Université Sultan Moulay Slimane

FST Béni Mellal

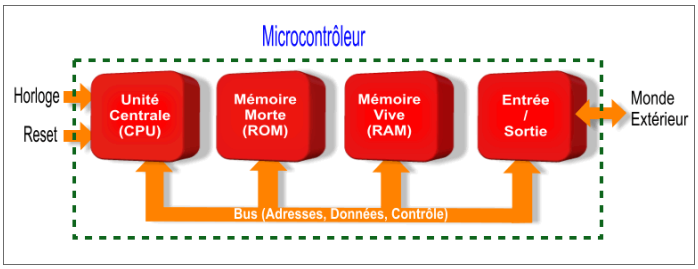
LST Ingénierie Electronique et Télécommunication

Département de Physique

COURS
MICROCONTRÔLEURS PIC

Par : E. AGOURIANE

Introduction aux microcontrôleurs PIC



Un microcontrôleur est un circuit intégré qui contient en interne, c'est-à-dire dans un seul et même boîtier, l'équivalent de la structure complète d'un micro-ordinateur. La figure montre quels sont ces éléments dont voici les fonctions:

- l'unité centrale ou CPU (Central Processing Unit) est le coeur du microcontrôleur. C'est l'équivalent du microprocesseur avec une puissance généralement moindre, la vocation n'étant pas la même. C'est cette unité centrale qui exécute le programme et pilote ainsi tous les autres éléments. Elle dispose généralement de deux connexions avec l'extérieur, une pour son horloge et une pour sa réinitialisation ou reset.

- La mémoire morte ou ROM (Read Only Memory) est une mémoire dont le contenu est conservé même en cas de coupure de courant. Elle contient le programme que va exécuter l'unité centrale. C'est donc elle en fait qui personnalise le circuit, puisque c'est elle qui définit sa fonction.
- La mémoire vive ou RAM (Random Access Memory) est une mémoire dans laquelle l'unité centrale peut lire et écrire à tout instant. Elle est utilisée dans les phases de calcul du programme, pour stocker des résultats intermédiaires, stocker les variables d'une application.

- Les entrées/sorties permettent au microcontrôleur de communiquer avec le monde extérieur. C'est donc là que vont être connectés les claviers, afficheurs, poussoir, moteurs, relais, etc. que va utiliser l'application.

Tous ces éléments sont reliés entre eux par ce que l'on appelle un bus, c'est-à-dire un ensemble de liaisons transportant des adresses, des données et des signaux de contrôle.
La majorité des grands fabricants de circuits intégrés dispose aujourd'hui de plusieurs gammes de microcontrôleurs qui sont performantes (68HC11, ST6, PIC, …)

Les critères principaux que nous devons retenir pour choisir un microcontrôleur sont les suivants:

- le ou les circuits de la famille doivent être facilement disponibles sur le marché;
- le prix des circuits doit être à la portée;
- la programmation de la mémoire morte interne doit être facile ;
- et enfin, les outils de développement doivent être aussi peu coûteux que possible voir même gratuit.

A l'heure actuelle, les circuits qui répondent le mieux à ces critères sont les microcontrôleurs de la famille PIC de Microchip. Comble de chance, ces circuits connaissent actuellement un succès que l'on peut, sans exagérer, qualifier de planétaire et sont très largement utilisés dans l'industrie.

Le PIC (Programmable Interface Controler) est un circuit fabriqué par la société américaine Arizona MICROCHIP Technology. Les PIC sont des composants dits RISC (reduced Instructions set computer) c'est-à-dire composant à jeu d'instructions réduit (à l'opposé on trouve CISC: Complexe Instructions Set Computer). Cela lui confère l'avantage de la rapidité dans l'exécution et l'exécution en un seul cycle machine.
Les familles des PIC sont divisées en 3 grandes familles:

La famille base line : mot de 12 bits

La famille Mid-range : mot de 14 bits

La famille High-End: mot de 16bits

Dans ce cours on va aborder le PIC16F84 qu'est considéré comme l'entrée de gamme de la famille Mid-range.

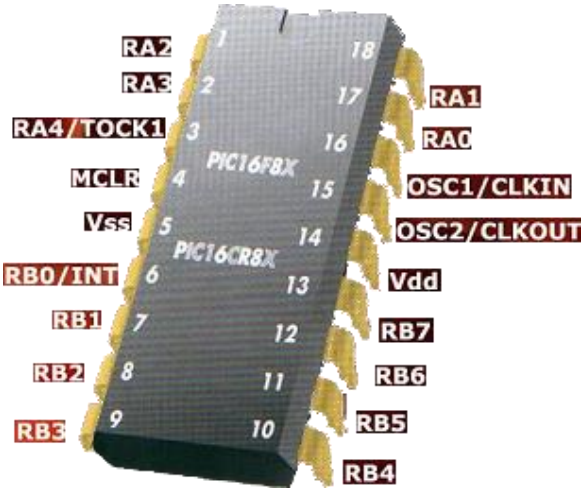
PIC 16F84

1. Introduction

a. Identification

Le 16F84-04 est un boîtier de 18 broches, c'est un PIC de la famille mid-range donc de 14 bits. Le numéro 16 indique la famille. La lettre F indique le type de mémoire ici flash (C: EPROM, CR: ROM). La mémoire flash est programmable et effaçable électriquement.

Les deux chiffres 84 identifient le composant. Et les deux derniers chiffres représentent la fréquence d'horloge maximale en MHz, ici elle est de 4MHz. Les PIC sont des composants statiques, ils fonctionnent même si la fréquence baisse.

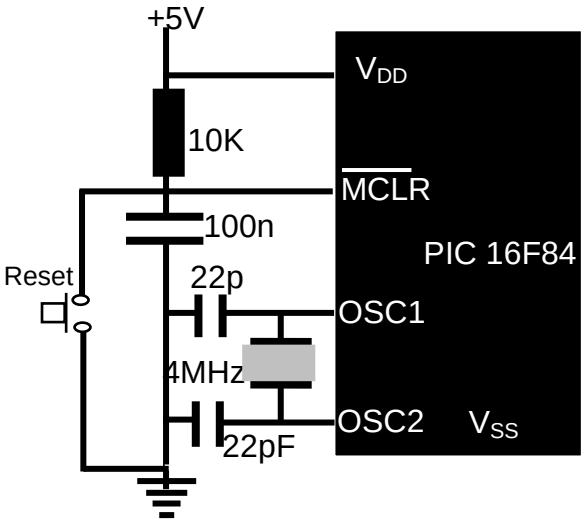


b. Présentation

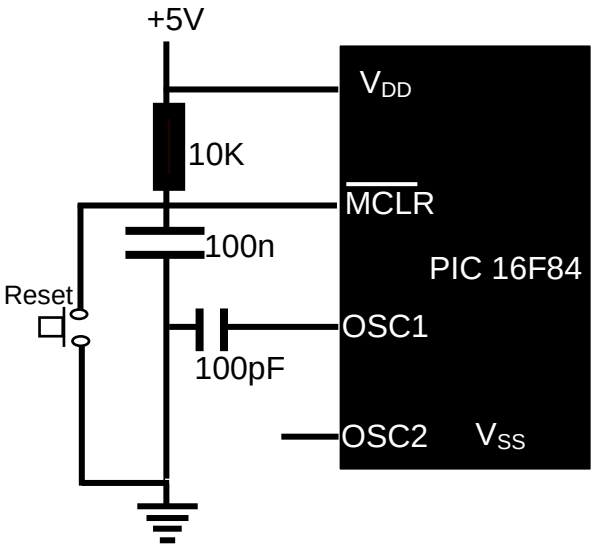
C'est un microcontrôleur 8 bits qui possède les caractéristiques suivantes:
1k mots de 14 bits de mémoire programme (1024 instructions max)
68 Octets de RAM libres
64 Octets D'EEPROM libres
13 Entrées/Sorties
2 Ports A, B
4 Sources d'interruption
1 Timer/Compteur
1 Chien de garde
4 Sources d'oscillateur sélectionnable.
Port A: RA4 RA3 RA2 RA1 RA0
Port B: RB7 RB6 RB5 RB4 RB3 RB2 RB1 RB0
OSC1 et OSC2: Connexion de l'oscillateur
MCLR: Interruption principale pour reset
INT: Entrée interruption sur broche RB0/INT
TOCK1: Horloge externe pour pilotage du Timer

Quelle que soit l'application, le PIC a besoin nécessairement:

- ✓ Une alimentation de 5 volts
- ✓ Un quartz et deux condensateurs pour un pilotage précis, ou une résistance et un condensateur pour une utilisation normale
- ✓ Un condensateur de découplage
- ✓ Un bouton poussoir et une résistance pour mise en place d'une reset manuelle.



Pilotage par Quartz

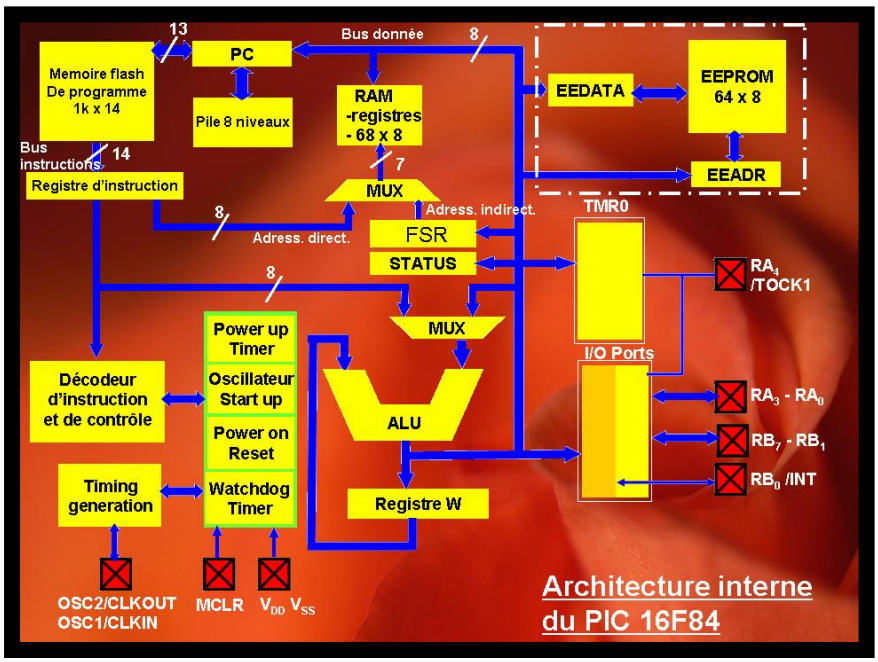


Pilotage par RC

Mode	freq.	OSC1/C1	OSC2/C2
LP	32 kHz 200 kHz	68 - 100 pF 15 - 30 pF	68 - 100 pF 15 - 30 pF
XT	100 kHz 2 MHz 4 MHz	68 - 150 pF 15 - 33 pF 15 - 33 pF	150 - 200 pF 15 - 33 pF 15 - 33 pF
HS	4 MHz 10 MHz	15 - 33 pF 15 - 47 pF	15 - 33 pF 15 - 47 pF

Valeur de capacité avec les résonateurs à quartz.

2. Architecture interne du PIC 16F84



Mémoire programme

En haut à gauche de la figure se trouve le bloc où est stocké le programme. Cette mémoire contient toutes les « instructions » que doit effectuer le microcontrôleur. Pour le PIC 16F84, chaque instruction est codée sur 14 bits et la taille mémoire est de 1 Ko x 14 bits (soit 1 024 instructions à la file). Cette taille reste suffisante pour les petites applications qui nous intéressent. La mémoire programme peut être de différente nature selon le microcontrôleur utilisé. Dans notre cas, la mémoire est de type FLASH/ROM, c'est-à-dire réinscriptible à volonté. Mais il existe aussi des mémoires EPROM qui sont effaçables par UV ainsi que des mémoires OTP qui ne peuvent être programmées qu'une seule fois.

Le Program Counter ou « PC » est un registre qui pointe l'instruction à exécuter et permet donc au programmeur de se déplacer à volonté dans cette mémoire. A la mise sous tension, ce registre est mis à 0 et pointe donc la première case mémoire. Durant l'exécution du programme, chaque instruction pointée transite dans le «registre d'instruction» afin qu'elle puisse être traitée par le «contrôleur et décodeur d'instructions». C'est ce dernier qui gère la suite d'actions internes nécessaires à la bonne exécution de cette instruction.

Unité de calcul – ALU

En bas et au centre, se trouve le cœur du système appelé « ALU » (Unité Arithmétique et Logique). C'est cette partie qui effectue physiquement toutes les actions internes dictées par le Contrôleur et Décodeur d'Instructions. Par exemple, l'ALU peut effectuer une addition, une soustraction ainsi que les opérations logiques telles que « ET », « OU », etc. Pour effectuer toutes ces opérations, l'ALU utilise les données en provenance d'un registre de travail appelé «W» (Work Register) qui est largement utilisé par les instructions du programme.

