

Chapitre 8 : Le microcontrôleur PIC16F84

1 Caractéristiques du microcontrôleur PIC16F84

C'est un microcontrôleur d'architecture RISC à 8 bits. Il est de la famille des PIC appelée Mid-Line. 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

Le PIC16F84 est doté de :

1. Une pile à huit niveaux de largeur 13 bits et plusieurs sources d'interruption internes et externes.
2. De deux bus, un bus d'instructions (14bits) et un bus de données (8 bits) qui sont séparés (architecture Harvard).
3. Toutes les instructions à cycle unique sauf pour les instructions de branchement qui sont à deux cycles.
4. Le pipeline d'instructions en deux étapes permet à toutes les instructions de s'exécuter en un seul cycle, à l'exception des instructions de branchement (les sauts) qui nécessitent deux cycles.
5. Un jeu d'instruction constitué de **35 instructions** (jeu d'instructions réduit).
6. La fréquence d'horloge externe max est de 10 MHz
7. Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) à 18 broches

Tableau 7.1. Caractéristiques générales du PIC16F84

		PIC16 F 84	PIC 16 CR 84
Mémoire	Mémoire de programme	1K × 14 bits (flash)	1K × 14 bits (ROM)
	Mémoire de données (octet)	68 × 8 bits	68 × 8 bits
	Mémoire de données EEPROM	64	64
	Pile	8 × 14 bits	8 × 14 bits
Périphériques	Modules Timer	Timer 0	Timer 0
Caractéristiques	Source d'interruption	4	4
	Les broches d'E/S	13	13
	Tension de fonctionnement	2V ~ 6V	2V ~ 6V
	Nombre de broches total	18	18
	Nombre de ports E/S	PORT A et PORT B	PORT A et PORT B
Horloge	Fréquence (max)	10 MHz	10 MHz

CR : Pour désigner la mémoire de programme est de type ROM.

F : Pour désigner la mémoire programme est de type Flash.

Les microcontrôleurs PIC avec mémoire Flash (**F**) permettent d'utiliser le même microcontrôleur pour le prototypage et la production. Ils sont reprogrammables, ce qui permet au code d'être mis à jour sans que le microcontrôleur ne soit retiré de carte électronique.

2 Architecture interne du PIC16F84

Le PIC16F84 est conçu selon une architecture RISC et utilise une **architecture Harvard**. Le microcontrôleur dispose d'un bus d'adresses pour la mémoire de programme et d'un autre pour la mémoire de données. La séparation de la mémoire de programme et de données permet aux instructions et aux données d'être dimensionnées différemment. Les mots de données sont de largeur 8 bits, par contre les codes machines des instructions sont de largeur 14 bits. Le bus de données de la mémoire de programme appelé **bus d'instructions**.

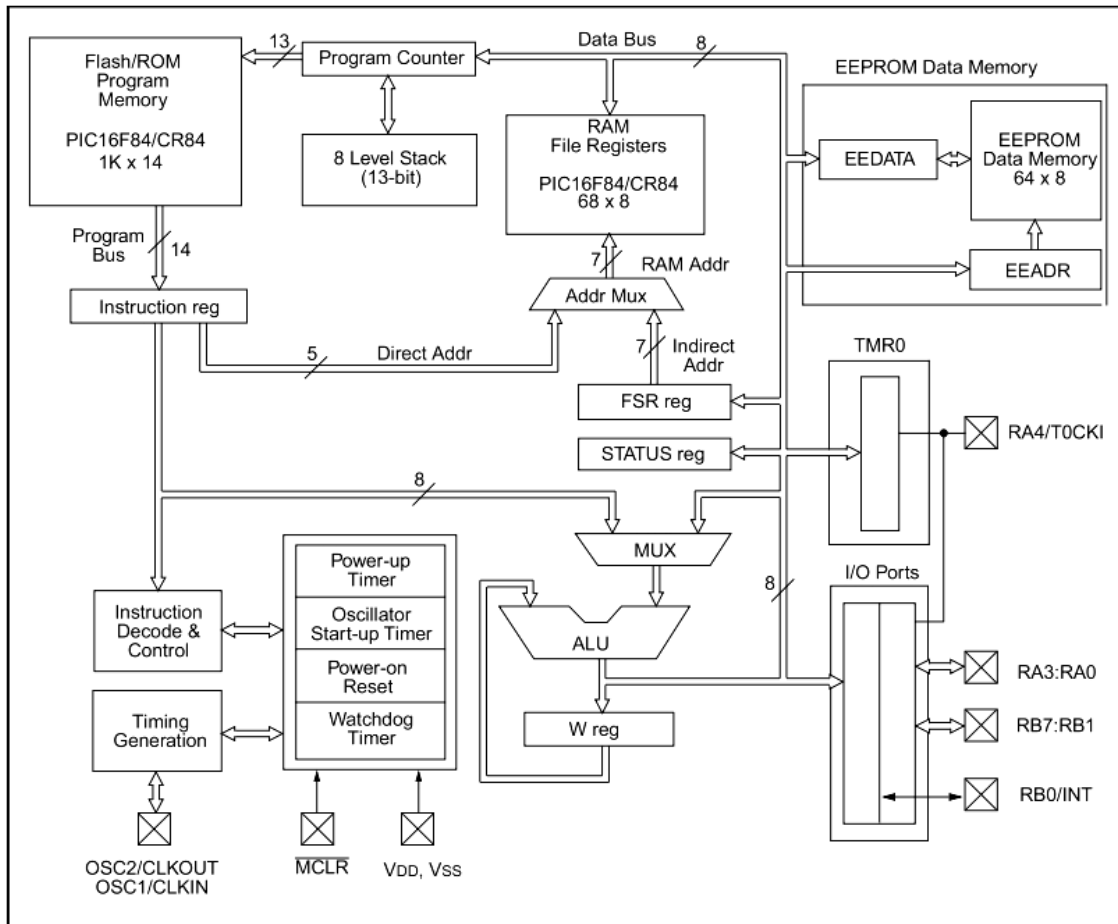


Figure 7.1. Architecture interne du microcontrôleur PIC16F84

3 Description des broches du pic16f84

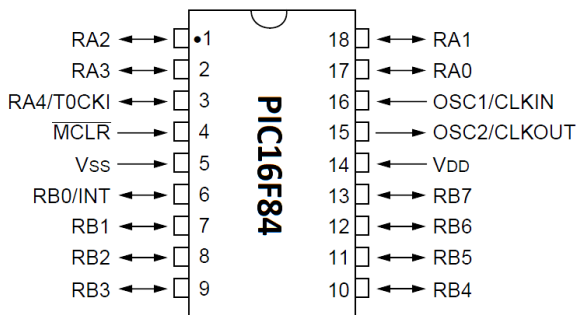


Figure 7.2. Brochage du PIC16F84

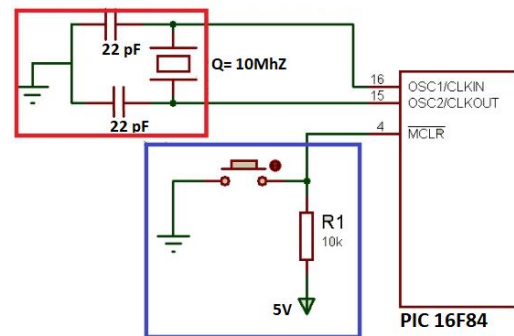


Figure 7.3. Circuits de reset et oscillateur pour le PIC

Les broches de l'alimentation

La broche 5 et la broche 14 doivent être respectivement associées aux bornes négative et positive de l'alimentation.

La broche de RESET

Il s'agit de la broche 14 appelée MCLR (Master Clear), elle sert à réinitialiser le programme du PIC à chaque fois qu'elle est maintenue à un niveau de tension bas (0V).

Les broches de l'horloge

Ces les broches 15 et 16 qui doivent être connectées à l'oscillateur à quartz, plus la fréquence du quartz utilisé est élevée, plus le processeur fonctionne rapidement.

Les ports d'entrées/sorties

Le PIC est doté de 13 broches GPIO (Global Purpose Input Output) réparties sur deux ports (PORTA et PORTB) et qui peuvent être configurées indépendamment comme entrée numérique ou comme sortie numérique. De plus, chaque broche peut fournir ou absorber un courant maximum de 25 mA par broche.

