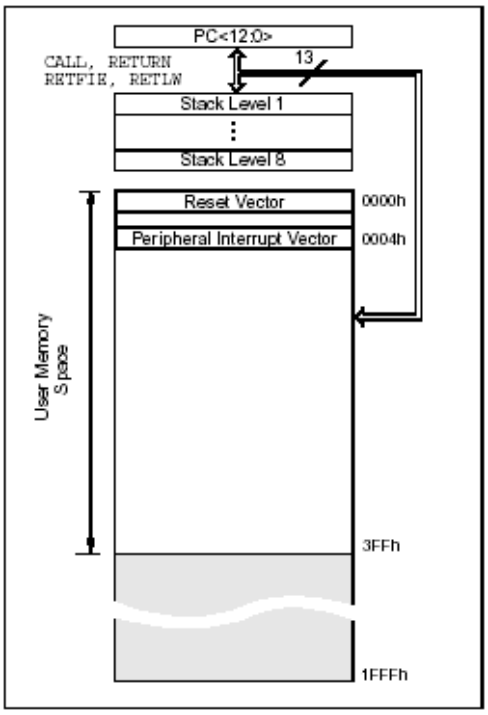


Figure* III.1 : *Organisation de la mémoire de programme et de la pile.*

III.2. Mémoire de données

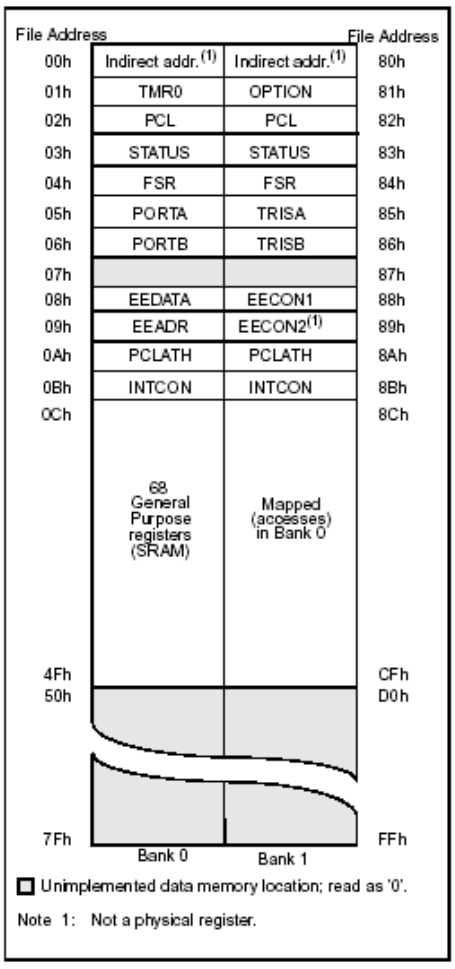


Figure* III.2 : *Organisation de la mémoire de données.*

Elle se décompose en deux parties de RAM (Figure* III.2) 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 dans le paragraphe IV, 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* III.2, 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.

III.2.1. Registres généraux

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

III.2.2. Registres spéciaux - SFRs

Ils permettent la gestion du circuit. Certains ont une fonction générale, d'autres un fonction spécifique attachée à un périphérique donné. La Figure* III.3 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 (Note 3)
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* III.3 : *Description des SFR.*

INDF (00h - 80h) : Utilise le contenu de FSR pour l'accès indirect à la mémoire (V.3).

TMR0 (01h) : Registre lié au compteur (VII).

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 (Figure* III.4).

FSR (04h - 84h) : Permet l'adressage indirect (V.3)

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 (VI).

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 (VIII).

EECON2 (89h) : Permet le contrôle d'accès à la mémoire EEPROM (VIII).

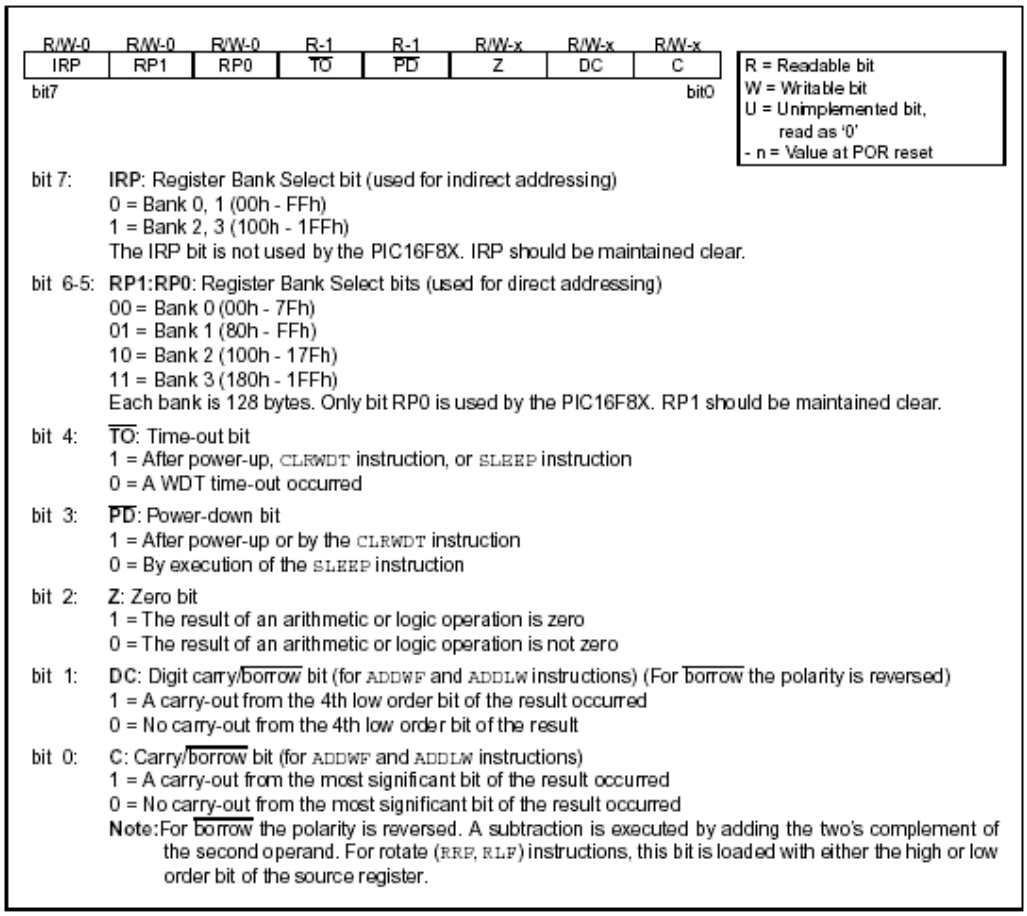
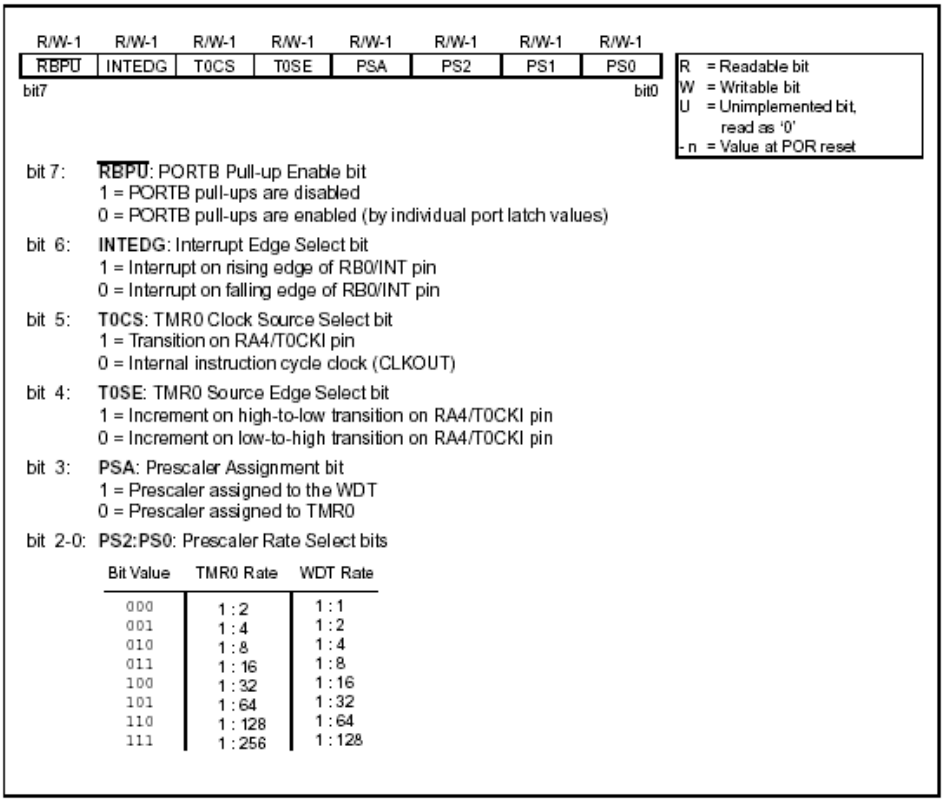