Mémoire RAM

Pour fonctionner, un programme doit généralement pouvoir stocker temporairement des données. Une zone est spécialement prévue à cet effet : c'est la RAM (Ramdom Acces Memory). Contrairement à la Mémoire Programme, cette dernière s'efface lorsque l'on coupe l'alimentation. On s'aperçoit que la taille de cette mémoire est de 68 x 8 bits. Comme chaque donnée est codée sur 8 bits, il est donc possible de mémoriser 68 données à la file. De plus, la mémoire RAM contient deux autres zones (appelées Bank 0 et Bank 1) réservées aux registres de configuration du système. Nous détaillerons la fonction de ces registres plus loin.

Mémoire de données EEPROM

Une particularité (et aussi un grand avantage) du PIC16F84 est de posséder une mémoire de 64 x 8 bits où l'on peut stocker des données qui ne disparaissent pas lors d'une coupure d'alimentation: c'est la mémoire EEPROM. Généralement cette zone sert à mémoriser des paramètres d'étalonnage ou de configuration. Mais attention, elle possède un inconvénient : le temps d'accès est relativement long. Elle ne convient donc pas pour une utilisation qui demande de la rapidité (calcul, asservissement, etc.).

Pour finir cette description générale, nous allons présenter succinctement les blocs restants.

- Le Status Register ou Registre d'état : ce registre donne plusieurs indications : le résultat d'une opération effectuée par l'ALU (résultat égal zéro par exemple), l'état de l'initialisation du microcontrôleur (reset), etc. Il permet aussi de définir la zone d'accès en mémoire RAM Bank 0 et Bank 1 pour accéder aux registres de configuration (pas d'inquiétude nous verrons cela plus tard).

- Le bloc nommé «8 Level Stack» ou «Pile» : il est utilisé par le microcontrôleur pour gérer le retour des sous programmes et des interruptions.

- Le bloc « TMR0 » : il sert au fonctionnement du TIMER.

- Le bloc I/O ports : il permet d'écrire ou de lire sur les ports A et B.

- Le bloc Timing Génération associé au bloc présent juste sur sa droite gère tous les signaux d'horloges du système.

- Le registre FSR : utilisé pour l'adressage indirect de la mémoire RAM.

- Le bus de donnée : il met en liaison les blocs utilisant des données.

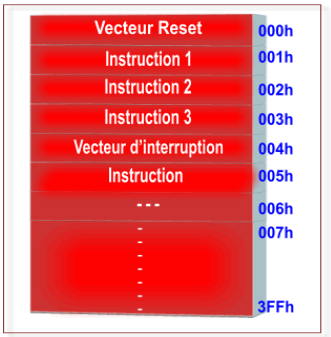
Cette description interne nous a permis d'identifier les blocs principaux de ce microcontrôleur et de mieux comprendre le déroulement d'un programme.

Nous allons maintenant nous intéresser plus particulièrement aux différentes mémoires afin de savoir où et comment stocker un programme, configurer les ports d'entrée/sortie et utiliser la mémoire de données.

3. Organisation de la RAM

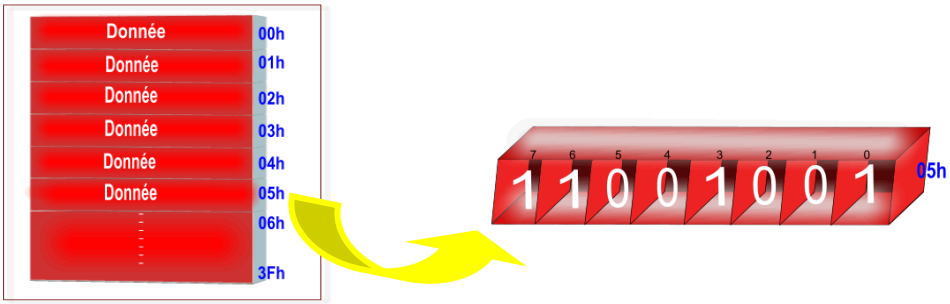
La mémoire est divisée en 3 parties:

a- La mémoire programme: constituée de 1k mots de 14 bits. Il faut 2 octets pour coder 14 bits, chaque instruction est codée sur 1 mot (allant de 000 à 3FF). Les adresses 0 et 4 sont réservées:
000h: réservé exclusivement au vecteur de reset
004h: réservé au vecteur interruption



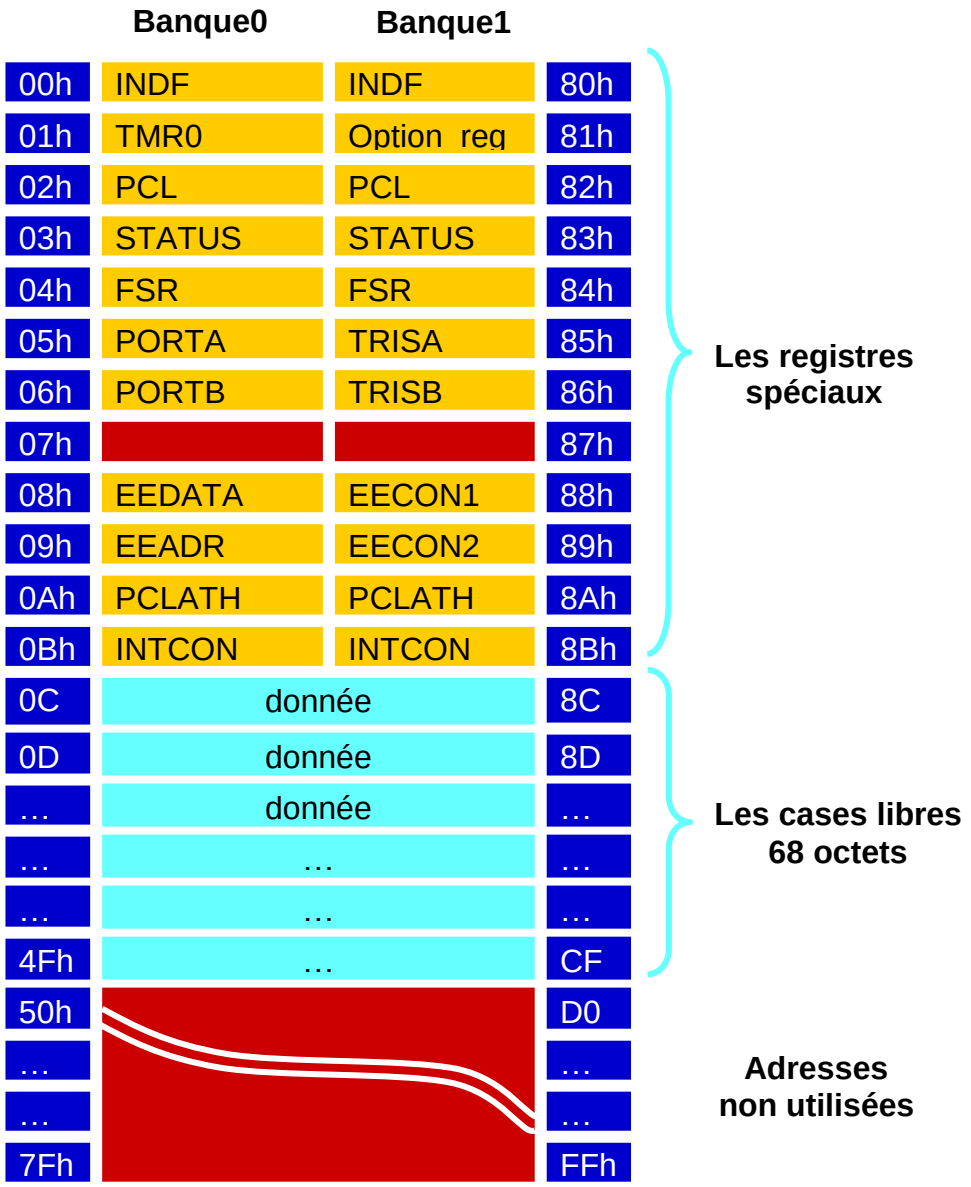
Mémoire programme

b- La mémoire EEPROM: constituée de 64 octets qu'on peut lire ou écrire depuis le pg en exécution, c'est utile pour conserver des paramètres semi permanents (de 0 à 3F) ou des événements survenants au cours de l'exécution du pg.



Mémoire EEPROM L'octet dans un registre

c- La mémoire RAM: constituée de 68 octets. Elle est organisée en deux banques. On y trouve des registres spéciaux et des cases mémoires libres. Deux banques de 68 octets auxquelles on peut accéder soit de la banque 0 ou la banque 1. Les registres spéciaux servent à provoquer un fonctionnement spécial du PIC.



Organisation de la RAM

4. Les registres spéciaux

Les registres spéciaux sont très importants et très utilisés dans le fonctionnement du PIC, ils sont au total 16 registres

1. EEADR: Registre où on écrit l'adresse de la case mémoire EEPROM à laquelle on veut accéder

2. EECON1: Registre de contrôle qui permet de définir ou d'indiquer le mode de fonctionnement de l'EEPROM

	4	3	2	1	0
……	EEIF	WRERR	WREN	WR	RD

RD. read……	A mettre à 1 durant lecture, remis à 0 automatiquement à la fin de la lecture
WR. write…….	A mettre à 1 durant écriture, remis automatiquement à 0 à la fin d'écriture
WREN. write enable……	A 0 il interdit l'écriture en EEPROM
WRERR. write error……	Normalement à 0, passe à 1 pour signaler une erreur (interruption d'écriture)
EEIF. eeprom interrupt flag..	Passe à 1 à la fin d'un cycle écriture d'EEPROM

3. EECON2: registre non physique, utilisé exclusivement dans la procédure d'écriture dans l'EPROM.

4. EEDATA : Registre qui contient la donnée qu'on a cherché ou qu'on veut stocker dans l'EEPROM.

5. FSR : Registre qui permet d'accéder aux mémoires par adressage indirect.

6. INDF : Registre utilisé pour accéder aux données de la RAM par adressage indirect

7. INTCON : Registre qui s'occupe des interruptions

8. OPTION : Registre qui fixe le fonctionnement de l'horloge interne.

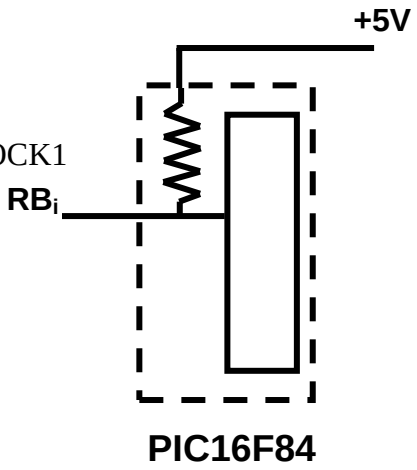
7	6	5	4	3	2	1	0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

RBPU : à 0 il valide les résistances de rappel à 5V pour le port B.
INTEDG : détermine le front de l'interruption sur la broche RB0/INT
1: front ↑ 0: front ↓

TOCS : choix du signal pour piloter le Timer
0: Timer piloté par horloge interne
1: Timer piloté par signal externe via la broche RA4/TOCK1

TOSE : fixe le front d'horloge externe
1: front ↑ 0: front ↓

PSA: affecte un facteur de division
0: pour le Timer
1: pour le chien de garde



PS2,PS1,PS0: définie la valeur du facteur de division.

9. PCL

10. PCLATH } Donnent l'adresse basse et haute du compteur programme (PC)

11. PORTA c'est un port bidirectionnelle de 5 bits (R/W)

7	5	6	4	3	2	1	0
…	…	…	RA4	RA3	RA2	RA1	RA0

Se sont des E/S compatible TTL,
Les broches RA3, RA2, RA1 et RA0 en sortie TOTEM
La broche RA4 en sortie drain ouvert et en entrée trigger de SCHMITT

12. PORTB c'est un port de 8 bits

7	5	6	4	3	2	1	0
RB_7	RB_6	RB_5	RB_4	RB_3	RB_2	RB_1	RB_0

C'est un port bidirectionnel de 8 bits, compatible TTL

13. TRISA

7	5	6	4	3	2	1	0
…	…	…	$TRISA_4$	$TRISA_3$	$TRISA_2$	$TRISA_1$	$TRISA_0$

Configure les bits du port A en entrée ou en sortie.
1: broche en entrée **0:** broche en sortie
Après le reset le port A est configuré en entrée automatiquement.