Le tableau ci-dessous décrit toutes les broches du PIC16F84

Tableau 7.1. Description des broches du PIC16F84

Nom de la broche	N° de la broche	Type	Description
OSC1/CLKIN	16	E	Entrée source d'horloge externe.
OSC2/CLKOUT	5	S	Sortie source d'horloge externe.
MCLR	4	E/A	(Master Clear) c'est l'entrée Reset, actif lorsqu'elle est au niveau bas.
VSS	5	A	Référence de masse (0V)
VDD	14	A	Alimentation positive
PORT A			
RA0	17	E/S	Bidirectionnelle : elle peut être configurée en entrée ou en sortie
RA1	18	E/S	Bidirectionnelle
RA2	1	E/S	Bidirectionnelle
RA3	2	E/S	Bidirectionnelle
RA4/ TOCKI	3	E/S	Bidirectionnelle/ Peut également être sélectionné pour être l'entrée d'horloge du temporisateur/compteur TMR0.
PORT B			
RB0/INT	6	E/S	Peut également être sélectionné comme broche d'interruption externe.
RB1	7	E/S	
RB2	8	E/S	
RB3	9	E/S	
RB4	10	E/S	Bidirectionnelle / Interruption lors du changement de broche.
RB5	11	E/S	Bidirectionnelle / Interruption lors du changement de broche.
RB6	12	E/S	Bidirectionnelle / Interruption lors du changement de broche.
RB7	13	E/S	Bidirectionnelle / Interruption lors du changement de broche.

Désignation : E : entrée, S : sortie, A : alimentation

Les broches du Port A et le du Port B sont bidirectionnelles

4 Cycle d'exécution d'une instruction

Chaque **cycle d'instruction (Tcy)** est composé de quatre cycles Q (Q1-Q4). Le cycle Q est le même que le cycle de l'oscillateur du microcontrôleur (TOSC). Le diagramme suivant montre la relation entre les cycles Q et le cycle d'instruction.

Les quatre cycles Q qui composent un cycle d'instruction (Tcy) peuvent être généralisés comme suit :

1. Q1 : Cycle de lecture d'instructions.
2. Q2 : Cycle de décodage d'instructions.
3. Q3 : Traiter les données
4. Q4 : Cycle d'écriture du résultat.

Chaque instruction affichera le fonctionnement détaillé du cycle Q pour l'instruction.

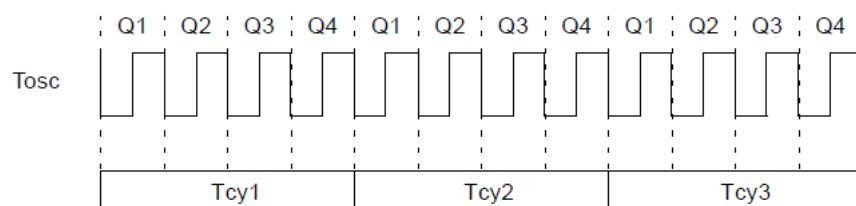


Figure 7.4. Cycle d'exécution d'une instruction

PC : c'est le compteur de programme (en anglais : 'Program Counter') ou pointeur d'instruction

Le temps de cycle

Le microcontrôleur PIC divise la fréquence d'horloge externe par 4 pour obtenir la fréquence de cycles d'instruction.

$$F_{cycle} = \frac{F_{osc}}{4}$$

$$T_{cycle} = T_{osc} \times 4$$

T_{osc} : période oscillateur (horloge externe)

Si l'oscillateur est de fréquence 10 Mhz, sa période égale $1/10^7$ s = **0,1 μ s = 100 ns**.

Le temps de cycle = **100ns \times 4 = 400ns**

Si horloge est de fréquence 4 Mhz, sa période égale $1/4 \cdot 10^6$ s = **250 ns**, et Le temps de cycle égale à 1000ns = 1 μ s.

Remarque : Toutes les instructions s'exécutent en un seul cycle, à l'exception les instructions de branchement.

5 Organisation de la mémoire de programme et de la pile

Le PIC16FXX dispose d'un compteur de programme 13 bits capable d'adresser un espace mémoire de programme de 8K x 14. Pour le PIC16F84, les premiers 1K x 14 (0000h-03FFh) sont physiquement implémentés (voir figure ci-dessous). Le vecteur de réinitialisation est à 0000h et le vecteur d'interruption est à 0004h.

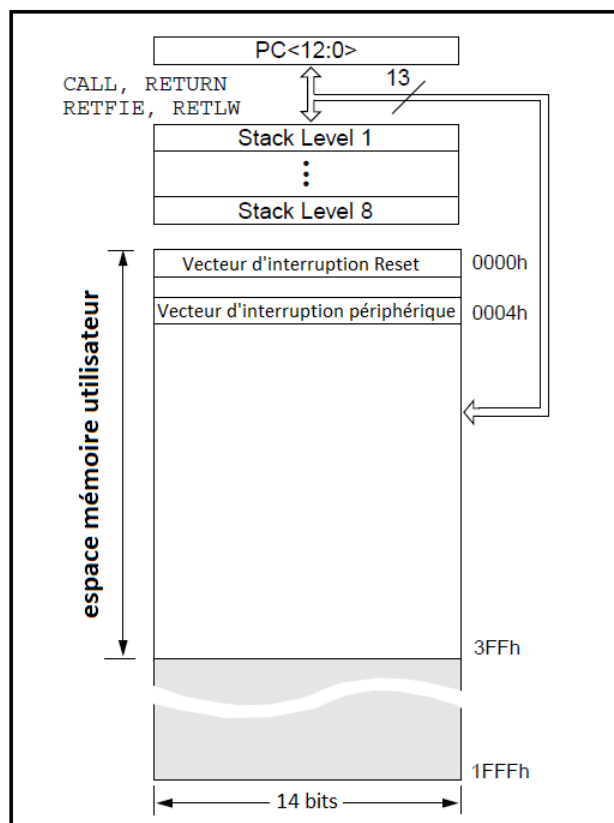


Figure7.5. Organisation de la mémoire de programme et de la pile

Compteur de programme (pointeur d'instruction)

Le compteur de programme (PC) a une largeur de 13 bits. L'octet de poids faible est le registre PCL, qui est un registre à lecture et à écriture. L'octet de poids fort du PC (PC<12:8>) n'est pas directement lisible ni inscriptible et provient du registre PCLATH.

Le registre PCLATH (PC latch high) est un registre de maintien pour PC<12:8>. Le contenu de PCLATH est transféré dans l'octet supérieur du compteur de programme lorsque le PC est chargé avec une nouvelle valeur. Cela se produit lors d'un CALL, d'un GOTO ou d'une écriture dans PCL. Les bits de poids fort de PC sont chargés à partir de PCLATH, comme illustré à la Figure ci-dessous.

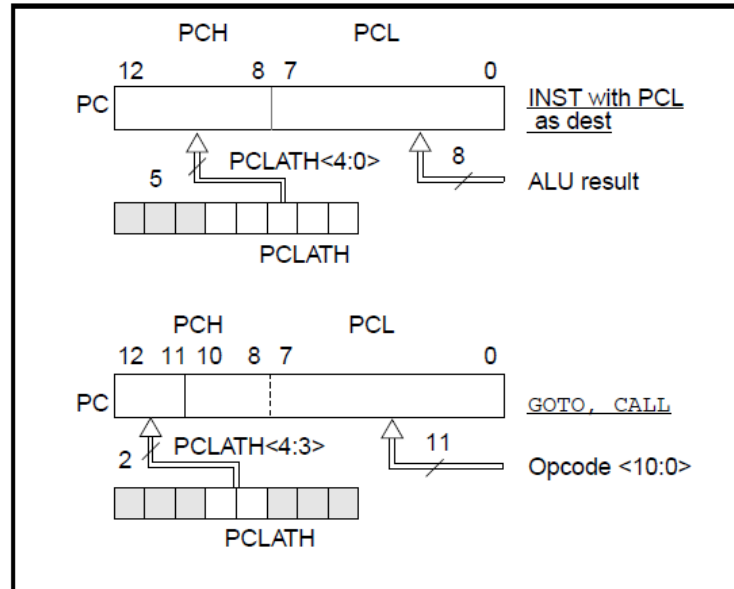


Figure 7.6. Chargement du pc dans différentes situations.

5.1 Organisation de la mémoire de données

La mémoire de données est divisée en deux zones (voir figure ci-dessous). La première est la zone des registres de fonctions spéciales (**SFR**) en anglais **Special Function Registers**, tandis que la seconde est la zone des registres à usage général (**GPR**) en anglais **General Purpose Registers**. Les SFR contrôlent le fonctionnement du microcontrôleur.

La mémoire de données est divisée en deux banques qui contiennent les registres à usage général et les registres de fonctions spéciales. La sélection de la banque nécessite l'utilisation de bits de contrôle. Ces bits de contrôle sont situés dans le registre STATUS (le registre d'état).

La sélection des banques se fait par l'intermédiaire des bits PR0 et PR1 du registre d'état (STATUS). La banque 0 est sélectionnée en positionnant (PR0=0 et PR1 = 0) et en positionnant (PR0=1 et PR1 = 0) du STATUS registre.