Figure* III.4 : *Registre d'étai du PIC - STATUS.*



R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **RBPU**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled (by individual port latch values)

bit 6: **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5: **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4: **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Figure* III.5 : *Registre de configuration de périphériques - OPTION_REG.*

III.2.3. 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 peut s'exécuter que selon des séquences particulières décrite au paragraphe VIII.

IV. Jeu d'instructions

Les PICs sont conçus selon une architecture RISC. Programmer avec un nombre d'instructions réduit permet de limiter la taille de leur codage et donc de la place mémoire et du temps d'exécution. Le format des instructions est présenté au paragraphe IV.1. La liste des instructions est ensuite donnée (IV.3) avant l'étude d'un exemple de description d'une instruction (IV.2).

IV.1. Format général

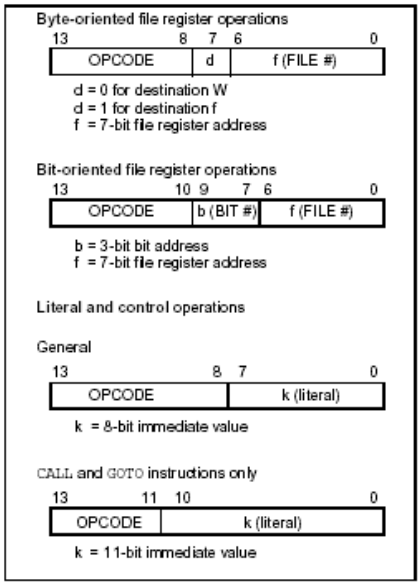


Figure* IV.1 : *Format général d'une instruction.*

Toutes les instructions sont codées sur 14 bits. Elles sont regroupées en trois grands types (Figure* IV.1) :

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

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

IV.2. Exemple d'instruction – le transfert

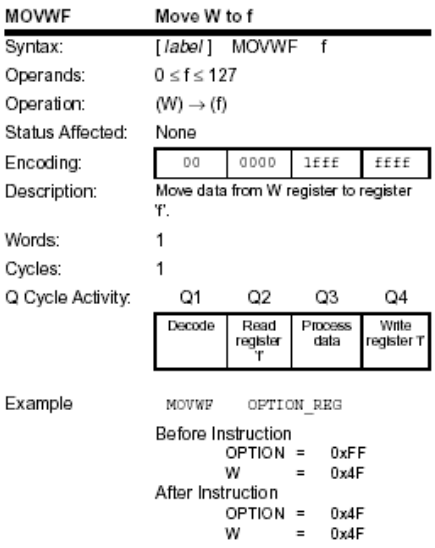
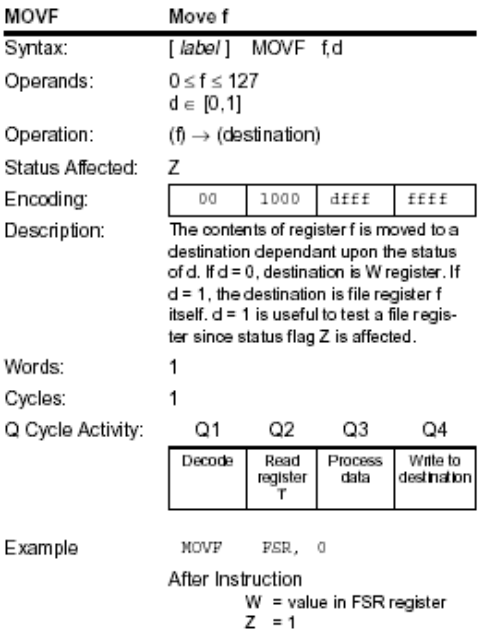


Figure* IV.2 : *Transfert du registre W dans le registre f.*

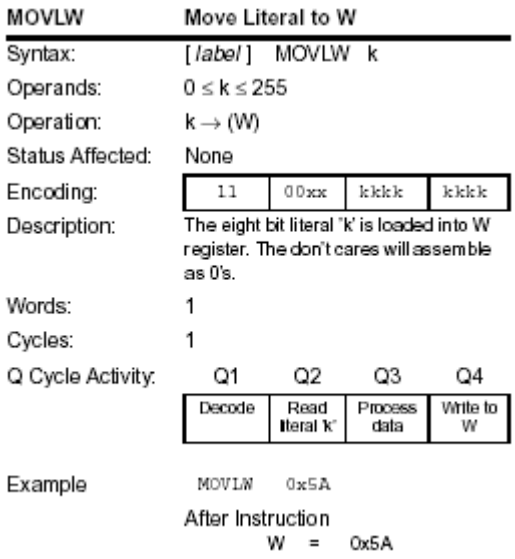
Trois instructions de transfert sont disponibles sur le PIC 16F84. La première (Figure* IV.2) permet de transférer le contenu du registre W dans un registre f. On peut noter la valeur du bit 7 à 1 et les bits 0 à 6 donnant le registre concerné.