14. TRISB

7	5	6	4	3	2	1	0
$TRISB_7$	TRISB	TRISB	$TRISB_4$	$TRISB_3$	$TRISB_2$	$TRISB_1$	$TRISB_0$

Même chose que TRISA

15. TMR0

Timer: c'est un registre de comptage. Il peut compter soit:
- ✓ les impulsions sur l'entrée RA4/TOCK1 c'est le mode **compteur**
- ✓ les cycles d'horloge du PIC (compter le temps) c'est le mode **Timer**

16. STATUS
Contient des informations sur l'exécution de l'instruction en cours

7	5	6	4	3	2	1	0
IRP	RP1	RP0	TO	PD	Z	DC	C

- C : carry (retenue)
- DC : retenue entre bits 3 et 4
- Z : zéro (résultat nulle)

- TO: indique dépassement du temps du watchdog
  =1 au démarrage, après CLRWDT ou SLEEP
  =0 dépassement du temps du watchdog
- PD : indique le passage du PIC en mode sommeil
  =1 au démarrage ou par CLRWDT
  =0 par l'instruction SLEEP
- RP0: permet de fixer la banque (0: banque0, 1: banque1)
- RP1: « =0 » non utilisé pour 16F84
- IRP : « =0 » non utilisé pour 16F84

5. Instructions du PIC 16F8XX

Architecture RISC  **jeux d'instructions réduit**

35 instructions

- **Instructions sur les registres: octet**
- **Instructions sur les registres: bit**
- **Instructions de contrôle et de saut**

Les instructions sont codées sur 14 bits

1- Instructions sur les registres: octet

Se sont les instructions qui manipulent les données, elles sont codées comme suit:

- Les 6 bits de poids fort contiennent le code de l'instruction (opcode)
- Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)
- Le bit 7 indique où sera placé le résultat de l'opération:
  Bit 7=0 -> le résultat sera dans le registre de travail (W)
  Bit 7=1 -> le résultat sera dans la même case mémoire RAM (ou le registre)

Instructions orientées octet													
13	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Opcode						**0/1**	**Adresse du registre concerné**						

Exp : ***Movf Reg_temp, W***

2- Instructions sur le bit d'un registre

Se sont les instructions qui manipulent directement les bits d'un registre, elles sont codées comme suit:

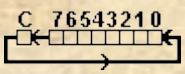
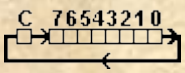
- Les 4bits de poids fort contiennent le code de l'instruction (opcode)
- Les bits 7, 8 et 9 déterminent le bit concerné par l'opération (bit 0 à bit 7)

- Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)

Instructions orientées bit													
13	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Opcode				**Bit concerné**			**Adresse du registre concerné**						

Exp : ***BSF Port_b, 3***

Instructions sur les registres (octets)

mnémonique	Description	cycle	bits
ADDWF f,d	d:=W+f	1	C,DC,Z
ANDWF f,d	d:=W AND f	1	Z
CLRF f	f:=0	1	Z
CLRW	W:=0	1	Z
COMF f,d	d:=NOT(f)	1	Z
DECF f,d	d:=f-1	1	Z
DECFSZ f,d	d:=f-1 ; Skip if Zero	1(2)	.
INCF f,d	d:=f+1	1	Z
INCFSZ f,d	d:=f+1 ; Skip if Zero	1(2)	.
IORWF f,d	d:=W OR f	1	Z
MOVF f,d	d:=f (permet de savoir si f=0 en faisant	1	Z
MOVWF f	W:=f	1	.
NOP	n'effectue aucune opération	1	.
RLF f,d	d=f SHR 1 	1	C
RRF f,d	d=f SHL 1 	1	C
SUBWF f,d	d:= f-W (en complément à 2)	1	C,DC,Z
SWAPF f,d	d:= f[4..7] <--> f[0..3] (inverse les quartets)	1	.
XORWF f,d	d:= W XOR f	1	Z

Instructions sur les registres (bit par bit)

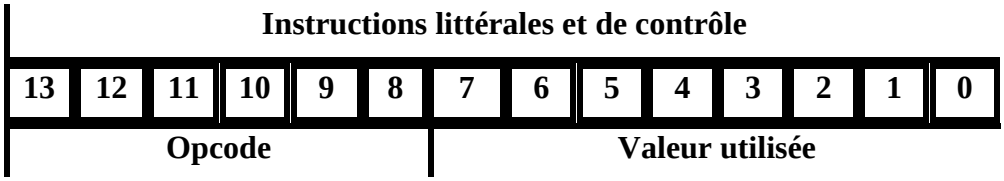
mnémonique	Description	cycle	bits affect
BCF f,b	f[b]:=0 (mets à 0 le bit b de f)	1	.
BSF f,b	f[b]:=1 (mets à 1 le bit b de f)	1	.
BTFSC f,b	teste le bit b de f ; Skip if Clear (0)	1(2)	.
BTFSS f,b	teste le bit b de f ; Skip if Set (1)	1(2)	.

3- Instructions générales de contrôle

Se sont les instructions qui manipulent les données qui sont codées dans l'instruction directement:

- Les 6 bits de poids fort contiennent le code de l'instruction (opcode)

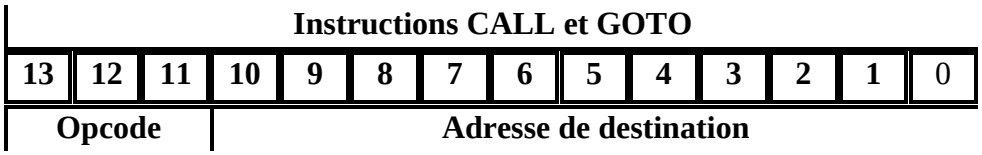
- Les 8 bits de poids faible contiennent la valeur numérique ou littérale qui sera utilisée par l'instruction.



Exp : ***Movlw 01010101b***

4- Instructions de saut et appel

Se sont les instructions qui provoquent une rupture dans la séquence de déroulement du pg qui est écrit dans la mem programme (1k) 10bits



Exp : ***Call sous_P***

Instructions de contrôle et de saut

mnémonique	Description	cycle	bits affect
ADDLW k	W:=W+k	1	C,DC,Z
ANDLW k	W:=W AND k	1	Z
CALL k	appel un sous programme	2	.
CLRWDT	remet à 0 le timer du chien de garde	1	TO,PD
GOTO k	se branche à l'adresse k	2	.
IORLW k	W:=W OR k	1	Z
MOVLW k	W:=k	1	.
RETFIE	fin d'une interruption	2	.
RETLW k	w:=k , puis effectue un retour de sous programme	2	.
RETURN	effectue un retour de sous programme	2	.
SLEEP	circuit en mode sommeil et stoppe l'oscillateur	1	TO,PD
SUBLW k	W:= k-W	1	C,DC,Z
XORLW k	W:=W XOR k	1	Z

ADDLW

Additionner le registre W et une valeur immédiate, la somme est stockée dans W

(W) + v -> (W)

Syntaxe :

Addlw 0x0F *W + Ox0F le résultat est stocké dans W*

SUBLW

Soustraction entre une valeur et le registre W

k - (W) → (W)

Syntaxe :

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

(Complément à 2)

ANDLW

Opération " ET " entre le contenu du registre W et une constante c

(W) AND c → (W)

Exemple :

Port_A equ 05h on affecte 05 à la variable PortA utilisée ci - après

Movf Port_A, W on met le contenu du port A dans le registre W

Andlw 55h on masque un bit sur 2 (55 h = 0101 0101)

IORLW

Opération logique "OU" entre le registre W et une constante c

(W) OR c → (W)

Exemple :

Dans cet exemple on charge une valeur dans le registre W puis on exécute un ou logique avec valeur immédiate

Movlw 01010101b *on charge 01010101b dans le registre W*

Iorlw 11110000b *après le OU logique W = 11110101b*

ADDWF

Additionner le registre W et (f) , la somme est stockée en (d)

(W) + (f) -> (d)

f est l'emplacement mémoire d 'un registre

Exemple :

Movf valeur1, W on met la valeur1 dans le registre W

Addwf valeur2, W valeur 1 + valeur 2 stocké dans W

SUBWF

Soustraire le contenu du registre W du contenu du registre f résultat dans d

(f) - (W) -> (d)

Exemple :

Dans cet exemple on charge une valeur dans le registre W puis on la soustraie avec le contenu d'un registre temporaire, le résultat est dirigé vers le registre temporaire.. (Méthode du complément à 2)

Movlw 00001010b on charge 0Ah dans le registre W

Movwf Reg_temp on met W dans le registre temporaire

Movlw 00000001b on charge 01h dans le registre W

Subwf Reg_temp, f on soustraie avec W le registre temporaire= 00001001b

ANDWF

Opération " ET " entre le contenu du registre W et f le résultat est en d

(W) AND (f) → (d)

Exemple :

R_etat equ 03h *on affecte 03 à la variable R_etat utilisée ci - après*

Movlw 00000111b *on met le contenu 00000111 dans le registre W*

Andwf R_etat, F *on masque et on configure le registre d' état*

IORWF
Opération logique "OU" entre le registre W et f résultat dans d
(W) OR (f) → (d)
Exemple :
Dans cet exemple on charge une valeur dans le registre W puis on exécute un ou logique avec le contenu d'un registre temporaire, le résultat est dirigé vers le registre temporaire..
Movlw 01010101b on charge 55h dans le registre W
Movwf Reg_temp on met W dans le registre temporaire
Movlw 00001111b on charge 0Fh dans le registre W
Iorwf Reg_temp, f ou logique avec W le contenu du registre temporaire= 01011111b

INCF
Incrémente f et range le résultat dans d
(f) + 1 → (d)
Exemple : Dans cet exemple on charge une valeur dans un registre ici 55h puis on incrémente cette valeur qui devient alors 56h.
Movlw 01010101b on charge 55h dans le registre W
Movwf Reg_temp on met W dans le registre temporaire
Incf Reg_temp, f on complémente le contenu du registre temporaire

INCFSZ
Incrémente f et sauter l'instruction suivante si (f) = 0
(f) + 1 → (d) et saute si le résultat = 0
Syntaxe:
Incfsz reg_temp,1 reg_temp est incrémenté, si reg=0 on saute goto sinon exécuter instruction suivante
Goto loop
--

BSF
Positionne le bit "b" de f à 1
1 → (f{b})
Exemple :
BSF Port_b, 3 on met à 1 le bit 3 du port B

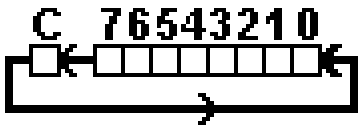
XORLW
Opération logique XOR (ou exclusif) entre le contenu de W et une constante c
(W) XOR c → (W)
Exemple :
Dans cet exemple on charge une valeur dans le registre W puis on exécute un ou exclusif avec une valeur immédiate, le résultat est dirigé vers le registre W
Movlw 01010101b on charge 55h dans le registre W
Xorlw 00111100b ou exclusif avec W le contenu du registre W = 01101001b

XORWF
Opération logique XOR (ou exclusif) entre le contenu de W et f le résultat en d
(W) XOR (f) → (d)
Exemple : Dans cet exemple on charge une valeur dans le registre W puis on exécute un ou exclusif avec le contenu d'un registre temporaire, le résultat est dirigé vers le registre temporaire..
Movlw 01010101b on charge 55h dans le registre W
Movwf Reg_temp on met W dans le registre temporaire
Movlw 00001111b on charge 0Fh dans le registre W
Xorwf Reg_temp, f ou exclusif avec W le contenu du registre temp.= 01011010b

DECF
Décrémente f et range le résultat dans d
(f) - 1 → (d)
Exemple : Dans cet exemple on charge une valeur dans un registre ici 01h puis on décrémente cette valeur qui devient alors 00h (flag Z = 1 dans cet exemple).
Movlw 01h on charge 01h dans le registre W
Movwf Reg_temp on met W dans le registre temporaire
Decf Reg_temp, f on décrémente le contenu du registre temporaire

BCF
Efface le bit spécifié de f
0 → (f{b})
Exemple :
BCF Port_b, 2 on met à 0 le bit 2 du port B

RLF
Rotation à gauche de (f) au travers du bit carry © le résultat va dans d
Carry → (d0)
(f0) → (d1)
(f1) → (d2)
(f2) → (d3)
(f3) → (d4)
(f4) → (d5)
(f5) → (d6)
(f6) → (d7)
(f7) → (Carry)



Syntaxe:
RLF reg_temp,0 *si reg_temp=11100110 et C=0* alors *w=11001100 et C=1*

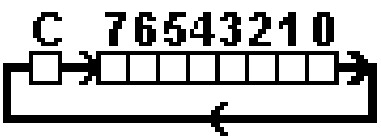
DECFSZ
Décrémente f et saute l'instruction suivante si f = 0
(f) - 1 → (d) et sauter l'instruction suivante si f=0
Syntaxe:
Decfsz reg_temp,1 *on décremente reg_temp, on saute si Z=0 sinon on éxcute instruction suivante*
Goto loop

CALL
Appel du sous programme s
(PC) + 1 → Haut de la pile
s → (PC)
Exemple :
Call sous_P Appel du sous programme sous_P

SLEEP
Mise en veille du microcontrôleur (Power Down mode)
0 → PD
1 → TO
00 → WDT
00 → Prédiviseur de WDT
Syntaxe : SLEEP
La consommation de courant est minimale

CLRF
Efface le contenu de (f)
00 → (f)
Syntaxe : CLRF registre

RRF
Rotation à droite de (f) au travers du bit carry © le résultat va dans d
(f0) → Carry
(f1) → (d0)
(f2) → (d1)
(f3) → (d2)
(f4) → (d3)
(f5) → (d4)
(f6) → (d5)
(f7) → (d6)
Carry → (d7)



Syntaxe:
RRF reg_temp,0 *si reg_temp=11100110 et C=0 alors w=01110011 et C=0*

NOP
Aucune opération
(PC) + 1 → (PC)
Nombre de cycle d'horloge : 1
Indicateurs positionnés : Aucun
Syntaxe : NOP

CLRWDT
Réinitialise le temporisateur du chien de garde
00 → WDT
00 → prédiviseur de WDT
Syntaxe : CLRWDT