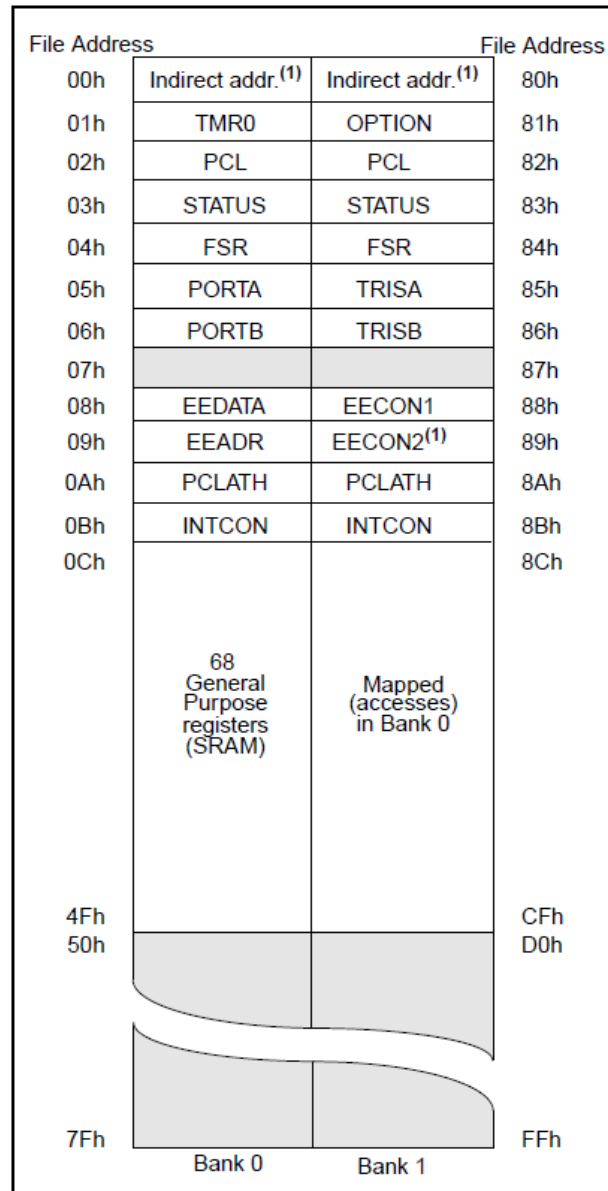


Figure 7.7. Organisation de la mémoire de programme et données

□ : zone mémoire non utilisé, lit comme '0'.

(1) : indirect addr n'existe pas physiquement.

5.1.1 Présentation des registres mémoire du PIC16F84

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/ \overline{TOCKI}	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⁾				---	0000	---	0000	
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	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	RBPU	INTEDG	TOCS	TOSE	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				---	1111	---	1111	
86h	TRISB	PORTB data direction register								1111	1111	1111	1111
87h		Unimplemented location, read as '0'								----	----	----	----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---	x000	---	q000
89h	EECON2	EEPROM control register 2 (not a physical register)								----	----	----	----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC ⁽¹⁾				---	0000	---	0000	
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	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.

5.1.2 Registre d'état (STATUS) (ADDRESS 03h, 83h)

Le registre d'état appelé STATUS d'adresse 03 dans la bank0, contient l'état de certaines opérations effectuées par le processeur (C, DC et Z). De plus, les bits TO et PD sont à lecture seulement. Les bits PR0 et PR1 sont utilisés pour la sélection des banques de données dans la mémoire de données.

Le registre STATUS possède une copie dans la bank1 à l'adresse 83H.

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	TO	PD	Z	DC	C	
bit7								bit0

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

bit 0 : C : bit de retenue

1 : si retenue.

0 : s'il n'y a pas de retenue.

bit 1 : DC : Demie retenue

1 : si on a une retenue du quarter (4 bits) de poids faible dans le quarter de poids plus fort.

0 : pas de retenue du quarter (4 bits) de poids faible dans le quarter de poids plus fort.

bit 2 : Z : bit zéro

1 : lorsque le résultat d'une opération arithmétique ou logique est zéro

0 = lorsque le résultat d'une opération arithmétique ou logique n'est pas zéro

bit 4 : TO : bit de temporisation (Timer0)

1 = Après la mise sous tension, instruction CLRWDT ou instruction SLEEP

0 = Une temporisation WDT s'est produite

bit 3 : PD : bit de mise hors tension

1 = Après mise sous tension ou par l'instruction CLRWDT

0 = Par exécution de l'instruction SLEEP

bit 6-5 : RP1:RP0 : Bits de sélection de banque de registre (utilisés pour l'adressage direct)

00 = Banque 0 (00h - 7Fh)

01 = Banque 1 (80h - FFh)

Chaque banque est de 128 octets.

bit 7 : IRP : bit de sélection de banque de registre (utilisé pour l'adressage indirect)

0 = Banque 0, 1 (00h - FFh)

Le bit IRP doit être maintenu à '0'.

5.1.3 OPTION_REG REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBP \bar{U}	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 2-0: PS2:PS0: Bits de sélection de taux de pré-diviseur.

PS2 PS1 PS0	Taux pour TMRO	Taux WTD
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

bit 3: PSA: bit d'assignation du pré-diviseur

- 1 = Prescaler assigné au WDT (watch dog : en français chien de garde)
- 0 = Prescaler assigné au TMRO

bit 4: T0SE: bit de sélection de type de front utilisé par la broche d'entrée RA4/TOCKI

- 1 = Interruption sur front montant de la broche RA4/TOCKI.
- 0 = Interruption sur front descendant de la broche RA4/TOCKI.

bit 5: T0CS: TMRO Bit de sélection de la source horloge utilisée par le timer

- 1 = Transition sur la broche d'entrée RA4/TOCKI.
- 0 = Horloge de cycle d'instructions interne (CLKOUT).

bit 6: INTEDG: bit de sélection de type de front utilisé par l'interruption RBO

- 1 = Interruption sur front montant de la broche RBO/INT.
- 0 = Interruption sur front descendant de la broche RBO/INT.

bit 7: RBP \bar{U} : PORTB Pull-up Enable bit

- 1 = Les pull-ups du PORTB sont désactivés
- 0 = Les pull-ups du PORTB sont activés

5.1.4 INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							bit0

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

bit 0: RBIF: ce bit signale qu'une des entrées RB4 à RB7 a été modifiée.

- 1 = Lorsqu'au moins une des broches RB7:RB4 a changé d'état (doit être effacée par programme).
- 0 = Aucune des broches RB7:RB4 n'a changé d'état.

bit 1: INTF: ce bit signale une transition sur la broche RBO dans le sens demandé.

- 1 = une transition sur la broche RBO est effectuée.
- 0 = pas de transition sur la broche RBO n'a été effectuée.

bit 2: TOIF: ce bit signale le débordement du TMR0.

- 1 = débordement du TMR0 effectué (il doit être remis à 0 par programme).
- 0 = débordement du TMR0 n'est pas effectué.

bit 3: RBIE: validation de l'interruption si un changement sur l'une des broches RB4 :RB7.

- 1 = activation de l'interruption si un changement sur l'une des broches RB4 :RB7.
- 0 = désactivation de l'interruption si un changement sur l'une des broches RB4 :RB7.

bit 4: INTE: validation de l'interruption si un changement de niveau dans la broche RBO.

- 1 = activation de l'interruption RBO.
- 0 = désactivation de l'interruption RBO.

bit 5: TOIE: bit de validation de l'interruption générée par le débordement du timer0.

- 1 = activation de l'interruption générée par le débordement du timer0.
- 0 = désactivation de l'interruption générée par le débordement du timer0.

bit 6: EEIE: Ce bit permet de valider l'interruption de fin d'écriture en EEPROM.

- 1 = activation de l'interruption de fin d'écriture en EEPROM.
- 0 = désactivation de l'interruption de fin d'écriture en EEPROM.

bit 7: GIE: bit de validation de toutes les interruptions

- 1 = activation de toutes les interruptions masquable
- 0 = désactivation de toutes les interruptions

6 Les Modes d'adressages

Les instructions de transfert de données permettent de déplacer :

- Une valeur immédiate (L) vers l'accumulateur (W), en utilisant l'instruction **MOVLW**.
- Le contenu de l'accumulateur (W) vers une donnée mémoire, en utilisant l'instruction **MOVWF**.
- Une donnée mémoire vers l'accumulateur (W), en utilisant l'instruction **MOVFW**.

La totalité de la mémoire de données est accessible soit directement en utilisant l'adresse directe de chaque registre mémoire, soit indirectement via le registre de sélection de fichier (FSR). L'adressage indirect utilise la valeur actuelle des bits RP1: RP0 du registre d'état pour accéder aux zones mises en banque de la mémoire de données.

6.1 Adressage immédiat

L'opérande apparaît dans l'instruction, l'opérande est une valeur **constante** (appelée aussi en anglais **Literal**).

Exemple : **MOVLW 0XC4** ; L'instruction charge le accumulateur **W** (working register) avec la valeur hexadécimale C4H.

Remarque : le microcontrôleur PIC16F84 manipule des données sur 8 bits seulement.