MOVF	Move f
Syntax:	[*label*] MOVF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) → (destination)
Status Affected:	Z
Encoding:	00 | 1000 | dfff | ffff
Description:	The contents of register f is moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register since status flag Z is affected.
Words:	1
Cycles:	1
Q Cycle Activity:	Q1: Decode | Q2: Read register f | Q3: Process data | Q4: Write to destination
Example	MOVF FSR, 0 After Instruction W = value in FSR register Z = 1

Figure* IV.3 : *Transfert du contenu du registre f dans le registre W ou le registre f.*

La seconde (Figure* IV.3) permet de transférer une donnée contenue dans un registre f vers le registre W ou le registre f. Dans ce second cas, l'intérêt est de positionner le bit Z. On peut noter ici le bit 7 qui prend la valeur d fournie dans le code de l'instruction pour choisir la destination : W ou f.



MOVLW	Move Literal to W
Syntax:	[*label*] MOVLW k
Operands:	$0 \le k \le 255$
Operation:	k → (W)
Status Affected:	None
Encoding:	11 | 00xx | kkkk | kkkk
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.
Words:	1
Cycles:	1
Q Cycle Activity:	Q1: Decode | Q2: Read literal 'k' | Q3: Process data | Q4: Write to W
Example	MOVLW 0x5A After Instruction W = 0x5A

Figure* IV.4 : *Transfert d'une constante dans le registre W.*

La dernière instruction de transfert permet de charger une constante dans le registre W. Ici, la valeur à charger est donnée sur 8 bits, le bit 7 n'étant pas utile puisque le code de l'instruction dit que la valeur est à charger dans le registre W.

IV.3. Liste des instructions

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.
3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure* IV.5 : *Liste des instructions.*

La Figure* IV.5 donne la liste de toutes les instructions.

IV.4. Exécution d'un programme – notion de pipe-line

La Figure* IV.6 montre l'enchaînement des instructions tous les 4 cycles d'horloge. Pendant un premier cycle machine, l'instruction à exécuterest stockée en mémoire RAM. Le cycle suivant, elle est exécutée. Chaque instruction dure donc 2 cycles machine.

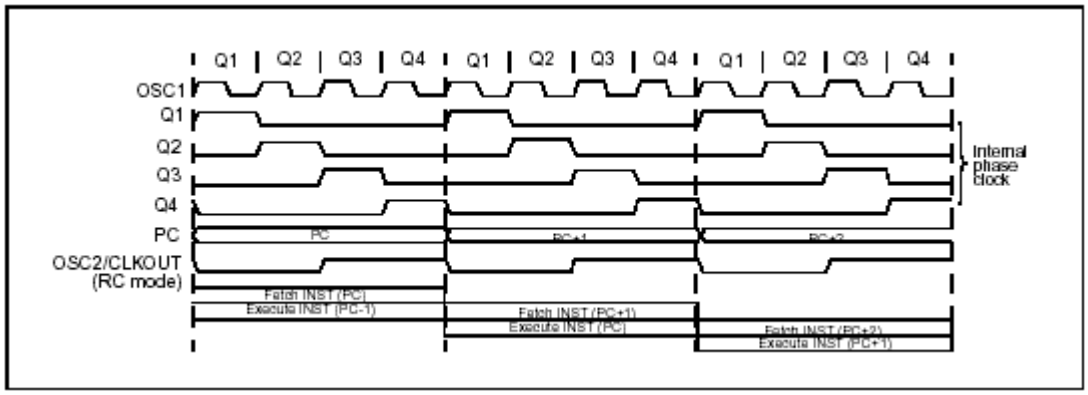


Figure* IV.6 : *Enchaînement des instructions.*

La notion de pipeline permet de réduire ce temps à un seul cycle machine. L'idée est d'exécuter l'instruction n-1 pendant que l'instruction n est chargée en mémoire RAM. Ainsi, une fois le système enclenché, pendant chaque cycle machine une instruction est chargée et un autre exécutée.

On a donc l'équivalent d'une instruction par cycle machine. La Figure* IV.7 montre un exemple d'exécution d'un programme. Notons que l'instruction CALL dure 2 cycles machine comme toutes les instructions de branchement.

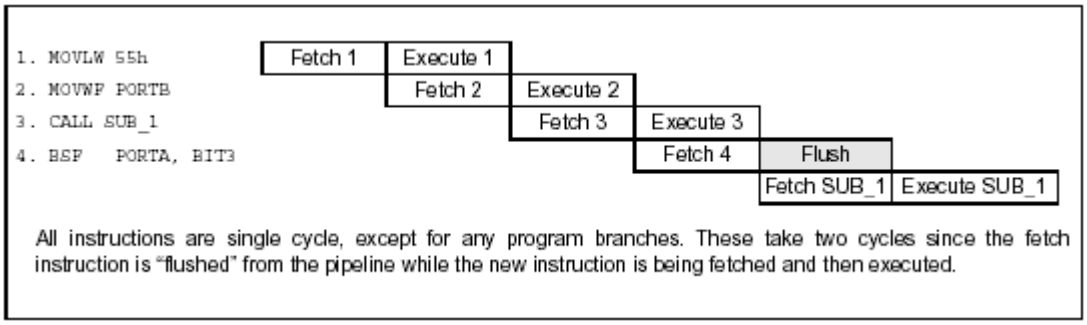


Figure* IV.7 : *Pipeline du PIC.*

V. 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 (V.1) : La donnée est contenue dans l'instruction.
- Adressage direct (V.2) : La donnée est contenue dans un registre.
- Adressage indirect (V.3) : L'adresse de la donnée est contenue dans un pointeur.

V.1. Adressage immédiat

La donnée est contenue dans l'instruction.

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

V.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.**

L'adresse 0x2B peut correspondre à 2 registres en fonction de la banque choisie (Figure* V.1). Le bit RP0 permet ce choix, le bit RP1 étant réservé pour les futurs systèmes à 4 banques.

V.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* V.1).