CLRW
Efface le contenu du registre W
00 → W
Syntaxe : CLRW

BTFSC
Vérifier l'état du bit b de f et sauter l'instruction suivante si b = 0
"Saute l'instruction si" (f{b}) = 0
Exemple :
Incf reg_temp on incrémente la variable reg_temp
Btfsc reg_temp,1 on vérifie si bit1 du reg_temp =0 on saute goto sinon on excute l'instruction goto
Goto process

MOVF
Copier le registre (f) dans (d)
(f) + 1 → (d)
Exemple : *Dans cet exemple on copie le contenu d'un registre temporaire dans le registre W*
Movf Reg_temp, W on copie Reg_temp dans W

COMF
Stocke en d le complément de f
f → (d)
Exemple : Dans cet exemple on charge une valeur dans un registre ici 55h puis on complémente cette valeur qui devient alors AAh.
Movwf Reg_temp *on met W dans le registre temporaire*
Comf Reg_temp, f *on complémente le contenu du registre temporaire reg_temp=AAh*
Movlw 01010101b *on charge 55h dans le registre W*

SWAPF
Échange de quartets entre le registre (f) et (d)
(f bit 0 à bit 3) -> (d bit 4 à bit 7)
(f bit 4 à bit 7) -> (d bit 0 à bit 3)
Syntaxe:
Swapf reg_temp,0 *si reg_temp=0xA5 alors w=0x5A*

GOTO
Branchement inconditionnel à l'adresse a
a → PC
Exemple :
Goto fin *Va à l'étiquette nommée "fin"*
-
-
fin : *Etiquette fin*

RETFIE
Retour d'interruption
1 → GIE Haut de pile → (PC)
Syntaxe : RETFIE *retour d'interruption*

RETURN
Retour de sous programme
Haut de pile → (PC)
Syntaxe : **RETURN** *retour d'un sous programme*

MOVLW
Charge une constante c dans le registre W
c → (W)
Exemple :
Dans cet exemple on charge une valeur dans le registre W
Movlw 01010101b *on charge 55h dans le registre W*

MOVWF
Charge le registre W dans le registre f
(W) → (f)
Exemple : Dans cet exemple on charge une valeur dans le registre W puis on charge le contenu vers un registre temporaire..
Movwf Reg_temp *on met W dans le registre temporaire*

BTFSS
Vérifier l'état du bit b de f et sauter l'instruction suivante si b = 1
"Saute l'instruction si" (f{b}) = 1
Exemple :
Incf reg_temp *on incrémente la variable reg_temp*
Btfsc reg_temp,0 *on vérifie si bit0 du reg_temp =1 on saute goto sinon on excute l'instruction goto*
Goto process

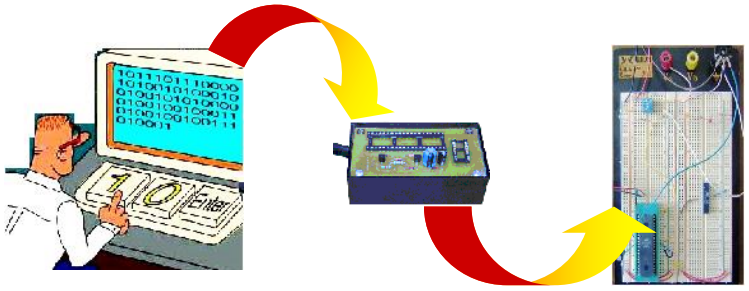
RETLW
Retour de la routine avec chargement d'une valeur c dans le registre W
c → (W) Haut de pile → (PC)
Syntaxe : RETLW afh *retour de sous programme et on charge afh dans W*

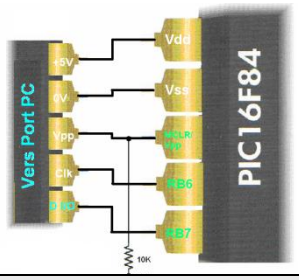
6. Programmation en assembleur

On peut écrire le programme avec plusieurs langages: assembleur, basic, pascal …Cependant, on s'intéressera uniquement à la programmation en assembleur puisque le compilateur est gratuit et téléchargeable depuis le site du constructeur www.microchip.com

Les outils nécessaires pour la programmation en assembleur sont:

- Un PC permet de confectionner le fichier .asm (un éditeur qlq , ou MPLAB)
- L'assembleur MPLAB fourni par la Société Microchip permet de confectionner le fichier .hex
- Un circuit programmeur de PIC, relié au PC, permet de transférer le pg .hex sur la mémoire programme du PIC en utilisant un software, on choisira le logiciel IC-PROG

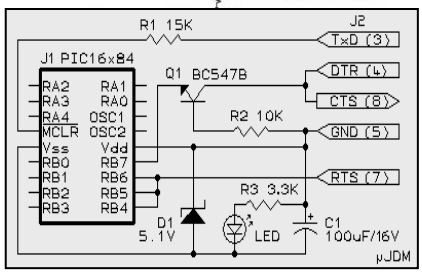




6.1. Circuit programmeur ou programmeur de PIC

Pour programmer le PIC, il est nécessaire de faire passer la tension de la broche MCLR à 12V (12 à 14V).

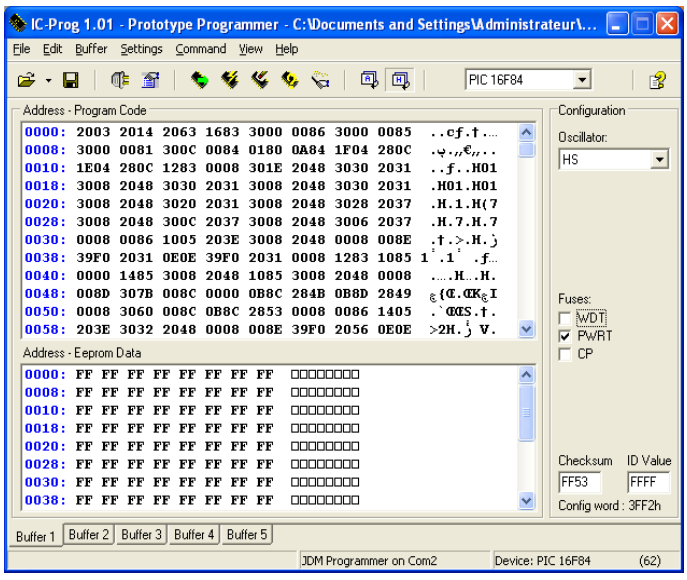
Le PIC16F84 se programme en appliquant un signal d'horloge sur la broche RB6 et les informations binaires (I/O) sérialisées sur la broche RB7. Chacune des informations qui transitent sur la broche 7 est validée par le signal d'horloge sur la broche 6.



6.2. Le software: IC-Prog

Ce programme permet de transférer le programme vers la mémoire du PIC, pour cela il faut plusieurs étapes :

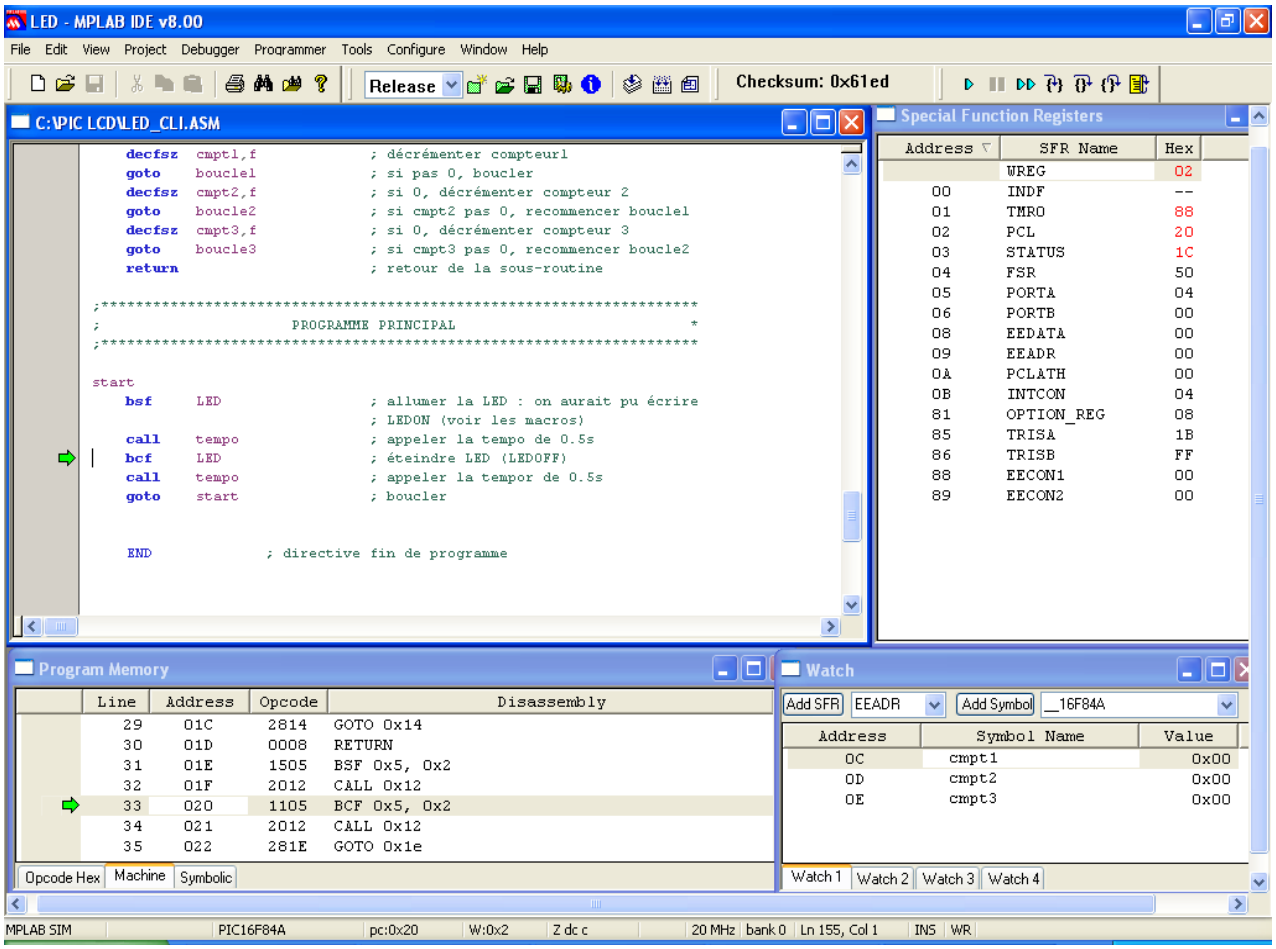
1- Fixer le type du PIC utilisé
2- Fixer le type du programmeur
3- Insérer le PIC dans le Programmeur et brancher au port série du PC
4- Effacer la mémoire du PIC
5- Ouvrir le fichier .hex à transférer
6- Envoyer le fichier vers la mémoire du PIC



6.3. Software MPLAB

Le programme écrit en assembleur doit porter l'extension .asm qu'on peut ouvrir sous MPLAB.
Il faut ensuite fixer le type du PIC, le système de numérotation utilisé, …

On démarre la compilation s'il n'y a pas d'erreur on obtient plusieurs fichiers et parmi eux un avec extension .hex c'est le fichier exécutable par le PIC.
L'opération suivante c'est la simulation du programme toujours sous MPLAB, ce qui permet de visualiser le fonctionnement du programme étape par étape en donnant les valeurs des registres spéciaux des variables utilisées …
On peut même simuler le changement d'état d'un pin (ou plusieurs) entrée (événement extérieur : Example bouton poussoir) pour provoquer une interruption.



6.3.1. Les directives de MPASM

Les directives de l'assembleur sont des commandes qu'on ajoute dans le programme et qui seront interprétées par l'assembleur MPASM. Ce ne sont pas des instructions destinées au PIC. Les directives les plus utilisées:

• LIST : permet de définir un certain nombre de paramètres comme le processeur utilisé (p), la base par défaut pour les nombres (r), le format du fichier hex à produire (f) ainsi que d'autres paramètres. Exemple :

LIST p=16F84A, r=dec, f=inhx8m

• INCLUDE : permet d'insérer un fichier source. Par exemple le fichier p16f84A.inc contient la définition d'un certain nombre de constante comme les noms des registres ainsi que les noms de certains bits;

INCLUDE "p16f84A.inc"

• __CONFIG : permet de définir les 14 fusibles de configuration qui seront copié dans l'EEPROM de configuration lors de l'implantation du programme dans le PIC (protection de code, type d'oscillateur, chien de garde et temporisation du départ)

```
__CONFIG B'11111111111001'
__CONFIG H'3FF9'
```

si le fichier p16f84.inc a été inséré, on peut utiliser les constantes prédéfinies :

```
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF
```

• EQU : permet de définir une constante ou une variable :

```
XX EQU 0x20
```

Chaque fois que le compilateur rencontrera XX, il la remplacera soit par la constante 0x20.

• #DEFINE : définit un texte de substitution

```
#DEFINE pos(x,y,z) (y-2z+x)
```

Chaque fois que le compilateur rencontrera le texte pos(x,y,z), il le remplacera par (y-2z+x)

• ORG : définit la position dans la mémoire programme à partir de laquelle seront inscrites les instructions suivantes.