6.2 L'adressage direct

Avec l'adressage direct, il faut en premier sélectionner la banque (banque 0 ou banque 1) de l'emplacement mémoire désiré, puis indiquer l'adresse de l'emplacement mémoire à l'instruction à exécuter.

Exemple : **BCF STATUS, 5** ; Sélection de la banque 0 (**RP1 = 0 et RP0 = 0**)
MOVF 0X6, 0 ; Charger le contenu du registre mémoire d'adresse 6H dans W.

Sachant que l'adresse 06H est l'adresse du registre PORTB dans la mémoire de donnée.

6.3 L'adressage indirect

Le registre **INDF** n'est pas un registre physique. Le registre INDF contient le contenu de l'emplacement mémoire pointé par le registre mémoire FSR. **Le registre FSR est un pointeur mémoire**, il contient l'adresse de l'emplacement du registre mémoire.

Exemple :

Le registre mémoire 0CH contient la valeur 22H et Le registre mémoire 0DH contient la valeur 33H. Ils appartiennent tous les deux à la banque 0.

```

BCF STATUS, 5 ; Sélectionner la banque 0
BCF STATUS, 6
MOVLW 0X0C ; initialisation du pointeur
MOVWF FSR ; FSR est pointeur mémoire, contient la valeur 0CH
MOVF INDF, 0 ; transfert le contenu du registre mémoire d'adresse 0CH dans W puisque d=0.
(W=22H)
INCF FSR ; FSR ← FSR + 1, donc FSR = 6
MOVF INDF, 0 ; transfert le contenu du registre mémoire d'adresse 0DH dans W puisque d=0.
(W=33H)

```

7 Les ports d'entrée/sortie

Le PIC16F84 a deux ports d'entrées/sorties, le PORTA et le PORTB, appelés PortA et PortB. Les valeurs de sorties sur les ports sont mémorisées. Les valeurs lues à partir des ports ne sont pas mémorisées.

La mise à '1' sur une position n du PORT en sortie, correspond à une tension de 5V sur la Broche n.

7.1 Le PORTA

Il possède 5 pattes d'entrée/sortie bi-directionnelles, le registre **PORTA**, d'adresse 05h dans la banque 0 peut-être configuré en entrée (lecture) ou en sortie (écriture). Chaque bit de ce registre représente une patte.

Tableau7.x Présentation des registres associés au fonctionnement du PORTA

	Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BANK0	05h	PORTA	—	—	—	RA4/TOCKI	RA3	RA2	RA1	RA0
BANK1	85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

Le registre **TRISA**, d'adresse 85h dans la banque 1, permet de choisir le sens de chaque broche (entrée ou sortie).

- Chaque bit positionné à **1** du TRISA configure la broche correspondante **en entrée**.
- Chaque bit positionné à **0** du TRISA configure la broche correspondante **en sortie**.

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

Exemple :

Pour configurer toutes les broches du port A en entrée, on doit charger la valeur FFh dans le registre TRISA.

```
BSF      STATUS, RPO    ; sélectionner la Bank 1
MOVLW   0xFF           ; W ← FFh
MOVWF   TRISA          ; TRISA ← W
```

Si on veut configurer les broches RA0 et RA1 en entrée, les broches RA2 et RA3 en sortie et RA4 en entrée

```
BCF      STATUS, RPO    ; Sélectionne la Bank 0, puisque le registre PORTA est situé dans la Bank 0.
CLRF    PORTA          ; Initialisation des broches du PORTA en sorties à '0'.
BSF      STATUS, RPO    ; Sélectionne la Bank 1, pour configurer la direction des broches
MOVLW   B'00010011'    ; Configure RA<1:0> en entrée, RB<3:2> en sortie et RB4 en entrée
MOVWF   TRISA
```

7.2 Le PORTB

Il est doté de 8 broches d'entrée/sortie bi-directionnelles, le registre **PORTB**, d'adresse 06h dans la banque 0 peut-être configuré en entrée (lecture) ou en sortie (écriture). Chaque bit de ce registre représente une patte.

Tableau7.x Présentation des registres associés au fonctionnement du PORTB

	Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BANK0	06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
BANK1	86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
BANK1	81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Le registre **TRISB**, d'adresse 86h dans la banque 1, permet de choisir le sens de chaque broche (entrée ou sortie).

- Chaque bit positionné à **1** du TRISB configure la broche correspondante **en entrée**.
- Chaque bit positionné à **0** du TRISB configure la broche correspondante **en sortie**.

Exemple:

```
BCF      STATUS, RPO    ; Sélectionne la Bank 0, puisque le registre PORTB est situé dans la Bank 0.
CLRF    PORTB          ; Initialisation du PORTB
BSF      STATUS, RPO    ; Sélection de la Bank 1
MOVLW   B'11001111'    ; Valeur utilisée pour initialiser la direction du PORTB
MOVWF   TRISB          ; Configure RB<3:0> en entrée, RB<5:4> en sortie et RB<7:6> en entrée
```

8 Le module TIMER0

Le PIC16F84 est doté d'un seul module de Timer sur 8 bits, contrairement à d'autres microcontrôleur PIC de la même famille (Mid-range) tel que le PIC16F877 qui comporte 3 timers (voir chapitre précédent). La principale fonction du Timer est le comptage (autrement dit c'est compteur).

Le tableau ci-dessous présente les différents registres associés au fonctionnement du module TIMER0.

Tableau7.x Présentation des registres associés au fonctionnement du TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h	TMR0								
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
81h	OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

NB : Les bits en couleur rouge sont associés au fonctionnement du TIMER.

8.1 Les modes de fonctionnement du module Timer

Il a deux modes de fonctionnement, **temporisateur** ou **compteur d'impulsion**. La sélection de l'un de ces deux modes s'effectue par le bit 5 du registre **OPTION_REG** appelé **TOCS** (en anglais **Timr0 Clock Source Select bit**).

- **TOCS = 0** : Fonctionnement en mode **timer**
- **TOCS = 1** : Fonctionnement en mode **compteur**

8.1.1 Mode compteur d'impulsions

Dans ce mode, il compte les impulsions reçues sur la broche **RA4/TOKI**. Il peut compter que jusqu' à 255 impulsions puisque le registre TMR0 est un registre mémoire 8 bits, en cas de dépassement, il recommence à 0. La lecture du registre TMR0 nous donnera le nombre d'impulsion survenus sur la broche RA4/TOKI.

Dans ce mode, nous devons préciser sur quel front (montant ou descendant) le comptage s'effectuera. Ceci est précisé grâce au bit 4 du registre **OPTION** appelé **TOSE** (en anglais **Timer0 Source Edge select bit**).

- **TOSE = 0** : le comptage se fait sur **front montant**, lors du passage de l'entrée (RA4/TOKI) de **0 à 1**.
- **TOSE = 1** : le comptage se fait sur **front descendant**, lors du passage de l'entrée (RA4/TOKI) de **1 à 0**.

8.1.2 Mode temporisateur (ou compteur de temps)

Dans ce mode, il compte les **cycles d'horloge du PIC**, donc il compte le temps. Le débordement du registre TMR0, c-à-d le passage de FFH à 00H positionne le drapeau **TOIF** du registre **INTCON** à **1**.

Le débordement peut être détecté par scrutation ou par interruption :

- **Par scrutation** : le programme interroge le bit TOIF pour détecter le débordement de TMR0.
- **Par interruption** : il faut activer l'interruption timer en positionnant le bit **TOIE** à 1 (**TOIE TMR0 Overflow Interrupt Enable**). Lorsque **TOIF** se positionne à 1, l'interruption se produit. Le bit TOIF doit être remis à 0 par le sous-programme de service d'interruption Timer0 avant de réactiver cette interruption.

Remarque : L'interruption TMR0 ne peut pas réveiller le processeur du sommeil (mode sleep) puisque le timer est à l'arrêt pendant le mode SLEEP.

```

CLRF      TMR0      ; début du comptage dans 2 cycles
BCF      INTCON , TOIF ; effacement du flag
encore
BTFS     INTCON , TOIF ; tester si compteur a débordé
GOTO     encore     ; non, attendre débordement
.
XXX      ; poursuivre le programme

```

Supposons que vous désiriez attendre 100 incréments. Il suffit dans ce cas de placer dans tmr0 une valeur telle que 100 incréments plus tard, le TMR0 déborde.