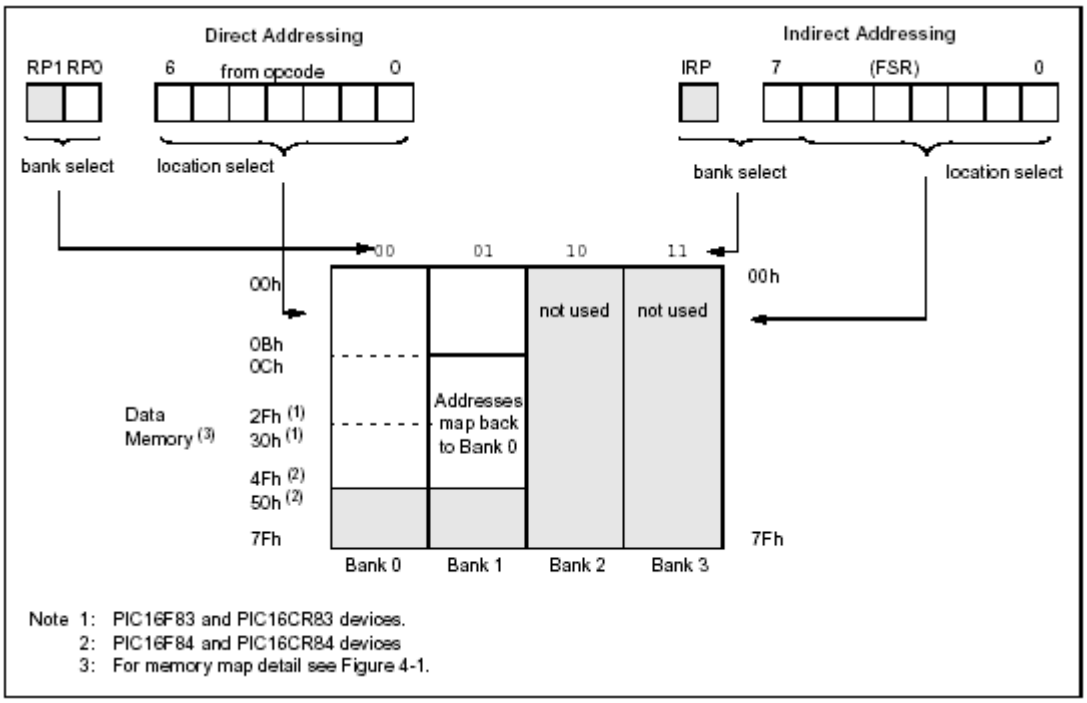


Figure* V.1 : *Adressages direct et indirect à la mémoire de données.*

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     ; Charge 1Ah dans W
           movwf   FSR      ; Charge W, contenant 1Ah, dans FSR
           movf    INDF, 0  ; Charge la valeur contenue à l'adresse 1Ah dans W
```

VI. Ports d'entrées/Sorties

Le PIC 16F84 est doté de deux ports d'entrées/Sorties appelés PortA et PortB.

VI.1. Port A

Il comporte 5 pattes d'entrée/sortie bi-directionnelles, notées RAx avec x={0,1,2,3,4} sur le brochage du circuit (Figure* II.2). Le registre PORTA, d'adresse 05h dans la banque 0, permet d'y accéder en lecture ou en écriture. Le registre TRISA, 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 Figure* VI.1 donne le câblage interne d'une patte du port A :

- "Data Latch" : Mémorisation de la valeur écrite quand le port est en sortie.
- "TRIS Latch" : Mémorisation du sens (entrée ou sortie) de la patte.
- "TTL input buffer" : Buffer de lecture de la valeur du port. La lecture est toujours réalisée sur la patte, pas à la sortie de la bascule d'écriture.
- Tansistor N : En écriture : Saturé ou bloqué suivant la valeur écrite.
  En lecture : Bloqué.
- Transistor P : Permet d'alimenter la sortie.

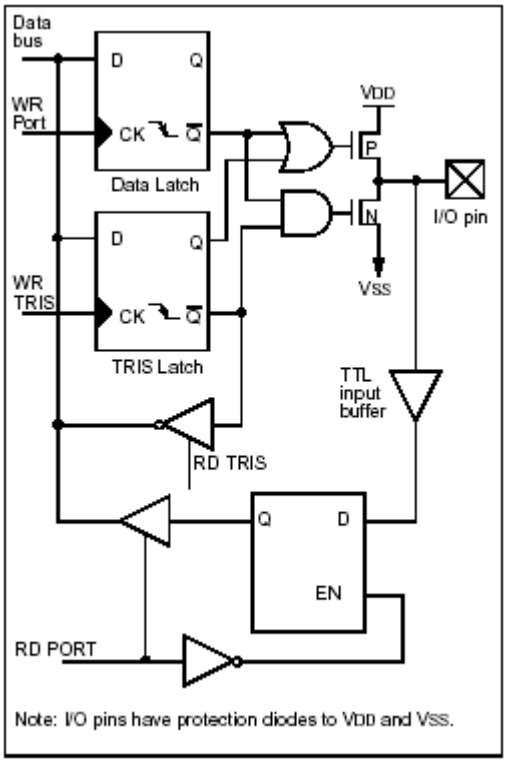


Figure* VI.1 : *Câblage interne d'une patte du port A.*

La patte RA4 peut aussi servir d'entrée de comptage pour le timer0.

VI.2. Port B

Il comporte 8 pattes d'entrée/sortie bi-directionnelles, notées RBx avec x={0,1,2,3,4,5,6,7} sur le brochage du circuit (Figure* II.2). Le registre PORTB, d'adresse 06h dans la banque 0, permet d'y accéder en lecture ou en écriture. Le registre TRISB, 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.

Le câblage interne d'une porte du port B ressemble beaucoup à celui du port A (Figure* VI.2). On peut noter la fonction particulière pilotée par le bit RBPU (OPTION_REG.7) qui permet d'alimenter (RBPU=0) ou non (RBPU=1) les sorties.

Les quatre bits de poids fort (RB7-RB4) peuvent être utilisés pour déclencher une interruption sur changement d'état (VII).

RB0 peut aussi servir d'entrée d'interruption externe.

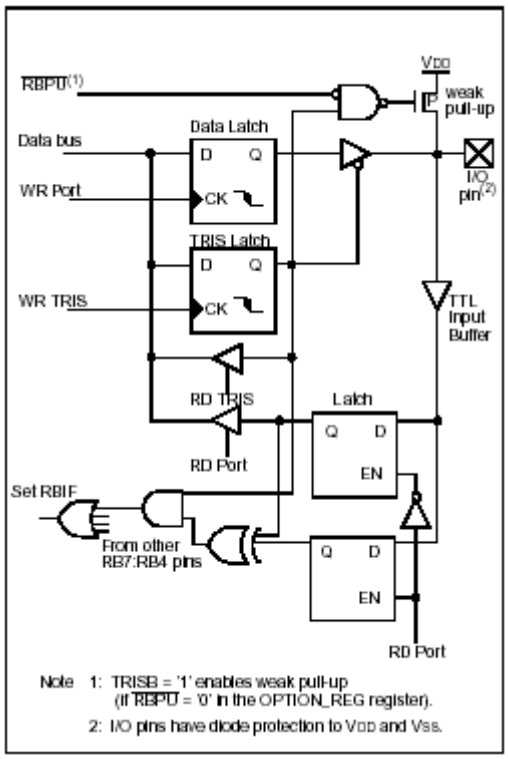


Figure* VI.2 : *Câblage interne d'une patte du port B.*

VII. Compteur

Le PIC 16F84 est doté d'un compteur 8 bits. La Figure* VII.1 en donne l'organigramme.