• CBLOCK/ENDC : définit un bloc de constantes

```
CBLOCK 0X0C ; var1=0x0C, var2=0x0D, k=0x0D
var1,var2
k
ENDC
```

• END : indique la fin du programme

6.3.2. Structure d'un programme écrit en assembleur

Un programme écrit en assembleur doit respecter une certaine syntaxe et un certain nombre de règles afin qu'il soit facile à lire et à débuguer :

- Tout ce qui commence à la première colonne est considéré comme une étiquette (label) permettant de faire des renvois et aussi des assignations de constantes et de variables.

- Tout ce qui suit un point virgule est considéré comme un commentaire non interprété par le compilateur.

- Un programme apparaît donc comme un texte écrit sur 3 colonnes :
  - la colonne de gauche contient les étiquettes
  - la colonne du milieu contient les instructions
  - la colonne de droite contient des commentaires

6.3.3. Exemple de programme

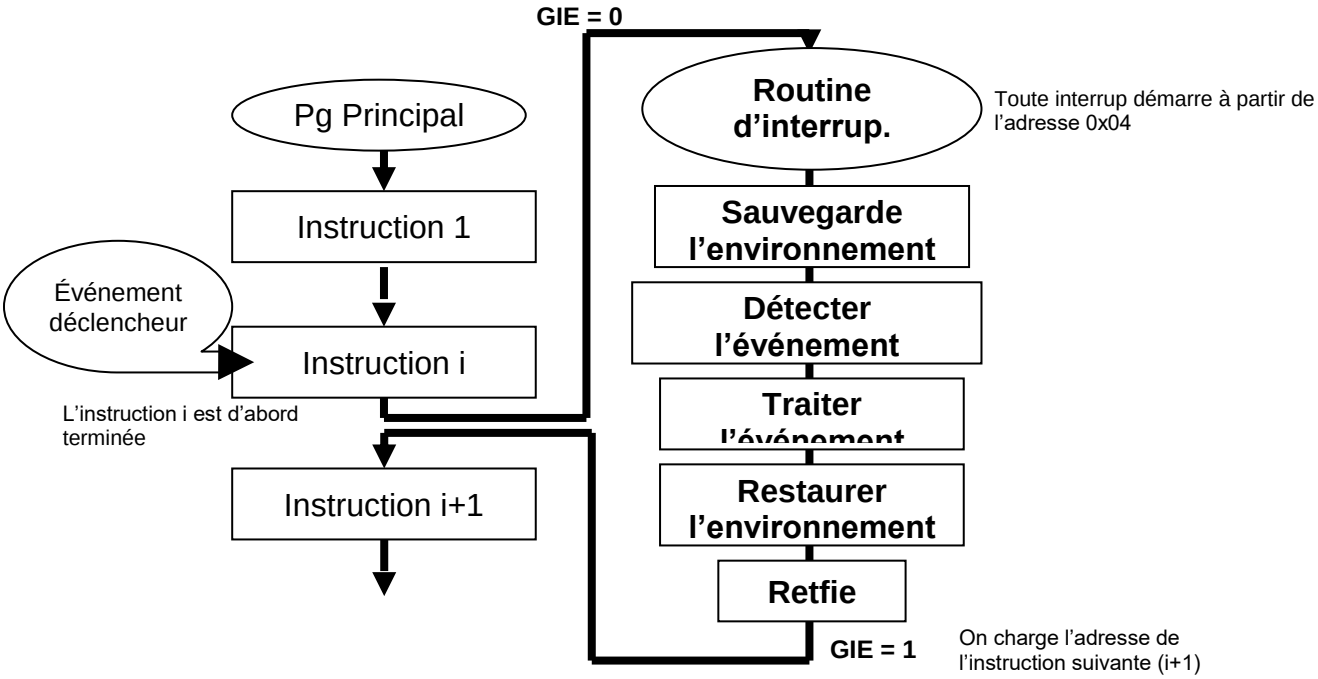
```
; Programme led_int.asm ************************************************************
; on connecte un interrupteur sur RB0 (entrée) et une LED sur RB1 (sortie)
; Si on place l'interrupteur à 1, la LED doit s'allumer, si on le met à zéro, elle ; doit s'éteindre
;************************************************************************************
    LIST p=16f84A              ; Définition de processeur
    INCLUDE "p16f84A.inc"   ; Définitions de variables
    __CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF ; fixe les
                                                ;fusibles
    bsf STATUS,RP0          ; bank 1
    movlw B'00000001'       ; charger w par 01h
    movwf TRISB             ; pour configurer RB0 en entrée et les autres
                           ;en sortie
    bcf STATUS,RP0          ; bank 0
tst                         ; étiquette
   btfss PORTB,0            ; teste si RB0=1
   goto off                 ; sinon goto off
   bsf PORTB,1              ; si oui mettre RB1=1
   goto tst                 ; et goto tst
off                         ; étiquette
   bcf PORTB,1                    ; mettre Rb1=0
   goto tst                 ; revenir au test
   end                      ; fin du programme
```

7. Interruptions:

a) Introduction :

L'interruption est un mécanisme fondamental dans le fonctionnement des microcontrôleurs, elle permet de prendre en compte des événements extérieurs et de leurs associer un traitement spécifique.

La séquence classique de fonctionnement d'une interruption :



Dans le cas du 16F84 les différentes sources d'interruption sont au nombre de 4:

INT: interruption externe broche RB0/INT
TMR0: interruption interne fin du comptage
PORTB: interruption externe changement d'état du PORTB (RB4 à RB7)
EEPROM: interruption interne fin d'écriture en EEPROM

***b)* Le registre INTCON :**

Le registre qui gère les interruptions est le registre INTCON qui est un registre de bit:

7	6	5	4	3	2	1	0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE : permet la validation générale des interruptions
EEIE: permet la validation de l'interruption de EEPROM
TOIE: permet la validation de l'interruption TMR0
INTE: permet la validation de l'interruption sur la broche RB0/INT
RBIE: permet la validation de l'interruption RB4 à RB7
TOIF: bit signal le débordement du TMR0
INTF: bit signal une transition sur RB0/INT
RBIF: bit signal un changement d'état de l'une des entrées RB4 à RB7

Les 3 bits de signal doivent être remis à 0 par le pg.

Le bit qui manque EEIF qui signal fin de l'écriture EEPROM se trouve dans le registre EECON1

c) Sauvegarde et restauration de l'environnement

Il faut sauvegarder les registres qui permettent au pg principal de fonctionner correctement après interruption.
En général c'est:

- STATUS
- Le registre de travail W

Le registre PC est sauvegardé automatiquement.

Pour sauvegarder W :
movwf w_temp ; il faut réserver un octet dans la RAM dans la zone des déclarations

Pour sauvegarder STATUS:
movf STATUS , w ; écrire la valeur de STATUS dans W
movwf sauve_Status ; sauvegarder STATUS dans sauve_Status
Malheureusement la 1ère instruction modifie le STATUS !!!

Pour remédier on utilise l'instruction SWAP qui permet d'inverser les 4 bits de poids faibles avec ceux de poids forts d'un octet.
Swapf STATUS , W ; swap STATUS dans W
movwf Status_temp ; sauve W dans Status_temp

Pour restaurer STATUS
Swapf Status_temp , W ; ancien STATUS dans W
movwf STATUS ; restaure STATUS

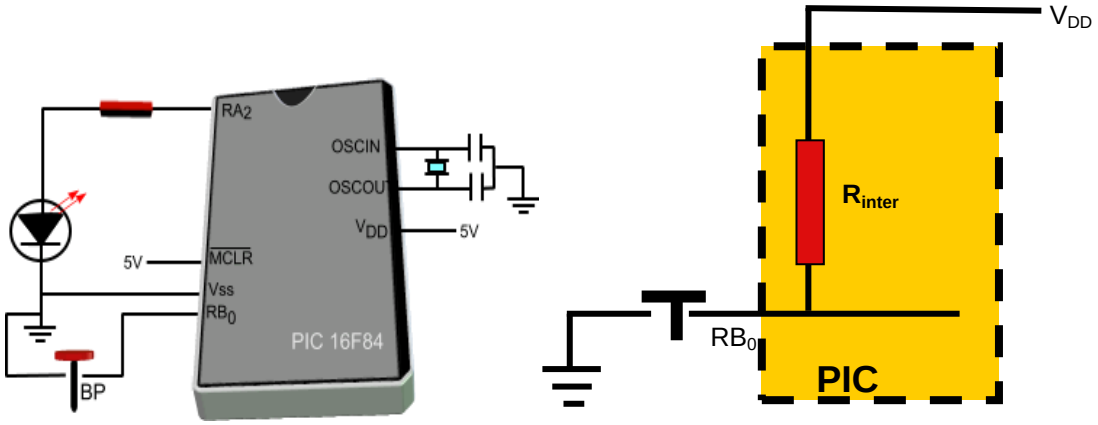
Pour restaurer W il faut faire attention pour ne pas modifier le registre STATUS :
Swapf w_temp , f ; swap w-temp dans elle même
Swapf w_temp , W ; swap w_temp dans W on obtient l'ancien valeur de W

d) Résumé : La structure de base d'une routine d'interruption

```
;-----------------------------------------------------------------------------------------------------------
;                                    Routine d'interruption
;-----------------------------------------------------------------------------------------------------------
                    ; Sauvegarde des registres
        Org 0x04            ; adresse des interruptions
        movwf w_temp        ; sauve W
        swapf STATUS , w    ; sauve STATUS
        movwf status_temp   ; ----------
                             ; on teste l'origine de l'interruption (on test les bist signal)
                              ; on traite l'interruption, on efface son bit de signal (flag ou drapeau)
                             ; on restaure les registres
        swapf status_temp , w       ; restaure STATUS
        movwf STATUS        ; --  -- -- --  -- -- --  --
        swapf w-temp , f    ; restaure W
        swapf w_temp , W    ; -- -- -- -- -- --
        retfie              ; retour de l'interruption (GIE =1 automatiquement)
```

e. Exemple d'utilisation d'une interruption:

Programme qui inverse l'allumage d'une Led à chaque pression sur un bouton poussoir BP
Pour cela on active les résistances internes de rappel du PORTB. L'entrée RB0 est à 1 et lorsqu'on appui sur BP elle passe à 0 et lorsqu'on relâche elle revient à 1.



Calculons la valeur à envoyer dans le registre OPTION:
b7=0 (RBPU) pour utiliser les résistances de rappel
b6=0 (INTEDG) pour fixer le front de l'interruption (↓)
b5=0 à **b0=0** aucun importance dans le pg (concerne Timer et Watchdog)

La valeur à mettre dans OPTION est donc : **0x00**

Pour le registre INTCON:
b7=1 (GIE) pour valider les interruptions
b6=0 (EEIE) pas d'interruption EEPROM
b5=0 (TOIE) pas d'interruption TMR0
b4=1 (INTE) Interruption RB0/INT activée
b3=0 (RBIE) pas d'interruption PORTB (RB4 à RB7)
b2=0 (TOIF) efface flag du TMR0
b1=0 (INTF) efface flag de RB0/INT
b0=0 (RBIF) efface flag du PORTB