Exemple :

```

MOVLW    156           ; charger 156
MOVWF    TMRO         ; initialiser TMRO
BCF      INTCON,TOIF  ; effacement du drapeau
encore
BTFSS    INTCON,TOIF  ; tester si compteur a débordé
GOTO     encore       ; non, attendre débordement
.
XXX      ; oui, poursuivre : 100 évènements écoulés (256-156=100)
    
```

8.2 Configuration des modes de fonctionnement du module TIMER

La figure ci-dessous montre le diagramme du bloc fonctionnel du timer 0

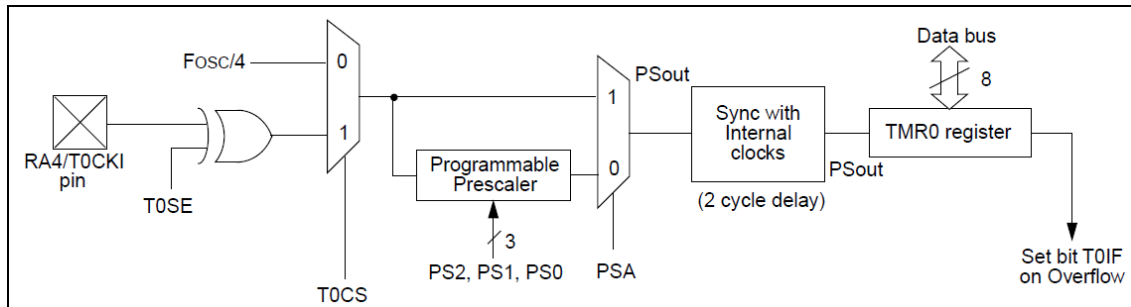


Figure 7.8. Diagramme du bloc fonctionnel du Timer0

Le mode temporisateur est sélectionné en positionnant le bit T0CS à 0 (TOC situé à OPTION_REG). En mode temporisateur, le registre TMR0 s’incrémentera chaque cycle horloge (sans prédiviseur).

Le mode compteur est sélectionné en positionnant le bit T0CS à 1. Dans ce mode, le registre TMR0 s’incrémentera à chaque front montant ou descendant de la broche RA4/T0CKI. Le front d’incrémentaion est déterminé par le bit T0SE du registre OPTION_REG.

- **T0SE = 0** : incrémentaion du TMR0 sur front montant.
- **T0SE = 1** : incrémentaion du TMR0 sur front descendant.

Le pré-diviseur est partagé entre le module Timer0 et le Watchdog Timer (chien de garde). L’affectation du pré-diviseur est contrôlée par programmation, par le bit de contrôle PSA du registre OPTION_REG<3>.

- **PSA = 0** : Le pré-diviseur est affecté au module Timer0.
- **PSA = 1** : Le pré-diviseur est affecté au chien de garde (WDT).

Le taux de la pré-division est déterminé par le tableau ci-dessous :

PS2 PS1 PS0	Taux de division TMR0	Taux de division WDT
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

Exemple :

Supposons maintenant que la fréquence de l’horloge externe est de 4MHz (l’oscillateur doté d’un quartz de 4MHz), dans ce cas fréquence du cycle interne est :

$$F_{cycle} = \frac{F_{osc}}{4} = \frac{4 \cdot 10^6}{4} = 1 \text{ MHz.}$$

Donc la période du cycle interne $T_{cycle} = 1 \mu\text{s}$.

Si nous utilisons le timer0 dans sa fonction timer et en mode interruption. Nous aurons donc une interruption toutes les 256µs.

Si on désire clignoter une LED à une fréquence de 1Hz, nous aurons besoin d'une temporisation de 500ms. Pour ce faire nous disposons d'un pré-diviseur pour diminuer la base de temps du timer.

- Le pré-diviseur permet d'incrémenter le registre timer0 plus lentement, ce qui d'augmenter le temps de production de l'interruption.

$$\frac{T}{2} = 500 \text{ ms} = 500 \times 1000 = 500000 \mu\text{s}$$

PS2 PS1 PS0	Taux de division TMRO	Taux de division WTD
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

9 Gestions des interruptions de PIC16F84

Le PIC16F84 dispose de **4 sources d'interruptions** possibles. Les événements susceptibles de déclencher une interruption sont les suivants :

- **RB0/INT** : Une interruption peut être générée lorsque, la pin RB0, encore appelée **INTerrupt** pin, étant configurée en entrée, le niveau qui est appliqué est modifié.
- **TMRO** : Le module TIMER0, puisque le débordement du registre TMR0 (lorsque le contenu du TMR0 passe de FFH à 00H) une interruption peut être générée.
- **PORTB** : Une interruption peut être générée lors du changement d'un niveau sur une des broches RB4 à RB7. L'interruption sera effective pour les 4 broches ou pour aucune.
- **EEPROM** : cette interruption peut être générée lorsque l'écriture dans une case EEPROM interne est terminée.

9.1 Validation des interruptions des périphériques

Pour valider une ou plusieurs interruptions, il faut commencer par autoriser les interruptions de façon globale, en positionnant le bit **GIE** (**Global Interrupt Enable** de **INTCON<7>**) à **1**. Puis, on autorise les interruptions qui nous intéressent seulement, en positionnant leurs bits de validation à **'1'**.

Lorsqu'un périphérique demande une interruption, le processeur appelle la routine de service d'interruption appropriée **ISR** (en anglais **Interrupt Service Routine**) qui détermine comment gérer l'interruption.

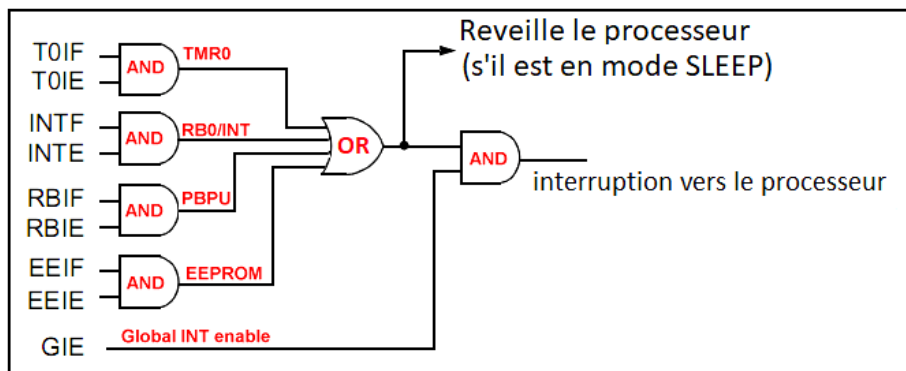


Figure 7.9 Schéma de gestion des interruptions du PIC16F84

- Le bit **INTE** (**INTCON<4>**) : pour activer l'interruption sur la broche RB0/INT
- Le bit **RBIE** (**INTCON<3>**) : pour activer l'interruption "RB"
- Le bit **TOIE** (**INTCON<5>**) : pour activer l'interruption timer 0
- Le bit **EEIE** (**INTCON<6>**) : pour activer l'interruption de fin d'écriture de l'EEPROM

Avant de valider une interruption, il faut effacer (mettre à '0') le drapeau qui signale cette interruption.

- Le bit **INTF** (**INTCON<1>**) : pour signaler une interruption sur la broche RB0/INT.
- Le bit **RBIF** (**INTCON<0>**) : pour signaler une interruption sur RB4..7.
- Le bit **TOIF** (**INTCON<2>**) : pour signaler l'interruption du timer 0.
- Le bit **EEIF** (**EECON1<4>**) : pour signaler l'interruption de fin d'écriture de l'EEPROM.

	7	6	5	4	3	2	1	0
Registre INTCON (Bank0)	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
	7	6	5	4	3	2	1	0
Registre EECON1 (Bank1)	-	-	-	EEIF	WRERR	WREN	WR	RD

Figure7. Registre responsable de la gestion des interruptions du PIC16F84

9.2 Prise en charge d'une interruption par le microcontrôleur

Quand une interruption survient, le programme principal est interrompu

1. Le processeur finit l'exécution de l'instruction en cours du programme principal, Empile la valeur du registre PC (Program Counter) de retour dans la pile.
2. Le processeur se branche à l'adresse '0004H' de la mémoire de programme (autrement dit, le vecteur d'interruption est situé à l'adresse H'0004').
3. Le programmeur doit sauvegarder les registres STATUS et W dans des registres temporaires dans la RAM.
4. Le programme appelle le sous-programme d'interruption (ISR) par l'instruction **CALL**.
5. Une fois l'ISR terminé, le programmeur doit effacer le drapeau qui a signalé l'interruption.
6. Le programmeur doit restituer les registres STATUS et W depuis la RAM.
7. Le programmeur annonce la fin de l'interruption par l'instruction **RETFIE** (Interrupt return), dès que cette instruction est exécutée, le contenu du registre PC est restauré à partir de la pile et le microprocesseur reprend l'exécution du programme qu'il avait interrompu.

La figure ci-dessous présente les différentes étapes qui permettent l'exécution de sous-programme d'interruption.