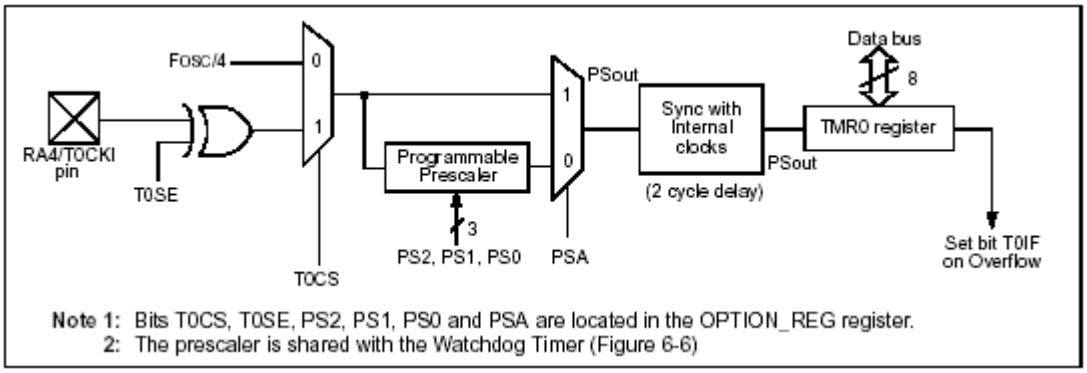


Figure* VII.1 : *Organigramme du Timer0.*

VII.1. Registre TMR0

C'est le registre de 8 bits qui donne la valeur du comptage réalisé. Il est accessible en lecture et en écriture à l'adresse 01h dans la banque 0.

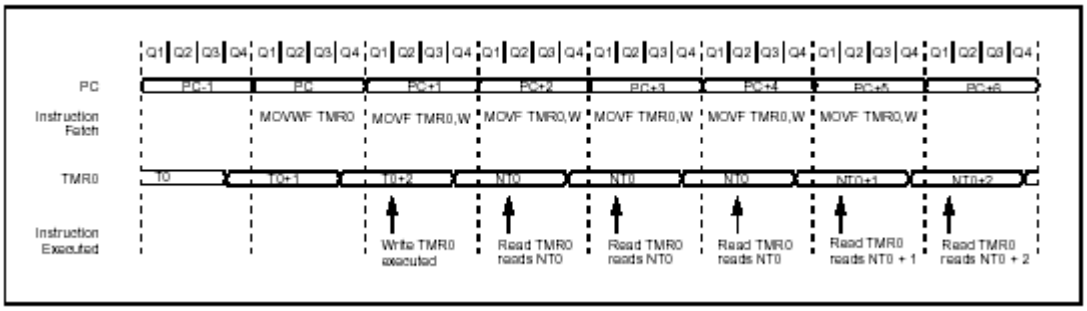


Figure* VII.2 : *Prise en compte de l'écriture dans le registre TMR0.*

Lors d'une écriture dans TMR0, le comptage est inhibé pendant deux cycles machine (Figure* VII.2). Si l'on veut déterminer un temps avec précision, il faut tenir compte de ce retard au démarrage.

VII.2. Choix de l'horloge

Le timer0 peut fonctionner suivant deux modes en fonction du bit T0CS (OPTION_REG.5). En mode timer (T0CS=0), le registre TMR0 est incrémenté à chaque cycle machine (si le pré-diviseur n'est pas sélectionné).

En mode compteur (T0CS=1), le registre TMR0 est incrémenté sur chaque front montant ou chaque front descendant du signal reçu sur la broche RA4/T0CKl en fonction du bit T0SE (OPTION_REG.4). Si T0SE=0, les fronts montants sont comptés, T0SE=1, les fronts descendants sont comptés.

VII.3. Pré-diviseur

En plus des deux horloges, un pré-diviseur, partagé avec le chien de garde, est disponible. 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** VII.3). Le bit PSA (OPTION_REG.3) permet de choisir entre la pré-division de timer0 (PSA=0) ou du chien de garde (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 VII.3 :** *Valeurs du pré-diviseur en fonction de PSA, PS2, PS2 et PS0.*

VII.4. 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 (§ VII). Le processeur est ainsi libre de travailler en attendant cet événement.

VII.5. Registres utiles à la gestion de timer0

Plusieurs registres ont été évoqués dans ce paragraphe. Ils sont synthétisés dans la Figure* VII.4.

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* VII.4 : *Registres utiles à la gestion de timer0.*

VIII. 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 peut s'exécuter que selon des séquences particulières.

VIII.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* VIII.1) 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 interrompu 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.

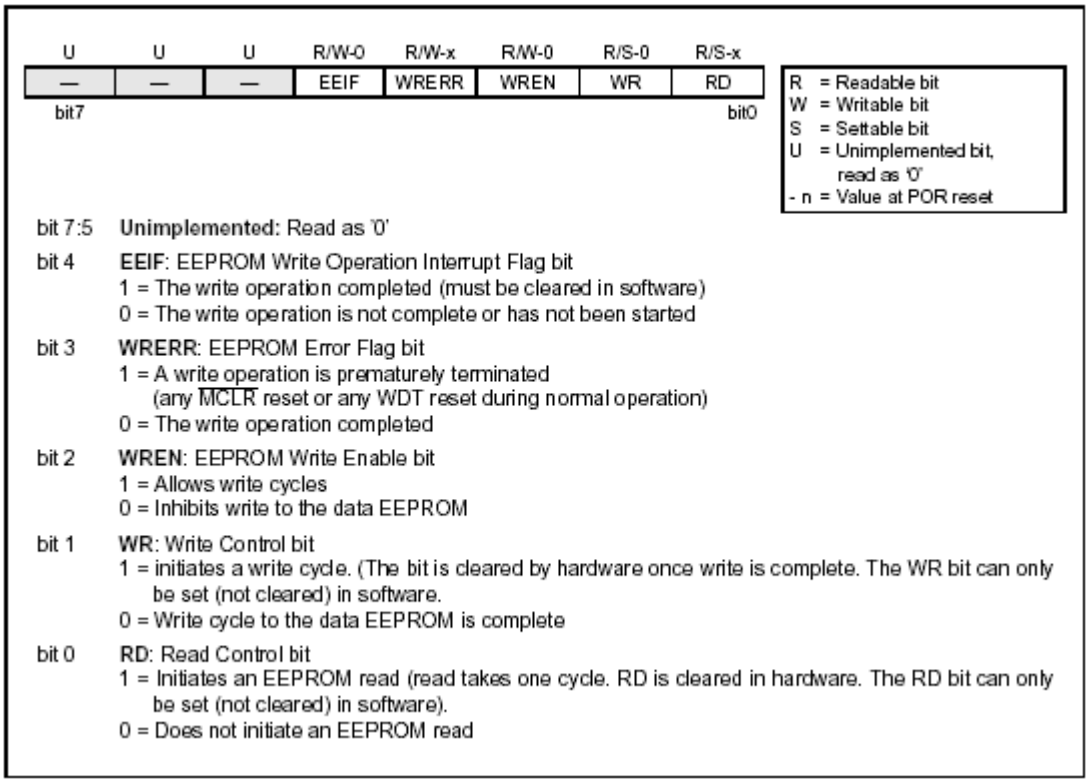


Figure* VIII.1 : *Registre EECON1.*

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

VIII.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
```