La valeur à mettre dans INTCON est donc : **0x90**

```
;*****************************************************************************************
;   Ce programme est un programme didactique destiné à montrer                          *
;   le fonctionnement des interruptions                                                 *
;*****************************************************************************************
;   NOM:      Interruption par bouton-poussoir sur RB0                                  *
;   Date:                                                                               *
;   Version:                                                                            *
;   Auteur:                                                                             *
;*****************************************************************************************
        LIST    p=16F84A            ; Définition de processeur
        #include <p16F84.inc>       ; Définitions de constantes

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

; '__CONFIG' précise les paramètres au moment de la programmation. Les définitions sont dans le fichier include.
; Voici les valeurs et leurs définitions :
;       _CP_ON                  Code protection ON : impossible de relire
```

```
;       _CP_OFF             Code protection OFF
;       _PWRTE_ON           Timer reset sur power on en service
;       _PWRTE_OFF          Timer reset hors-service
;       _WDT_ON             Watch-dog en service
;       _WDT_OFF            Watch-dog hors service
;       _LP_OSC             Oscillateur quartz basse vitesse
;       _XT_OSC             Oscillateur quartz moyenne vitesse
;       _HS_OSC             Oscillateur quartz grande vitesse
;       _RC_OSC             Oscillateur à réseau RC

;*****************************************************************************************
;       ASSIGNATIONS                                                                    *
;*****************************************************************************************
OPTIONVAL   EQU     H'0000'     ; Valeur registre option
INTERMASK   EQU     H'0090'     ; Masque d'interruption, interruptions sur RB0
;*****************************************************************************************
;               DEFINE                                                                  *
;*****************************************************************************************
#DEFINE     Bouton PORTB , 0    ; bouton poussoir
#DEFINE LED         PORTA , 2   ; LED
;*****************************************************************************************
;               MACRO                                                                   *
;*****************************************************************************************
BANK0       macro
        bcf     STATUS,RP0 ; passer en banque 0
        endm
BANK1       macro
        bsf     STATUS,RP0 ; passer en banque1
        endm
;*****************************************************************************************
;           DECLARATIONS DE VARIABLES                                                   *
;*****************************************************************************************
        CBLOCK 0x00C        ; début de la zone variables
        w_temp :1           ; Sauvegarde de W dans interruption
        status_temp : 1     ; Sauvegarde de STATUS dans interrupt
        cmpt1 : 1           ; compteur de boucles 1 dans tempo
        cmpt2 : 1           ; compteur de boucles 2 dans tempo
        flags : 1           ; un octet pour 8 flags
                            ; réservons b0 pour le flag tempo
        ENDC                ; Fin de la zone

#DEFINE     tempoF      flags , 0       ; Définition du flag tempo

;*****************************************************************************************
;           DEMARRAGE SUR RESET                                                         *
;*****************************************************************************************
        org     0x000 ; Adresse de départ après reset
        goto     init   ; Adresse 0: initialiser

;*****************************************************************************************
;           ROUTINE INTERRUPTION                                                        *
;*****************************************************************************************
                    ;sauvegarder registres
                    ;----------------------------
        org     0x004               ; adresse d'interruption
```

```
        movwf   w_temp              ; sauver registre W
        swapf   STATUS,w            ; swap status avec résultat dans w
        movwf   status_temp         ; sauver status swappé

                        ; Appel de l'interruption
                        ;-----------------------------
        call    intrb0          ; traiter interrupt RB0

                        ;restaurer registres
                        ;-----------------------
        swapf   status_temp,w       ; swap ancien status, résultat dans w
        movwf   STATUS              ; restaurer status
        swapf   w_temp,f            ; Inversion L et H de l'ancien W sans modifier Z
        swapf   w_temp,w            ; Réinversion de L et H dans W, W restauré sans modifier status
        retfie                      ; return from interrupt
;*****************************************************************************************
;               INTERRUPTION RB0/INT                                                    *
;*****************************************************************************************
; Inverse le niveau de RA2 à chaque passage,  interdit toute nouvelle interruption et  valide le flag tempo
;-------------------------------------------------------------------------------------------------------------- ----------
intrb0
        movlw   B'00000100'         ; bit positionné = bit inversé
        BANK0                       ; se placer en bank0
        xorwf   PORTA , f           ; inverser RA2
        bcf     INTCON , INTF       ; effacer flag INT/RB0
        bcf     INTCON , INTE       ; interdire autre inter. RB0
        bsf     tempoF              ; positionner flag tempo
        return                      ; fin d'interruption RB0/INT
;*****************************************************************************************
;               INITIALISATIONS                                                         *
;*****************************************************************************************
init
        clrf    PORTA               ; Sorties portA à 0
        clrf    PORTB               ; sorties portB à 0
        clrf    EEADR               ; permet de diminuer la consommation
        BANK1                       ; passer banque1
        movlw   OPTIONVAL           ; charger masque
        movwf   OPTION_REG          ; initialiser registre option

                        ; Effacer RAM
                        ; -----------------
        movlw   0x0c            ; initialisation pointeur
        movwf   FSR             ; pointeur d'adressage indirect
init1
        clrf    INDF            ; effacer ram
        incf    FSR,f           ; pointer sur suivant
        btfss   FSR,6           ; tester si fin zone atteinte (>=40)
        goto    init1           ; non, boucler
        btfss   FSR,4           ; tester si fin zone atteinte (>=50)
        goto    init1           ; non, boucler

                        ; configuer PORTS
                        ; ----------------------
```

```
        bcf     LED             ; RA2 en sortie (TRISA)
        BANK0                   ; passer banque0
        movlw INTERMASK         ; masque interruption
        movwf INTCON            ; charger interrupt control
        goto    start           ; sauter programme principal
;*****************************************************************************************
;               SOUS-ROUTINE DE TEMPORISATION                                           *
;*****************************************************************************************
; Cette sous-routine introduit un retard
; Elle ne reçoit aucun paramètre et n'en retourne aucun
;-----------------------------------------------------------------
tempo
        clrf    cmpt2           ; effacer compteur2
boucle2
        clrf    cmpt1           ; effacer compteur1
boucle1
        decfsz  cmpt1,f         ; décrémenter compteur1
        goto    boucle1         ; si pas 0, boucler
        decfsz  cmpt2,f         ; si 0, décrémenter compteur 2
        goto    boucle2         ; si cmpt2 pas 0, recommencer boucle1
        return                  ; retour de la sous-routine
;*****************************************************************************************
;               PROGRAMME PRINCIPAL                                                     *
;*****************************************************************************************
start
        btfss   tempoF          ; tester si tempo flag mis
        goto    start           ; non, attendre qu'il soit mis
        call    tempo           ; oui, exécuter tempo
        bcf     tempoF          ; effacer flag tempo
        bcf     INTCON,INTF     ; effacer flag INT
        bsf     INTCON,INTE     ; remettre interrupts INT en service
        goto    start           ; boucler

        END                     ; directive fin de programme
```

8. L'adressage indirect

Le registre INDF (adresse 00 et 80H) et le registre FSR (adresse 04 et 84H) permettent l'adressage indirect.
Dans le registre FSR on met l'adresse (c'est un pointeur), et le registre INDF contient alors la donnée qui se trouve à l'adresse pointée par FSR.

Exp.1:

Le pg suivant permet de lire les données qui se trouvent entre l'adresse 20 et 22 de la RAM en utilisant l'adressage indirect

Adresse	Valeur
20	3
21	4
22	5

```
        movlw   0x20        ; on met dans w la 1ère adresse
        movwf   FSR         ; on charge FSR par cette adresse
Init    movf    INDF , w    ; on met dans w la donnée qui se trouve dans l'adresse FSR
        movwf   var         ; on met le contenu de w dans var
        incrf   FSR , f     ; on incrémente FSR pour passer à l'adresse suivante
        goto    Init        ; on reboucle pour lire les autres valeurs
```

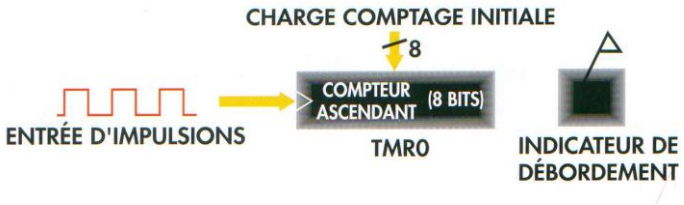
Exp.2:

Soit à effacer la zone RAM réservée à l'utilisateur de l'adresse 0x0C à 0x4F (0100 1111). Donc on va démarrer par l'adresse 0x0C et on va faire le test de fin sur l'adresse 0x50 (0101 0000) donc sur les bits 4 et 6.

```
        movlw 0Ch           ; on initialise par la première adresse
        movwf FSR           ; qu'on met dans FSR
INIT    clrf  INDF          ; on efface la case pointée
        incrf FSR , f       ; on passe à l'adresse suivante
        btfss FSR , 6       ; on teste la mise à 1 du bit6 si oui on saute l'instruction suivante
        goto  INIT          ; le bit6 n'est pas à 1 on reboucle pour effacer les autres adresses
        btfss FSR , 4       ; maintenant que le bit6 = 1 on teste la mise à 1 du bit4 si oui on saute
        goto  INIT          ; le bit4 n'est pas égale à 1 on boucle
        - -
        - -
```

9- Le Timer

Dans la majeure partie des applications, il est nécessaire de contrôler le temps, et afin de ne pas occuper le microcontrôleur qu'à cette tâche (boucle de comptage qui monopolise le µc), on le décharge en utilisant un Timer.
Le TMR0 est un compteur ascendant (qui compte) de 8 bits qui peut être chargé avec une valeur initiale quelconque. Il est ensuite incrémenté à chaque coup d'horloge jusqu'à ce que le débordement ait lieu (passage de FF à 00); Le principe est représenté sur la figure.

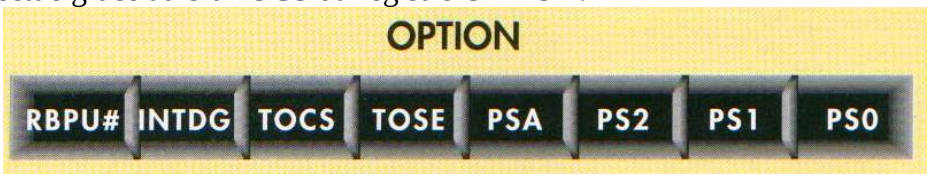


Le TMR0 peut remplir deux fonctions:

-Temporisateur ou contrôle du temps. Son entrée d'incrémentation est alors l'horloge qui correspond au cycle instruction (Fosc/4). Il est possible d'utiliser un pré diviseur de fréquence que nous verrons dans la suite.

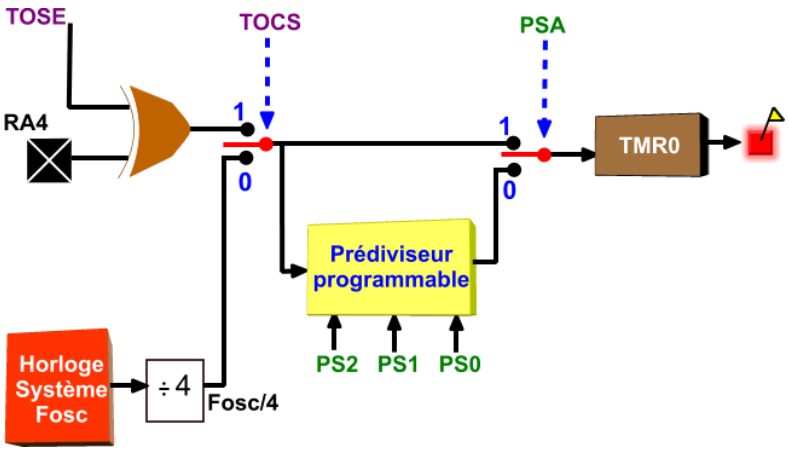
-Compteur d'événements. Dans ce cas les d'impulsions d'entrées du Timer sont fournies par la patte RA4/TOCK1

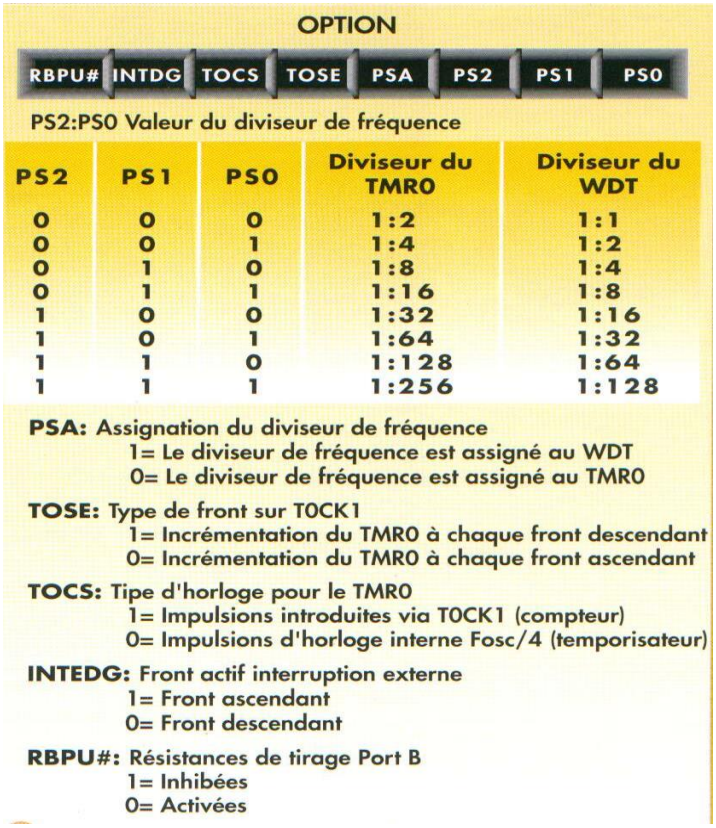
Le choix s'effectue grâce au bit TOCS du registre OPTION.



Le pic 16F84 dispose d'un **seul diviseur de fréquence** (prédiviseur) qui peut être assigné **soit** au chien de garde, **soit** au TMR0 (uniquement à un).

L'assignation du pré diviseur se fait grâce au bit PSA du registre OPTION.