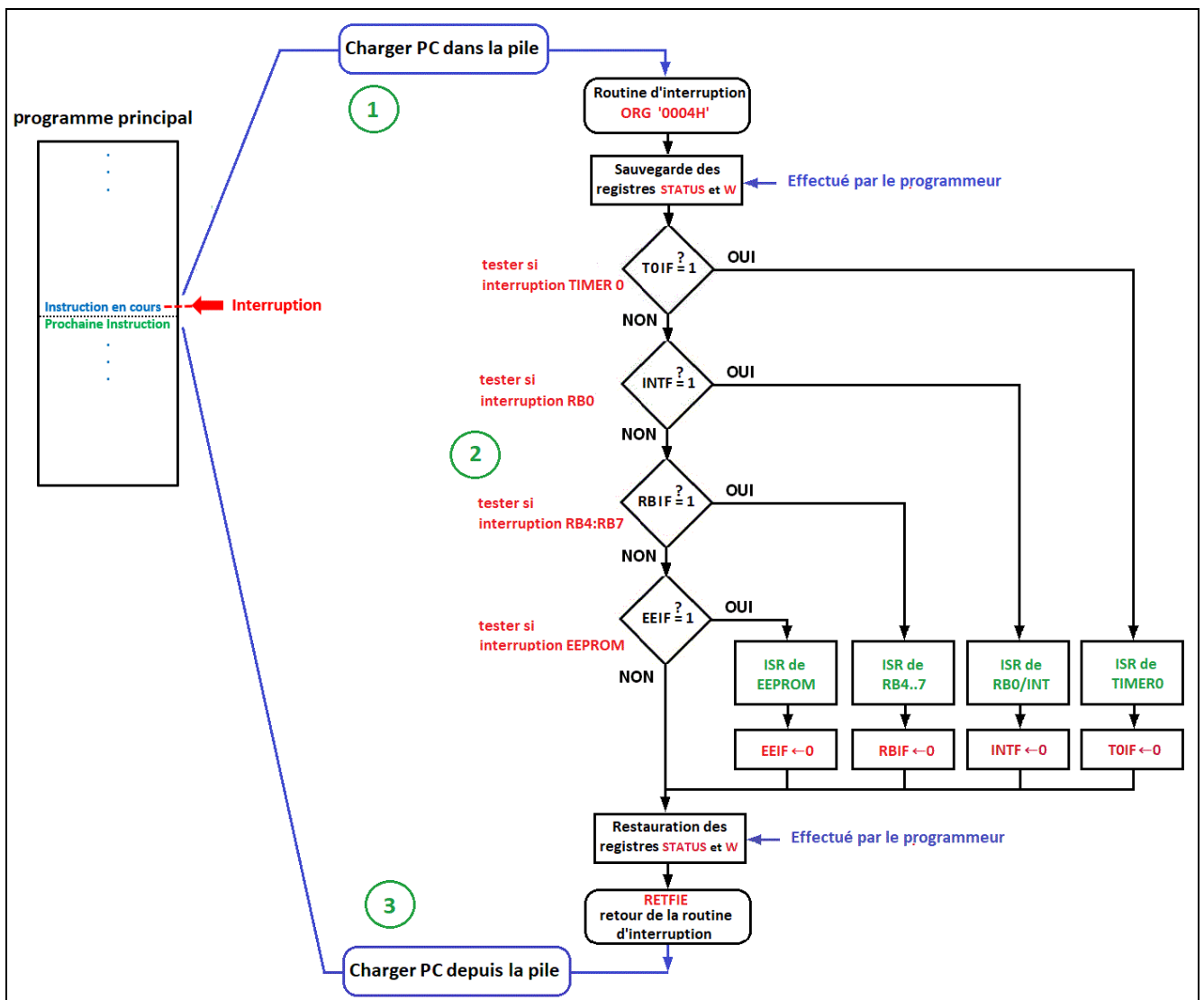


Figure 7.10. Organigramme de traitement des interruptions du PIC16F84

9.3 Le programme assembleur ci-dessous prend en charge l'exécution des interruptions du PIC16F84

```

ORG 0x004 ; adresse de vecteur d'interruption
;Sauvegarde des registre W et STATUS
movwf w_temp ; sauver registre W
swapf STATUS,w ; swap status avec résultat dans w
movwf status_temp ; sauver status swappé

; TESTER SI INTERRUPTION TIMER 0
btfsc INTCON,TOIE ; tester si interrupt timer autorisée
btfss INTCON,TOIF ; oui, tester si interrupt timer en cours
goto test_RB0 ; non test suivant
call ISR_TIMER0 ; oui, traiter interrupt timer
bcf INTCON,TOIF ; effacer flag interrupt timer
goto restore_registre ; et fin d'interruption

; TESTER SI INTERRUPTION RB0
test_RB0
btfsc INTCON,INTE ; tester si interrupt RB0 autorisée
btfss INTCON,INTF ; oui, tester si interrupt RB0 en cours
goto test_RB4_RB7 ; non sauter au test suivant
call ISR_RB0 ; oui, traiter interrupt RB0
bcf INTCON,INTF ; effacer flag interrupt RB0
goto restore_registre ; et fin d'interruption

; TESTER SI INTERRUPTION RB4..RB7
test_RB4_RB7
btfsc INTCON,RBIE ; tester si interrupt RB4/7 autorisée
btfss INTCON,RBIF ; oui, tester si interrupt RB4/7 en cours
goto test_EEPROM ; non sauter
call ISR_RB4_7 ; oui, traiter interrupt RB4/7
bcf INTCON,RBIF ; effacer flag interrupt RB4/7
goto restore_registre ; et fin d'interrupt

; TESTER SI INTERRUPTION DE FIN D'ECRITURE EEPROM
test_EEPROM
bsf STATUS,RP0 ; passer banquel
btfsc INTCON,EEIE ; tester si interrupt EEPROM autorisée
btfss EECON1,EEIF ; oui, tester si interrupt EEPROM
goto restore_registre ; Restaurer les registres
call ISR_EEPROM ; traiter interruption eeprom

restore_registre
swapf status_temp,w ; swap ancien status, résultat dans w
movwf STATUS ; restaurer status
swapf w_temp,f ; restaurer le registre W
swapf w_temp,w
retfie ; return from interrupt

; les sous-programmes de traitement d'interruption pour chaque periphérique
ISR_TIMER0
return ; fin d'interruption timer

ISR_RB0
return ; fin d'interruption RB0/INT

ISR_RB4_7
return ; fin d'interruption RB4/RB7

ISR_EEPROM
bcf EECON1,EEIF ; Effacer le flag d'interruption
bcf STATUS,RP0 ; passer en banque 0
return ; fin d'interruption eeprom

```

Remarques

- Une interruption ne peut pas interrompre une routine d'interruption
- Si une autre interruption survient pendant l'exécution de la routine d'interruption, elle est ignorée. Elle sera prise en compte à la fin de la routine d'interruption, c'est à dire au moment où le microcontrôleur retourne au programme principal.

9.4 Sauvegarde et restitution du contexte lors de l'interruption

Lors d'un appel de sous-programme d'interruption, seule la valeur du registre PC (Program Counter) de retour est enregistrée sur la pile. Il est impératif de sauvegarder le contenu de l'accumulateur **W** et le registre d'état **STATUS** avant d'exécuter les tâches de sous-programme d'interruption (SPI). Après exécution du SPI, il faut restaurer les valeurs des registres W et STATUS. On peut définir ses taches par les points suivant :

- 1) Stocke le registre W dans W_TEMP
- 2) Stocke le registre STATUS dans STATUS_TEMP.
- 3) Exécute le code de routine d'interruption de service.
- 4) Restaure du registre STATUS.
- 5) Restaure le registre W.

Exemple : programme assembleur qui permet la sauvegarde du STATUS and W registre dans la RAM

```

MOVWF W_TEMP      ; W → W_TEMP.
SWAPF STATUS, W   ; échanger le registre STATUS qui va être sauvé dans W.
MOVWF STATUS_TEMP ; sauvegarde STATUS au STATUS_TEMP registre.
:
:                 ; Service de l'interruption
:
:
:
SWAPF STATUS_TEMP, W ; échanger le registre STATUS_TEMP qui va être sauvé dans W
MOVWF STATUS         ; transfère W dans STATUS
SWAPF W_TEMP, F     ; échanger le registre W_TEMP avec lui-même.
SWAPF W_TEMP, W     ; échanger le registre W_TEMP avec me registre W.
    
```

Sauvegarde du contexte

Restitution du contexte

NB : on utilise l'instruction SWAP puisqu'elle effectue un transfert sans affecté les drapeaux du registre d'état STATUS.