VIII.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 respecter 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           ;
MOVWF   EECON2        ; Write 55h
MOVLW   AAh           ;
MOVWF   EECON2        ; Write AAh
BSF     EECON1,WR     ; Set WR bit
                      ;   begin write
BSF     INTCON, GIE   ; Enable INTs.
```

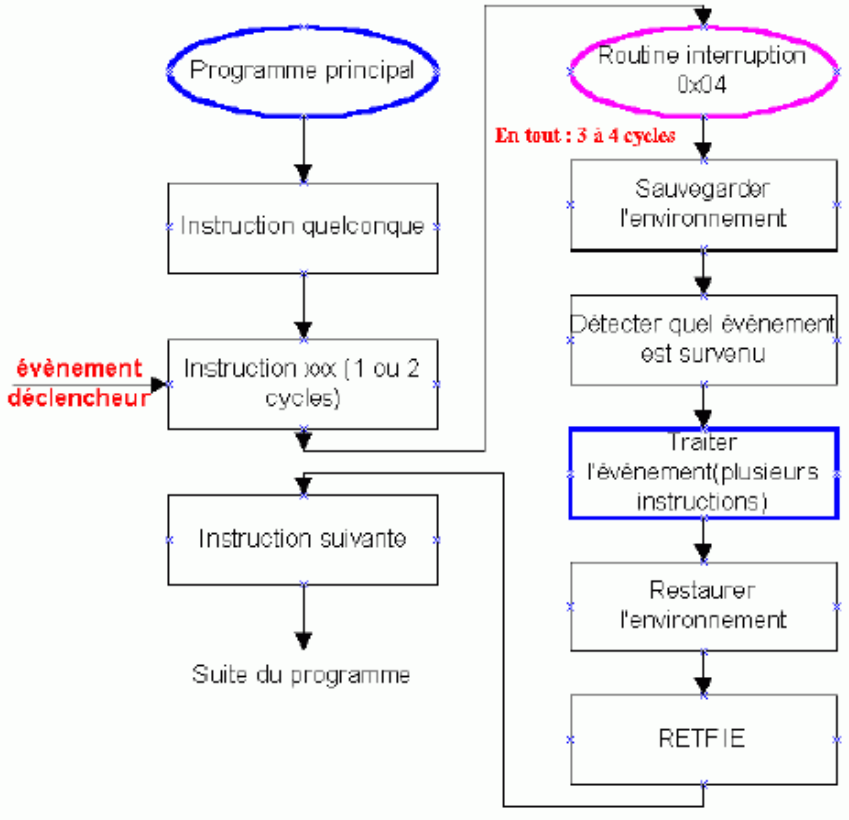
Required Sequence

IX. Interruptions

IX.1. Rappel - Notion de sous-programme

A développer

IX.2. Mécanisme



Figure[**] IX.1 : *Déroulement d'un programme lors d'une interruption.*

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

[**] Figure prise dans le cours de Bigonoff

IX.3. Différentes sources d'interruption

Dans le cas du PIC 16F84, il existe 4 sources d'interruption (Figure* IX.2) :

- 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

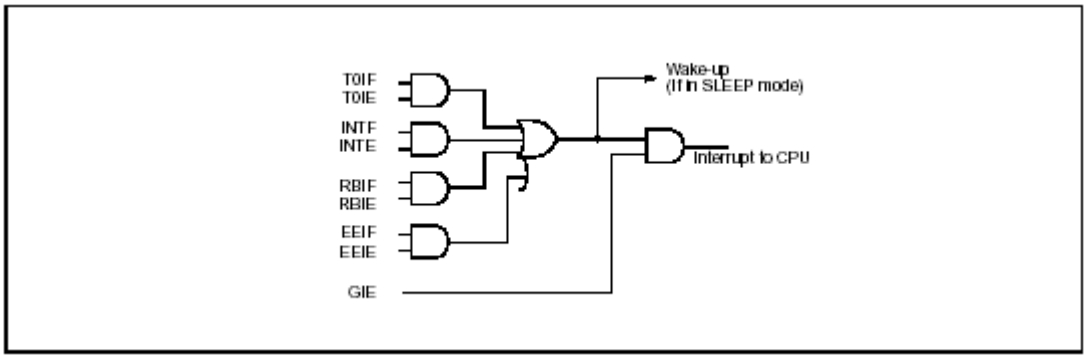


Figure* IX.2 : *Logique des événements associés aux interruptions.*

IX.4. Validation des interruptions

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **GIE**: Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts
Note: For the operation of the interrupt structure, please refer to Section 8.5.

bit 6: **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 5: **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: **INTE**: RB0/INT Interrupt Enable bit
1 = Enables the RB0/INT interrupt
0 = Disables the RB0/INT interrupt

bit 3: **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: **T0IF**: TMR0 overflow interrupt flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

bit 1: **INTF**: RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

bit 0: **RBIF**: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Figure* IX.3 : *Registre de contrôle d'interruption du PIC – INTCON*

Chacune de ses sources peut être validée indépendamment grâce aux bits 3 à 6 du registre INTCON (Figure* IX.3). Le bit GIE de ce même registre permet une validation générale des interruptions. Ainsi, pour que le déroutement du programme en cours soit déclenché, il faut qu'un des événements

extérieurs soit détecté, que l'interruption correspondante soit validée et que la validation générale soit activée.

IX.5. 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 (Figure* III.1). 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* IX.4 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

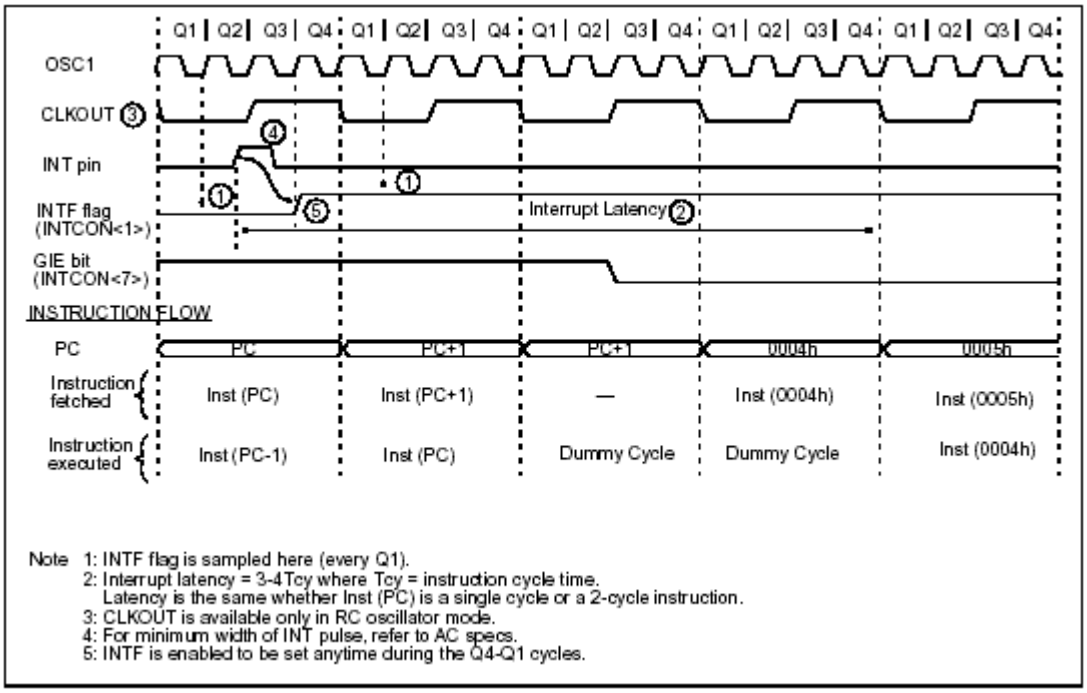


Figure* IX.4 : *Déroulement de l'appel à un sous-programme d'interruption.*

Deux cycles machine sont donc perdu à chaque interruption, sans compter la sauvegarde et la restitution du contexte (IX.6) et le retour au programme initial (IX.8).

IX.6. Sauvegarde et restitution du contexte

C'est un point important pour tous les sous-programme 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

registres sont utilisés dans le sous-programme d'interruption, il faut généralement les sauvegarder aussi.

IX.6.1. 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
```