OPTION

RBPU#	INTDG	TOCS	TOSE	PSA	PS2	PS1	PS0

PS2:PS0 Valeur du diviseur de fréquence

PS2	PS1	PS0	Diviseur du TMR0	Diviseur du 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

PSA: Assignation du diviseur de fréquence
1= Le diviseur de fréquence est assigné au WDT
0= Le diviseur de fréquence est assigné au TMR0

TOSE: Type de front sur T0CK1
1= Incrémentation du TMR0 à chaque front descendant
0= Incrémentation du TMR0 à chaque front ascendant

TOCS: Tipe d'horloge pour le TMR0
1= Impulsions introduites via T0CK1 (compteur)
0= Impulsions d'horloge interne Fosc/4 (temporisateur)

INTEDG: Front actif interruption externe
1= Front ascendant
0= Front descendant

RBPU#: Résistances de tirage Port B
1= Inhibées
0= Activées

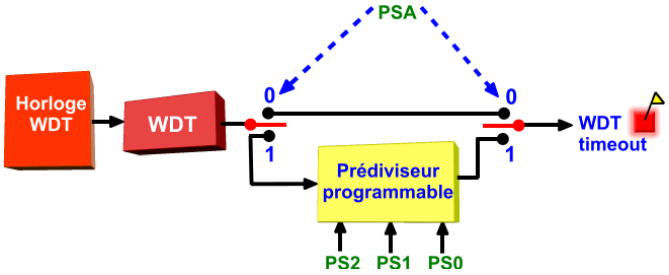
Si nous décidons d'utiliser le timer0 dans sa fonction Timer et en mode interruption; Nous aurons, avec un quartz de 4MHz, une interruption toutes les 256µs, soit à peu près tous les quarts de millième de seconde. Si on utilise le prédiviseur on multiplie ce temps soit par 2, 4, 8, …ou 256 fixé par les bits PS2 à PS0.

Donc on peut avoir une interruption toutes les 256µs ou 512µs ou … ou 65536µs.
On a le même résultats si on utilise le Timer en mode compteur, on aura une interruption tous les 256 événements, ou 512 événements, …ou 65536 événements.

PSA	PS2	PS1	PS0	/TMR0	/WD	Temps TMR0	Temps WTD (min)
0	0	0	0	2	1	512µs	18ms (7ms)
0	0	0	1	4	1	1024µs	18ms (7ms)
0	0	1	0	8	1	2048µs	18ms (7ms)
0	0	1	1	16	1	4096µs	18ms (7ms)
0	1	0	0	32	1	8192µs	18ms (7ms)
0	1	0	1	64	1	16384µs	18ms (7ms)
0	1	1	0	128	1	32768µs	18ms (7ms)
0	1	1	1	256	1	65536µs	18ms (7ms)
1	0	0	0	1	1	256µs	18ms (7ms)
1	0	0	1	1	2	256µs	36ms (14ms)
1	0	1	0	1	4	256µs	72ms (28ms)
1	0	1	1	1	8	256µs	144ms (56ms)
1	1	0	0	1	16	256µs	288ms (112ms)
1	1	0	1	1	32	256µs	576ms (224ms)
1	1	1	0	1	64	256µs	1152ms (448ms)
1	1	1	1	1	128	256µs	2304ms (996ms)

10- Le chien de garde (Watchdog)

Le watchdog, ou chien de garde est un mécanisme de protection du programme. Il sert à surveiller si celui-ci s'exécute toujours dans l'espace et dans le temps que vous lui avez attribués.
Le watchdog est destiné à vérifier que votre programme ne s'est pas «égaré» dans une zone non valide de votre programme (parasite sur l'alim.), ou s'il n'est pas bloqué dans une boucle sans fin (bug du programme). Il sert également à réveiller un PIC placé en mode SLEEP.
Sa mise en service ou son arrêt se fait au moment de la programmation par la directive:
_CONFIG _WDT_ON (_OFF).
Le fonctionnement du Watchdog est lié à un Timer interne spécifique, la durée spécifique de débordement est de 18ms, valeur qui varie avec les paramètres (tension, température,…). La valeur minimale utilisée dans la pratique est 7ms.
Si le prédiviseur est assigné au Watchdog le temps peut être multiplié par 2, 4, ..ou 128.



Chaque fois que l'instruction CLRWDT est envoyée au pic le Timer du Watchdog est remis à zéro. Si le PIC n'a pas reçu cette instruction dans le délai prévu, il redémarre à l'adresse 0x0000 et le bit TO du registre STATUS est mis à 0.
On doit placer plusieurs instructions « CLRWDT» dans le programme, de sorte qu'une instruction CLRWDT soit reçue dans les délais fixés par le PIC (7ms x le rapport du prédiviseur).

11- Le mode Sleep

C'est un mode qui permet au PIC de se mettre en sommeil afin de limiter sa consommation. Ce ci est réalisé à l'aide de l'instruction SLEEP, le PIC est placé alors en sommeil et cesse d'exécuter le programme:

- Le Timer du Watchdog est remis à 0
- Le bit TO du registre STATUS est mis à 1
- Le bit PD du registre STATUS est mis à 0
- L'oscillateur est arrêté

Le Watchdog continue de compter s'il est mis en service. Le passage en mode « Sleep » n'a réellement d'intérêt que s'il est possible d'en sortir. Les seuls événements susceptibles de réveiller le PIC:

- Niveau bas sur le pin MCLR ce qui provoque un reset à l'adresse 0000.

- Ecoulement du temps du Timer du Watchdog s'il est mis en service. Dans ce cas, le PIC est seulement réveillé, et il exécutera l'instruction qui suit l'instruction SLEEP.

- Une interruption RB0/INT, RB ou EEPROM, s'elles sont programmées et que le bit GIE n'est pas positionné (on positionne juste les bits des interruptions). Dans ce cas le PIC se réveille est exécute l'instruction suivante.

Exemple de programme de mise en veille du PIC

```
_CONFIG   _WDT_ON & ….
…….
…….
OPTIONVAL EQU H'8E'          ; Valeur à mettre dans le registre option
                             ; Préscaler wdt à 64 (1152ms nom et 448ms min)
……
#DEFINE LED PORTA, 2         ; LED de sortie
BCF   LED                    ; LED éteinte
…..
;*****************************************************************
;       Programme principal
;*****************************************************************
Start
  BSF      LED               ; Allumage de LED
  SLEEP                      ; Mise en sommeil
  BCF      LED               ; Extinction de la LED
  SLEEP                      ; Mise en sommeil
  GOTO     Start             ; Boucler
  END                        ; Fin
```

Cas typiques d'utilisation du mode SLEEP :

- ✓ Ce mode de fonctionnement est principalement utilisé dans les applications dans lesquelles la consommation en énergie doit être économisée (piles). On placera donc dans ce cas le PIC en mode POWER DOWN ou SLEEP aussi souvent que possible.

- ✓ Une autre application typique est un programme dans lequel le PIC n'a rien à faire dans l'attente d'un événement extérieur particulier. Dans ce cas, une instruction SLEEP fera efficacement l'affaire au lieu de mettre des boucles sans fin.

Attention: Avant d'exécuter l'instruction Sleep il faut s'assurer que les flags sont effacés

12. Astuces :

a. Variables locales

Des variables qui ne seront utilisées qu'à l'intérieur d'une sous-routine, et qui ne servent plus une fois la sous-routine terminée

CBLOCK 0X0C
Local1 : 1 ; variable locale 1
Local2 : 1 ; variable locale 2
Resultat : 1 ; résultat pour fonction

On attribue les noms des variables locales en ajoutant des DEFINE ou des EQU.

```
#DE IN FE     cmpt1   Local1      ; compteur1 = variable locale1
#DEFINE       cmpt2   Local2      ; compteur2 = variable locale2
#DEFINE       inter1   Local1     ; résultat intermédiaire1 = variable locale1
#DEFINE       inter2   Local2     ; résultat intermédiaire2 = variable locale2
```

b. Soustraire une valeur de w

Le premier réflexe est le suivant :
sublw 5
En réalité on a effectué (5-w) au lieu de (w-5)

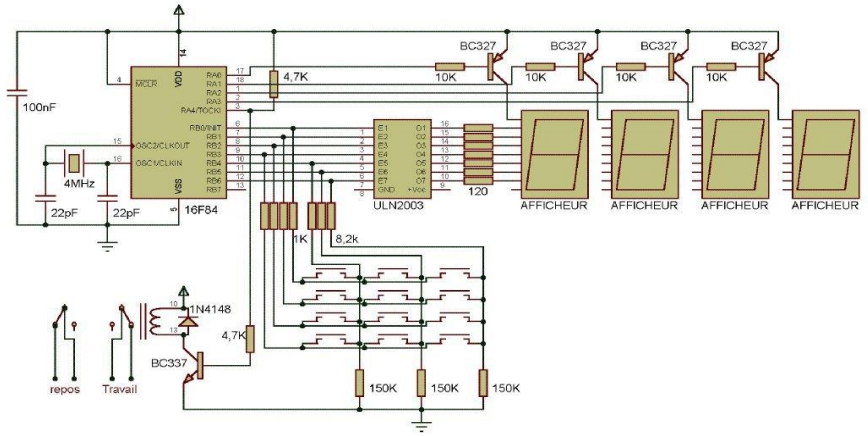
Effectuez plutôt ceci :
Addlw -5 ; (w=w+(-5))

c. Les tableaux en mémoire de programme :

Un tableau contenant le carré des nombres

```
carre
addwf PCL , f ; ajouter w à PCL
retlw .0 ; carré de 0 = 0
retlw .1 ; carré de 1 = 1
retlw .4 ; carré de 2 = 4
….
```

d. Contrainte nombre de entrées/sorties:



e. Autre directives de MPASM

DE : pour déclarer des donnés qui seront stockée dans l'EEPROM de donnée au moment de l'implantation du programme sur le PIC

ORG 0x2100

DE "Programmer un PIC, rien de plus simple", 70, 'Z'

DT : pour déclarer un tableau RETLW

proc addwf PCL,f ; saut à la position : (position suivante + W)

DT "Programmer un PIC",23,0x47 ; L'assembleur remplacera cette ligne par la suite d'instructions :

```
RETLW 'P'
RETLW 'r'
RETLW 'o'
. . .
RETLW 'C'
RETLW 23
RETLW 0x47
```

f. Accès en lecture dans la mémoire « EEPROM »

Pour lire une donnée en EEPROM, il suffit de placer l'adresse concernée dans le registre EEADR, positionner le bit RD à 1 et récupérer la donnée dans le registre EEDATA.

Ne pas oublier les changements de banques.

```
READEE  macro   adeeprom    ; macro avec 1 paramètre (argument)
        movlw   adeeprom    ; charger adresse eeprom (argument reçu)
        movwf   EEADR       ; adresse à lire dans registre EEADR
        bsf     STATUS , RP0 ; passer en banque1
        bsf     EECON1 , RD ; lancer la lecture EEPROM
        bcf     STATUS , RP0 ; repasser en banque0
        movf    EEDATA , w  ; charger valeur lue dans W
        endm                ; fin de la macro
```

g. L'accès en écriture à la zone EEPROM

La procédure à suivre consiste d'abord à placer la donnée dans le registre EEDATA et l'adresse dans EEADR. Ensuite une séquence spécifique en gris imposée par le constructeur.

```
WRITEE  macro   addwrite    ; la donnée se trouve dans W
        movwf   EEDATA      ; placer data dans registre
        movlw   addwrite    ; charger adresse d'écriture
        movwf   EEADR       ; placer dans registre
        bcf     INTCON , GIE ; interdire interruptions
        bsf     STATUS , RP0 ; passer en banque1
        bcf     EECON1,EEIF ; effacer flag de fin d'écriture
        bsf     EECON1,WREN ; autoriser accès écriture
        movlw   0x55        ; charger 0x55
        movwf   EECON2      ; envoyer commande
```

```
movlw   0xAA            ; charger 0xAA
movwf   EECON2          ; envoyer commande
bsf     EECON1 , WR     ; lancer cycle d'écriture

bcf     EECON1,WREN     ; verrouiller prochaine écriture
bsf     INTCON , GIE    ; réautorise interruptions
bcf     STATUS , RP0    ; repasser en banque0
endm
```

Microchip recommande que cette procédure spécifique ne soit pas interrompue par une interruption. Avant d'écrire sur l'EEPROM, Il faut toujours vérifier qu'une autre écriture n'est en cours.
A la fin de la procédure d'écriture, la donnée sera enregistrée dans l'EEPROM approximativement 10ms plus tard. La fin de l'écriture peut être constatée par:

- la génération d'une interruption (si le bit EEIE est positionné)
- la lecture du flag EEIF (s'il avait été remis à 0 avant l'écriture)
- la consultation du bit WR qui est à 1 durant tout le cycle d'écriture.

Le nombre de cycle d'écritures en EEPROM est limité environ 10 millions de cycles.

13 – La famille 16FXXX

PIC	Flash	RAM	EEPROM	I/O	A/D	Port //	Port série
16F870	2K	128	64	22	5	NON	USART
16F871	2K	128	64	33	8	PSP	USART
16F872	2K	128	64	22	5	NON	MSSP
16F873	4K	192	128	22	5	NON	USART/MSSP
16F874	4K	192	128	33	8	PSP	USART/MSSP
16F876	8K	368	256	22	5	NON	USART/MSSP
16F877	8K	368	256	33	8	PSP	USART/MSSP

De manière approximative un 16F876 est un 16F84 doté en supplément :
- De plus de mémoire RAM (répartie sur 4 banques), Flash, et EEPROM
- De plus de ports d'entrée/sortie
- De plus de Timers
- De nouvelles fonctionnalités, comme les gestions de ports « série »
- D'un convertisseur A/D (analogique/numérique) à plusieurs canaux d'une résolution de 10 bits.

14 interruptions disponibles sur le 16F876.