Exercice 1 :

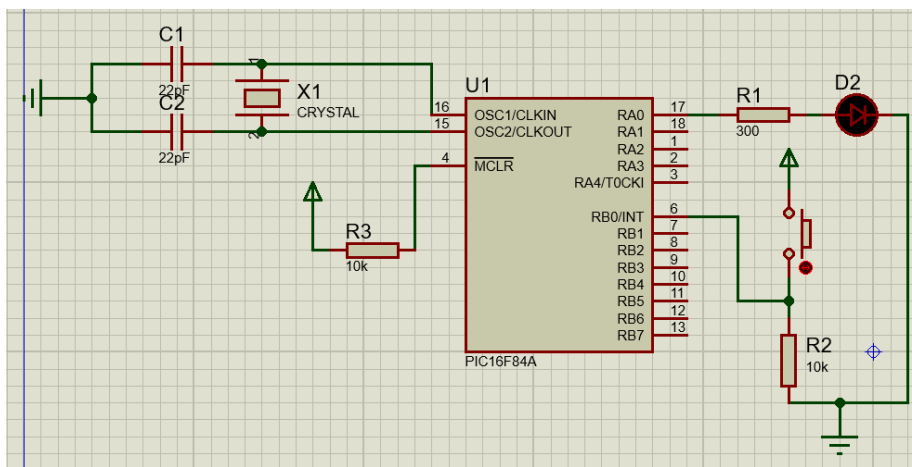


Figure 7.11. Schéma du montage

Soit le schéma ci-dessus, contient un bouton poussoir placé à l'entrée de la broche RB0 du PIC16F84. Et une LED liée à la broche RA0.

Ecrire un programme qui permet d'allumer la LED si le bouton est enfoncé.

Solution de l'exercice 1 :

Tout d'abord il faut configurer la direction des broches du microcontrôleur :

- RBO en entrée
- RA0 en sortie

```

LIST    p=16F84A                ; Définition de processeur
#include <p16F84A.inc>           ; Définitions des constantes

    __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
;=====
;      ASSIGNATIONS
;=====
OPTIONVAL    EQU    H'0000'      ; Valeur registre option

#DEFINE led      PORTA,0        ; Led
#DEFINE bouton  PORTB,0        ; bouton-poussoir
;=====
;      DECLARATIONS DES VARIABLES DU PROGRAMME
;=====
    CBLOCK 0x00C                ; début de la zone de donnée dans la RAM

    cmpt1 : 1                    ; déclaration de la variable mémoire cmpt1

    ENDC                        ; Fin de la zone
;=====
;      DEMARRAGE SUR RESET
;=====
    org    0x000                ; Adresse de départ après reset
    goto  init                    ; Adresse 0: initialiser

init
    clrf   PORTA
    clrf   PORTB                ; sorties portB à 0

    bsf    STATUS,RPO           ; sélectionner banque 1
    movlw  OPTIONVAL            ; charger la configuration du registre OPTION
    movwf  OPTION_REG          ; initialiser registre le option
    movlw  B'11111111'         ; configuration du PORTB en entrée
    movwf  TRISB
    clrf   TRISA
    bcf    STATUS,RPO           ; sélectionner banque 1
    goto  start                  ; sauter au programme principal
;=====
;      PROGRAMME PRINCIPAL
;=====
start
    btfss bouton
    goto  eteindre
    bsf    led
    goto  start

eteindre
    bcf    led
    goto  start                ; boucler
    END                        ; directive de fin de programme

```

10 Mode sommeil (mode SLEEP)

Ce mode est utilisé pour **minimiser la consommation d'énergie** dans les applications alimentées par batterie., lorsqu'il est mis en sommeil, le programme s'arrête jusqu'à son réveil. La mise en sommeil s'effectue grâce à l'instruction **SLEEP**.

11 Jeu d'instructions

Nous présentons l'ensemble des 35 instructions du PIC16F84 codées sur 14 bits. Vous trouverez dans le tableau les mnémoniques des instructions et une brève explication de ce que fait l'instruction.

Les opérandes peuvent être de plusieurs types :

Tableau 7.2. Jeu d'instructions du PIC16F84

Champ	Description
f	Adresse mémoire de registres (register file address) de 00 à 7F
W	Working register (registre de travail) accumulateur
b	Bit address within an 8-bit file register (0 to 7)
d	Selection de destination d = 0: sauvegarde du résultat dans W d = 1: sauvegarde du résultat dans f.
k	Champ littéral (8, ou 11 bits) valeur constante
TOS	Top of Stack (le sommet de la pile)
L	littéral signifie en valeur immédiate (constante)
()	Le contenu
→	Assigner à
< >	Pour désigner un ou les bits d'un registre
PC	PC compteur programme (pointeur d'instruction)

11.1 Les instructions de transfert de données

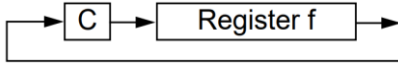
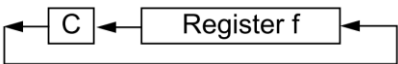
N°	Mnemonique	Description	Drapeau affectés
1	MOVF f,d	Transférer le contenu de f dans (f ou W) Si d=0, (f) → W Si d=1, (f) → f	Z
2	MOVWF f	Transférer le contenu de W dans f (W) → f	aucun
3	MOVLW k	Charger la valeur (littéral : constante) de k dans W , $0 \leq k \leq 255$ k → W	aucun
4	SWAPF f, d	échange de groupes 4-bit de f Si d=0, (f<3:0>) → W<7:4> et (f<7:4>) → W<3:0> Si d=1, (f<3:0>) → f<7:4> et (f<7:4>) → f<3:0>	aucun

11.2 Les instructions arithmétiques

N°	Mnemonic	Description	Drapeau affectés
5	ADDWF f, d	Additionne W avec F Si d=0, (W) + (f) → (W) Si d=1, (W) + (f) → (f)	C, DC, Z
6	ADDLW k	Additionner W et k , 0 ≤ k ≤ 255 constante (literal) Si d=0, k + (W) → W	C, DC, Z
7	INCF f, d	Incrément f Si d=0, (f) + 1 → W Si d=1, (f) + 1 → f	Z
8	SUBWF f, d	Soustraire W de f Si d=0, (f) - (W) → W Si d=1, (f) - (W) → f	Z
9	SUBLW k	Soustraire W de K , 0 ≤ k ≤ 255 constante (literal) Si d=0, k - (W) → W Si d=1, k - (W) → f	C, DC, Z
10	DECF f, d	Décrémenter f Si d=0, (f) - 1 → W Si d=1, (f) - 1 → f	Z

11.3 Les instructions logiques

N°	Mnemonic	Description	Drapeau affectés
11	ANDWF f, d	W and F Si d=0, (W).AND. (f) → (W) Si d=1, (W).AND. (f) → (f)	Z
12	ANDLW k	K and W , 0 ≤ k ≤ 255 (W).AND. (k) → W Si d=0, (W).AND. k → (W) Si d=1, (W).AND. k → (f)	Z
13	IORWF f, d	W OR f (W).OR. (k) → W Si d=0, (W).OR. k → (W) Si d=1, (W).OR. k → (f)	Z
14	IORLW k	K OR W , 0 ≤ k ≤ 255 (W)OR k → W	Z
15	XORWF f, d	W XOR F Si d=0, (W).XOR. (f) → (W) Si d=1, (W).XOR. (f) → (f)	Z
16	XORLW k	W XOR k , 0 ≤ k ≤ 255 Si d=0, (W).XOR. k → (W) Si d=1, (W).XOR. k → (f)	Z
17	COMF f, d	Complement à 1 du f Si d = 0, Complement à 1 de (f) → W Si d = 1, Complement à 1 de (f) → f	Z
18	CLRW	Mettre W à 0 00h → W 1 → Z	Z
19	CLRF f	Mettre f à 0 00h → f 1 → Z	Z

20	BCF f, b	Mettre à '0' le bit de position b de f 0 → f	Aucun
21	BSF f, b	Mettre à '1' le bit de position b de f 1 → f	Aucun
22	RRF f, d	Rotation à droite à travers le CF d'une position de f Si d=0, f après rotation → W Si d=1, f après rotation → f 	C
23	RLF f, d	Rotation à gauche à travers le CF d'une position de f Si d=0, f après rotation → W Si d=1, f après rotation → f 	C

11.4 Les instructions de branchement

N°	Mnemonique	Description	Drapeau affectés
24	DECFSZ f, d	Decrémenter f et sauter si zero Si d=0, (f) - 1 → W; sauter si resultat = 0 Si d=1, (f) - 1 → f; sauter si resultat = 0	Aucun
25	INCFSZ f, d	Incrémenter f et sauter si zero Si d=0, (f) + 1 → W; sauter si resultat = 0 Si d=1, (f) + 1 → f; sauter si resultat = 0	Aucun
26	BTFSC f, b	Tester le bit de f, sauter si '0' Sauter Si (f) = 0	Aucun
27	BTFSS f, b	Tester le bit de f, sauter si '1' Sauter Si (f) = 1	Aucun
28	CALL k	Sauvegarder l'adresse de retour, charge PC avec k (PC)+ 1 → TOS k → PC<10 :0> (PCLATH<4 :3>) → PC<12 :11>	Aucun
29	GOTO k	saut à l'adresse k (9 bits!)	Aucun
30	RETFIE	retour d'interruption	Aucun
31	RETURN	retour de sous-programme	Aucun
32	RETLW k	Positionne W à k et retour	Aucun