IX.6.2. 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"
```

IX.6.3. 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.
```

IX.6.4. Résumé

```
; Création des variables de sauvegarde
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
```

```
;---------------------------
; Routine d'interruption
;---------------------------
; Sauvegarde du contexte
movwf     Sauve_W              ; Sauvegarde de W
swapf     STATUS, 0            ; Ecrit STATUS "swapé" dans W
movwf     Sauve_STATUS         ; Sauvegarde de STATUS "swapé"
; Traitement de l'interruption
; …
; Restitution du contexte
swapf     Sauve_STATUS, 0      ; Ecrit Sauve_Status "sawpé" dans W
movwf     STATUS               ; Restitue STATUS, "swapé" deux fois
swapf     Sauve_W, 1           ; Ecrit Sauve_W "swapé" dans lui-même
swapf     Sauve_W, 0           ; Restitue W, "swapé" deux fois.
```

IX.7. 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
```

IX.8. 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.

X. Chien de garde

X.1. 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.

X.2. 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.

X.3. 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).

X.4. 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.

XI. Mode sommeil

XI.1. 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.

XI.2. Gestion

XI.2.1. 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

XI.2.2. 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 interruption 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.

XII. Programmation sur site

Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RB6	CLOCK	I	Clock input
RB7	DATA	I/O	Data input/output
$\overline{MCLR}$	$V_{TEST\ MODE}$	P*	Program Mode Select
V_{DD}	V_{DD}	P	Power Supply
V_{SS}	V_{SS}	P	Ground

Legend: I =Input, O = Output, P = Power

Figure* XII.1 : *Pattes utilisées pour la programmation sur site.*

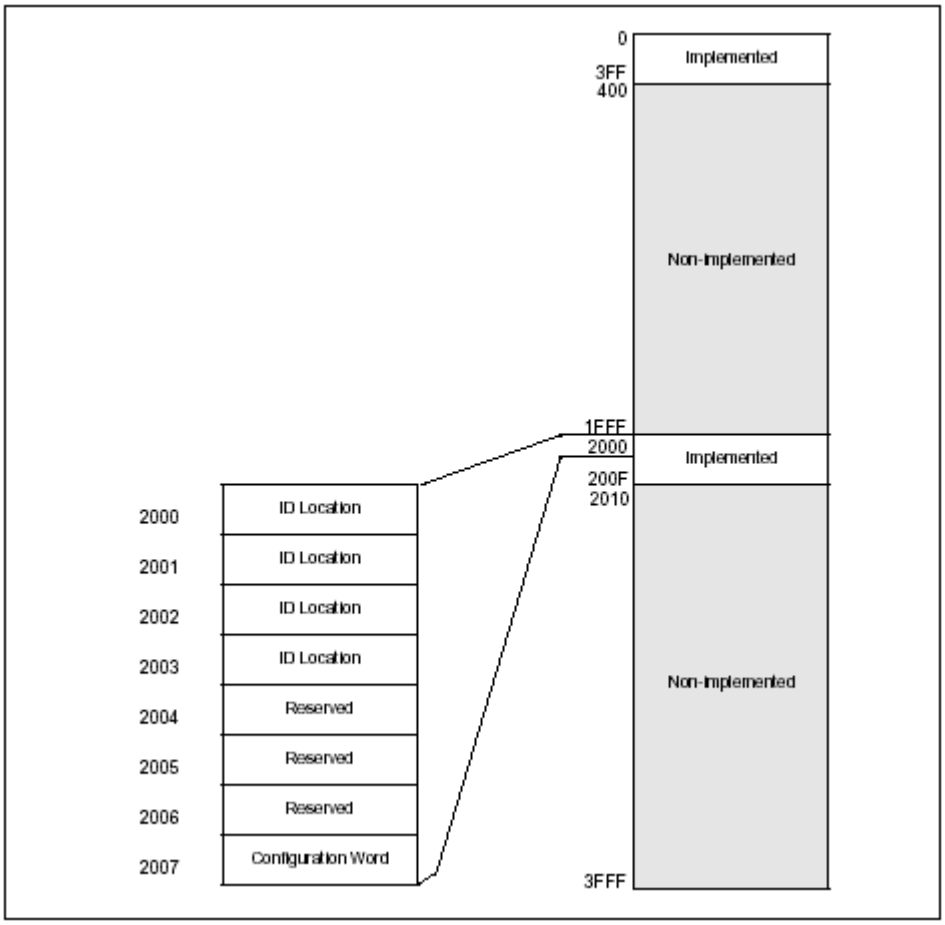


Figure* XII.2 : *Adresses de la mémoire de programme.*

Command	Mapping (MSB ... LSB)						Data
Load Configuration	0	0	0	0	0	0	0, data (14), 0
Load Data for Program Memory	0	0	0	0	1	0	0, data (14), 0
Read Data from Program Memory	0	0	0	1	0	0	0, data (14), 0
Increment Address	0	0	0	1	1	0	
Begin Programming	0	0	1	0	0	0	
Load Data for Data Memory	0	0	0	0	1	1	0, data (14), 0
Read Data from Data Memory	0	0	0	1	0	1	0, data (14), 0
Bulk Erase Program Memory	0	0	1	0	0	1	
Bulk Erase Data Memory	0	0	1	0	1	1	

Figure* XII.3 : *???.*

Standard Operating Conditions						
Operating Temperature	+10°C ≤ TA ≤ +40°C, unless otherwise stated, (25°C is recommended)					