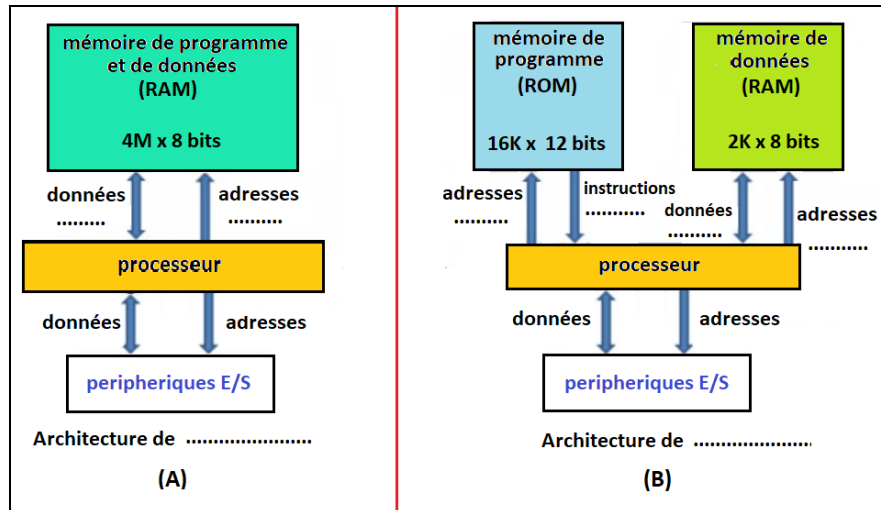
11.5 Les instructions de contrôle

N°	Mnemonique	Description	Drapeau affectés
33	NOP	Pas d'opération	Aucun
34	SLEEP	arrête le processeur	Aucun
35	CLRWDT	Reset du timer watchdog	Aucun

12 Exercices

Exercice 01 :

1. Déterminer de quel type d'architecture est conçu chaque schéma bloc.
2. Déterminer la taille des bus (adresses, données, instructions) sur chaque schéma bloc.



3. Où est-ce qu'on peut trouver les microcontrôleurs ?
4. Est-ce que tous les microcontrôleurs doivent être obligatoirement d'architecture Harvard.
5. Citez 6 composants qu'on peut trouver dans un microcontrôleur.

Exercice 02 :

Questions de cours

1. Quel type d'architecture le PIC16F84 a-t-il (RISC/CISC). Donnez le nombre total de ses instructions.
2. Quelle est la taille en bits, des instructions et des données mémoire du PIC16F84 ?
3. Quels sont les circuits qui sont indispensables au fonctionnement du microcontrôleur.
4. Si l'horloge externe du microcontrôleur PIC16F84 est dotée d'un quartz de 8Mhz. Déterminer le temps de cycle d'une instruction.
5. De quel type est la pile du PIC16F84 (FIFO, LIFO...etc.), et combien de niveau a-t-elle ?
6. Quelle est la taille de la mémoire programme du PIC16F84.
7. Quel est le nombre d'interruptions périphériques qui peuvent être gérées par le PIC16F84. Citez-les ?
8. Quel bit est positionné à '1' est utilisé pour autoriser toutes les interruptions du PIC16F84.
9. Si l'interruption TMR0 est autorisée (TOIE =1), Déterminer à quel moment l'interruption survient-elle ?
10. Quelle instruction annonce la fin de sous-programme d'interruption et oblige le processeur à retourner au programme principal.

Exercice 03 :

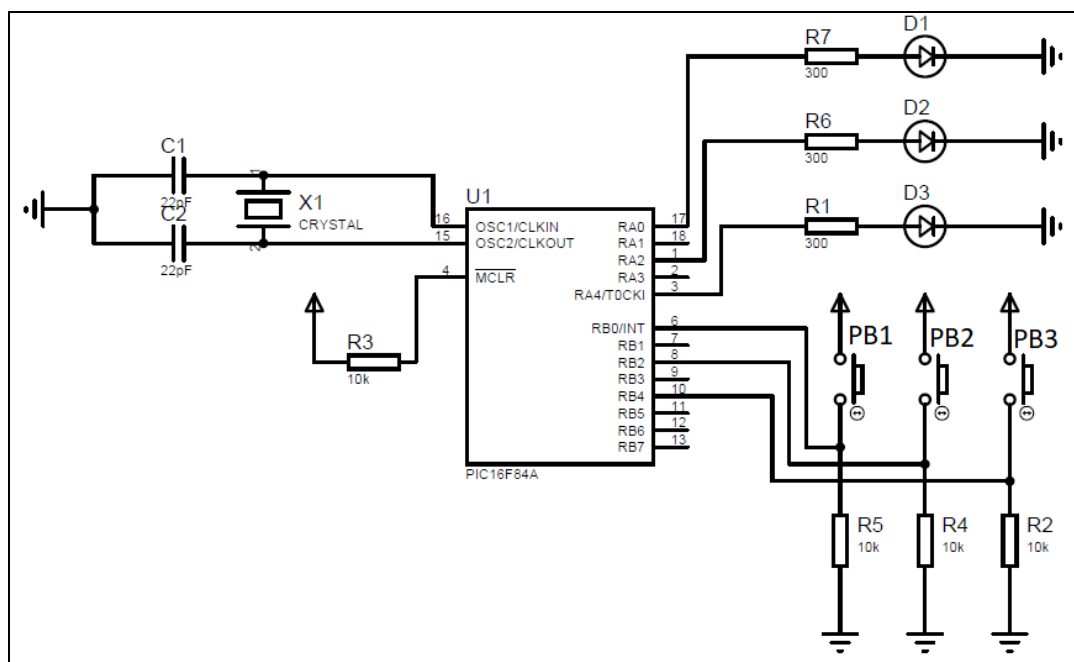
Répondez par (vrai) ou (faux) aux affirmations suivantes, et corrigez les affirmations fausses

1. Dans une architecture Harvard, le processeur peut facilement lire/écrire des données et accéder aux instructions à tout moment.
2. Lors de la mise sous tension du PIC16F84, le processeur pointe vers l'adresse 0000H de la mémoire du programme.
3. Un niveau de tension haut sur la broche MCLR provoque un reset du PIC16F84.
4. Si un périphérique provoque une interruption, le processeur se positionne à l'adresse 0004H.

5. Les registres TRISA et TRISB déterminent respectivement le sens du PORTA et du PORTB.
6. Le mode SLEEP est utilisé pour minimiser la consommation d'énergie dans les applications alimentées par batterie.
7. Une interruption sur la broche RB0/INT peut faire sortir le PIC du mode SLEEP (réveiller le PIC).
8. Dès la mise sous tension du PIC16F84 le registre TMRO s'incrémente automatiquement.
9. La pile du PIC16F84 permet la sauvegarde de l'adresse du PC de retour lors des appels des sous-programmes (CALL) ou lorsqu'une interruption survient.
10. Le programmeur doit effacer le drapeau qui a signalé l'interruption après l'avoir traitée.
11. L'instruction RETFIE permet de restaurer le contenu du registre PC à partir de la pile pour permettre au processeur de reprendre l'exécution du programme qu'il avait interrompu.
12. Le PIC16F84 peut sauvegarder des données mémoire en pile.
13. Dans une architecture de Van Neumann, la vitesse d'exécution est plus rapide car le processeur récupère simultanément les données et les instructions.

Exercice 04 :

Soit le schéma ci-dessous, composé de 3 boutons poussoirs PB1, PB2 et PB3 liés respectivement à RB0, RB2 et RB4, et de trois LEDs D1, D2 et D3 liées respectivement à RA0, RA2 et RA4. L'ensemble est géré par un microcontrôleur PIC16F84.



Nous allons décrire le fonctionnement du montage ci-dessus pour le bouton PB1 et la LED1, mais en fait le raisonnement est le même pour les composants restants.

- Si le bouton PB1 est enfoncé l'entrée RB0 est lue comme un '1' logique, la sortie RA0 doit être mise à un niveau de tension haut $\approx 5\text{Volts}$ ('1' logique) et la LED s'allume.
- Si le bouton PB1 est relâché l'entrée RB0 est lue comme un '0' logique, la sortie RA0 doit être mise à un niveau de tension bas (0 logique) et la LED s'éteint.

1. Ecrire un programme qui permet de lire les boutons poussoirs PB1, PB2 et PB3 et afficher leurs états respectivement sur D1, D2 et D3. Ci-dessous, une proposition de la structure du programme assembleur, qui peut être adoptée par le programmeur.

```
        org    0x000           ; Adresse de départ après reset
        goto  init           ; Adresse 0: initialiser

        init
; INITIALISATION DES PORTS
        .
        .
        .
        goto  start           ; sauter au programme principal
; PROGRAMME PRINCIPAL
        Start
        .
        .
        .
        goto  start           ; boucler
END                          ; directive de fin de programme
```