Operating Voltage	4.5V ≤ VDD ≤ 5.5V, unless otherwise stated.					
Characteristic	Sym.	Min.	Typ.	Max.	Units	Conditions/Comments
Supply voltage during programming	VDDP	4.5	5.0	5.5	V	
Supply voltage during verify	VDDV	VDD min.		VDD max.	V	Note 1
High voltage on $\overline{MCLR}$ for test mode entry	VIHH	12		14.0	V	Note 2
Supply current (from VDD) during program/verify	IDDP			50	mA	
Supply current from VIHH (on $\overline{MCLR}$)	IIHH			200	µA	
$\overline{MCLR}$ rise time (VSS to VIHH) for test mode entry	tVHHR			1.0	µs	
(RB6, RB7) input high level	VIH1	0.8 VDD			V	Schmitt Trigger input
(RB6, RB7) input low level $\overline{MCLR}$ (test mode selection	VIL1	0.2 VDD			V	Schmitt Trigger input
RB6, RB7 setup time (before pattern setup time)	tset0	100			ns	
Data in setup time before clock ↓	tset1	100			ns	
Data in hold time after clock ↓	thld1	100			ns	
Data input not driven to next clock input (delay required between command/data or command/command)	tdly1	1.0			µs	
Delay between clock ↓ to clock ↑ of next command or data	tdly2	1.0			µs	
Clock to data out valid (during read data)	tdly3	80			ns	

Note 1: Program must be verified at the minimum and maximum VDD limits for the part.
Note 2: VIHH must be higher than VDD + 4.5V to stay in programming/verify mode.

Figure* XII.4 : *Caractéristiques électriques.*

FIGURE 5-1: LOAD DATA FOR PROGRAM MEMORY COMMAND (SERIAL PROGRAM/VERIFY)

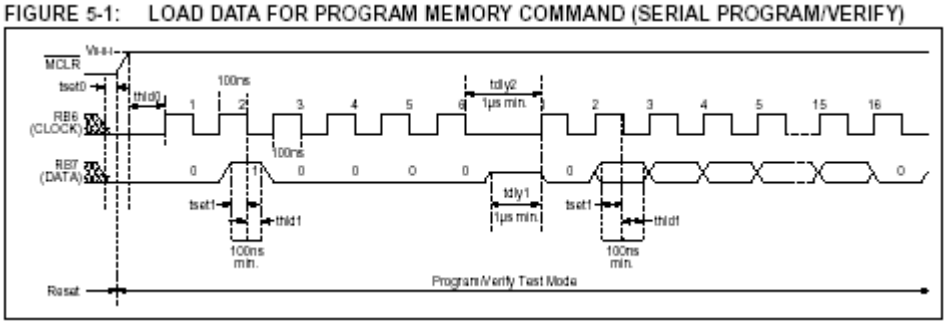


Figure* XII.5 : *Chronogramme d'écriture pour la programmation du PIC 16F84.*

FIGURE 5-2: READ DATA FROM PROGRAM MEMORY COMMAND (SERIAL PROGRAM/VERIFY)

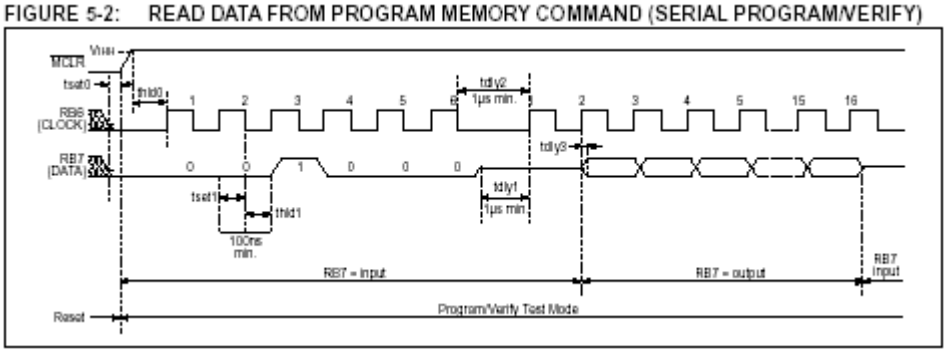


Figure* **XII.6 :** *Chronogramme de lecture pour la programmation du PIC 16F84.*

FIGURE 5-3: INCREMENT ADDRESS COMMAND (SERIAL PROGRAM/VERIFY)

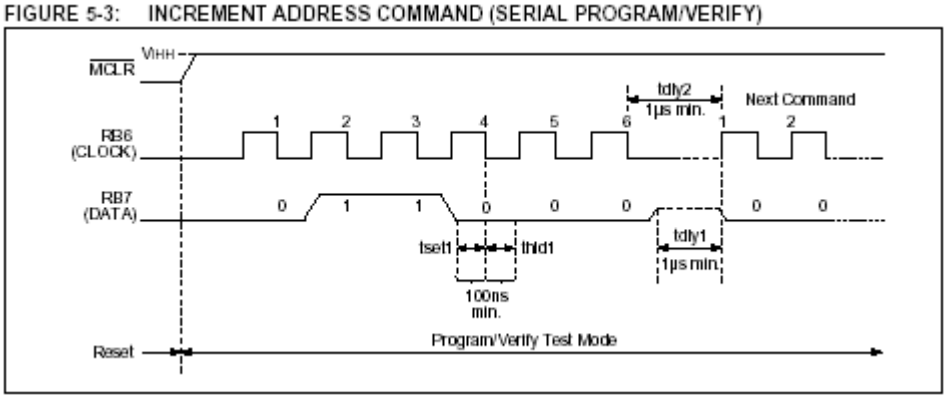


Figure* **XII.7 :** *Chronogramme de changement d'adresse pour la programmation du PIC 16F84.*

XIII. Bibliographie

1. Bigonoff (bigocours@hotmail.com) : "La programmation des PICs" - http://fribotte.free.fr/bdtech/cours/pic16f84/
   ** : Figures prises dans ce document
2. Lycée Jacquard : "Le PIC 16FXX" - http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm
3. Lycée Jacquard : "MPLAB" - http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm
4. Microchip : "PIC16F8X – 18-pin Flash/EEPROM 8-bit micro-controllers" – DS30430C, http://www.microchip.com/1010/suppdoc/
   * : Figures prises dans ce document
5. Microchip : "PIC16F8X – EEPROM memory programming specification" – DS30262E, http://www.microchip.com/1010/suppdoc/