Déclencheur	Flag	Registre	Adr.	Enable	Registre	Adresse
Timer 0	TOIF	INTCON	0x0B	TOIE	INTCON	0x0B
RB0/INT	INTF	INTCON	0x0B	INTE	INTCON	0x0B
RB4/RB7	RBIF	INTCON	0x0B	RBIE	INTCON	0x0B
Convert A/D	ADIF	PIR1	0x0C	ADIE	PIE1	0x8C
Rx USART	RCIF	PIR1	0x0C	RCIE	PIE1	0x8C
Tx USART	TXIF	PIR1	0x0C	TXIE	PIE1	0x8C
Port série SSP	SSPIF	PIR1	0x0C	SSPIE	PIE1	0x8C
Module CCP1	CCP1IF	PIR1	0x0C	CCP1IE	PIE1	0x8C
Module CCP2	CCP2IF	PIR2	0x0D	CCP2IE	PIE2	0x8D
Timer 1	TMR1IF	PIR1	0x0C	TMR1IE	PIE1	0x8C
Timer 2	TMR2IF	PIR1	0x0C	TMR2IE	PIE1	0x8C
EEPROM	EEIF	PIR2	0x0D	EEIE	PIE2	0x8D
SSP mode I2C	BCLIF	PIR2	0x0D	BCLIE	PIE2	0x8D

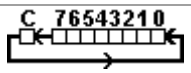
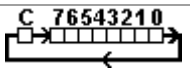
ADIE : Conversion analogique/digitale
RCIE : Réception d'un caractère sur le port série USART
TXIE : Emission d'un caractère sur le port série USART
SSPIE : Communication sur le port série synchrone SSP
CCP1IE : Evénement sur compare/capture registre 1
TMR2IE : Correspondance de valeurs pour le timer TMR2
TMR1IE : Débordement du timer TMR1
BCLIE : Collision de bus pour le port série synchrone I2C
CCP2IE : Evénement sur compare/capture registre 2

---------- **FIN** ----------

Les registres SFR des PICs

Adresse	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Valeur après Démarrage	Valeur après initialisation
Bank 0											
00h	INDF	Registre non réel utiliser dans l'adressage indirecte								- - - - - - - -	- - - - - - - -
01h	TMR0	Temporisateur/compteur en temps réel 8bit								xxxx xxxx	uuuu uuuu
02h	PCL	Les 8 bits de poids faibles du compteur programme PC								0000 0000	0000 0000
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Pointeur d''adresse pour l'adressage indirecte								xxxx xxxx	uuuu uuuu
05h	PORTA				RA4/TOCK1	RA3	RA2	RA1	RA0	xxxx xxxx	uuuu uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
07h		Adresse non utilisée								- - - - - - - -	- - - - - - - -
08h	EEDATA	Registre de données de l'EEPROM								xxxx xxxx	uuuu uuuu
09h	EEADR	Registre des adresses de l'EEPROM								xxxx xxxx	uuuu uuuu
0Ah	PCLATH				Les bits de poids forts du PC ne sont pas directement accessibles					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	utikiser dans l'adressage indirecte								- - - - - - - -	- - - - - - - -
81h	OPTION_reg	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Les 8 bits de poids faibles du compteur programme PC								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Pointeur d''adresse pour l'adressage indirecte								xxxx xxxx	uuuu uuuu
85h	TRISA				Registre de progammation du PORT A par bit					---1 1111	---1 1111
86h	TRISB	Registre de progammation du PORT B par bit [bit(i)=1: Rbi en entrée 0: sortie]								1111 1111	1111 1111
87h		Adresse non utilisée								- - - - - - - -	- - - - - - - -
88h	EECON1				EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	Registre de contrôle de l'EEPROM (Registre non réel)								---- ----	---- ----
8Ah	PCLATH				Les bits de poids forts du PC ne sont pas directement accessibles					---0 0000	---0 0000
8Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u

Les 35 instructions du pic 16F84

mnémonique	Description	cycle	bits affect
Instructions sur les registres (octets)			
ADDWF f,d	d:=W+f	1	C,DC,Z
ANDWF f,d	d:=W AND f	1	Z
CLRF f	f:=0	1	Z
CLRW	W:=0	1	Z
COMF f,d	d:=NOT(f)	1	Z
DECF f,d	d:=f-1	1	Z
DECFSZ f,d	d:=f-1 ; Skip if Zero	1(2)	.
INCF f,d	d:=f+1	1	Z
INCFSZ f,d	d:=f+1 ; Skip if Zero	1(2)	.
IORWF f,d	d:=W OR f	1	Z
MOVF f,d	d:=f	1	z
MOVWF f	W:=f	1	.
NOP	n'effectue aucune opération	1	.
RLF f,d	d=f SHL 1 	1	C
RRF f,d	d=f SHR 1 	1	C
SUBWF f,d	d:= f-W (en complément à 2 --> d:=f + not (W) +1)	1	C,DC,Z
SWAPF f,d	d:= f[4..7] <--> f[0..3] (inverse les quartets)	1	.
XORWF f,d	d:= W XOR f	1	Z
Instructions sur les registres (bit par bit)			
BCF f,b	f[b]:=0 (mets à 0 le bit b de f)	1	.
BSF f,b	f[b]:=1 (mets à 1 le bit b de f)	1	.
BTFSC f,b	teste le bit b de f ; Skip if Clear (0)	1(2)	.
BTFSS f,b	teste le bit b de f ; Skip if Set (1)	1(2)	.
Instructions de contrôle			
ADDLW k	W:=W+k	1	C,DC,Z
ANDLW k	W:=W AND k	1	Z
CALL k	appel un sous programme	2	.
CLRWDT	remet à 0 le Timer du chien de garde	1	TO,PD
GOTO k	se branche à l'adresse k	2	.
IORLW k	W:=W OR k	1	Z
MOVLW k	W:=k	1	.
RETFIE	fin d'une interruption	2	.

RETLW k	w:=k , puis effectue un retour de sous programme	2	.
RETURN	effectue un retour de sous programme	2	.
SLEEP	circuit en mode sommeil et stoppe l'oscillateur	1	TO,PD
SUBLW k	W:=k-W	1	C,DC,Z
XORLW k	W:=W XOR k	1	Z

Université Sultan Moulay Slimane
F.S.T. Béni Mellal
Département de Physique

TD
Microcontrôleur

Exercice 1 : Temporisation

Pour réaliser des temporisations il suffit de perdre du temps par des instructions sans rien faire. Un PIC cadencé par un quartz de fréquence 4MHz, exécute un cycle d'instruction par 1µs. Le PIC exécutera donc 1 millions d'instructions par seconde (1 MIPS). Essayons de réaliser une temporisation en utilisant une boucle qui va incrémenter ou décrémenter une variable cmpt1. Analyser la routine suivante :

```
Tempo
        clrf        cmpt1
boucle1
        decfsz      cmpt1,f
        goto        boucle1
        return
```

Calculer la durée de cette temporisation. On peut augmenter la durée de la temporisation en utilisant plusieurs boucles imbriquées. Déterminer la durée de la temporisation ci-dessous.

```
Tempo
        clrf        cmpt2
boucle2
        clrf        cmpt1
boucle1
        decfsz      cmpt1,f
        goto        boucle1
        decfsz      cmpt2,f
        goto        boucle2
return
```

Exercice 2 : Timer et prédiviseur

Le but rechercher dans cet exercice est d'utiliser l'interruption du TMR0 avec prédiviseur pour réaliser une temporisation de ½ s, soit 500.000 µs.

Quelle valeur faut-il mettre dans le registre INTCON ?

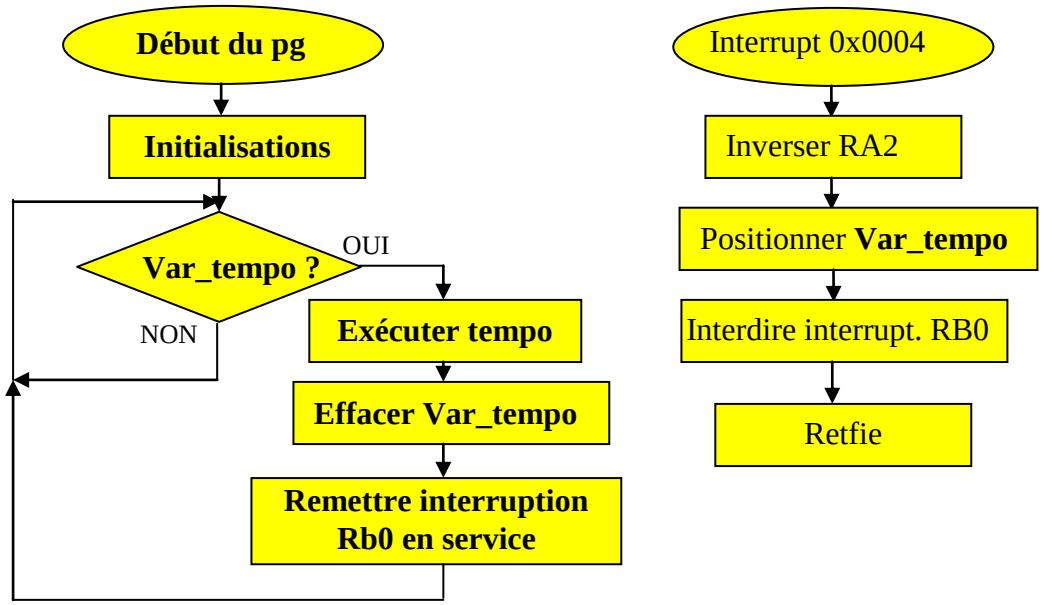
Sans prédiviseur le TMR0 génère une interruption toutes les 256µs, donc pour atteindre les 500000 µs nous avons besoin de 500000/256 débordement soit 1953,125 donc 1953 ce nombre dépasse la taille d'un octet. Avec un prédiviseur maximale 256 on a besoin de 500000/(256x256) soit 7,63 donc 7 interruption du TMR0. Quelle erreur on va commettre avec ce prédiviseur. Proposer une valeur du prédiviseur pour s'approcher le mieux de la temporisation cherchée.

Quelle valeur faut-il mettre dans le registre OPTION ?

Exercice 3 : Anti-rebond

Lorsqu'on appui sur un bouton poussoir (BP) la lame de celui-ci, après le premier contact, rebondit plusieurs fois avant de s'immobiliser. Ce rebondissement est considéré pour un microcontrôleur comme plusieurs appuis successifs. Comment peut–on remédier à ce problème.

Analyser les organigrammes suivants :



Exercice 4 : Analyse d'un programme.

```
LIST     p=16F84
         #include <p16F84.inc>
         __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
         OPTIONVAL       EQU      H'87'
                  INTERMASK      EQU      H'A0'
         TIMEBASE    EQU     H'07'
         #DEFINE LED         PORTA,2
         BANK0         macro
                  bcf      STATUS,RP0
                  endm

         BANK1         macro
                  bsf      STATUS,RP0
                  endm

         CBLOCK 0x00C
              w_temp :1
              status_temp : 1
              cmpt : 1
         ENDC

         org      0x000
         goto    init
                                    ; sauvegarder registres
                                    ;--------------------
```

```
        org     0x004
        movwf   w_temp
        swapf     STATUS,w
        movwf    status_temp
                        ; switch vers différentes interrupts
                        ;-------------------------------------

        btfsc   INTCON,T0IE
        btfss   INTCON,T0IF
        goto    intsw1
        call    inttimer
        bcf             INTCON,T0IF
        goto    restorereg

intsw1
        btfsc   INTCON,INTE
        btfss   INTCON,INTF
        goto    intsw2
        call    intrb0
        bcf             INTCON,INTF
        goto    restorereg

intsw2
        btfsc   INTCON,RBIE
        btfss   INTCON,RBIF
        goto    intsw3
        call    intrb4
        bcf             INTCON,RBIF
        goto    restorereg

intsw3
        BANK1
        btfsc   INTCON,EEIE
        btfss   EECON1,EEIF
        goto    restorereg
        call    inteep
                                        ; restaurer registres
                                        ;------------------
restorereg
        swapf  status_temp,w
        movwf   STATUS
        swapf   w_temp,f
        swapf   w_temp,w
        retfie
;*****************************************************************
;               INTERRUPTION TIMER 0                              *
;*****************************************************************
inttimer
        decfsz  cmpt,f
        return
```

```
        BANK0
        movlw b'00000100'
        xorwf  PORTA , f
        movlw TIMEBASE
        movwf cmpt
        return


;*****************************************************************************
;               INTERRUPTION RB0/INT                                          *
;*****************************************************************************
intrb0
        return                                  ; fin d'interruption RB0/INT
;*****************************************************************************
;               INTERRUPTION RB4/RB7                                          *
;*****************************************************************************
intrb4
        return                                  ; fin d'interruption RB4/RB7
;*****************************************************************************
;               INTERRUPTION EEPROM                                           *
;*****************************************************************************
inteep
        return                                  ; fin d'interruption eeprom
;*****************************************************************************
;                INITIALISATIONS                                              *
;*****************************************************************************
init
        clrf    PORTA
        clrf    PORTB
        clrf    EEADR
        BANK1
        movlw  OPTIONVAL
        movwf  OPTION_REG
                                        ; Effacer RAM
                                        ; ------------
        movlw 0x0c                              ; initialisation pointeur
        movwf FSR                               ; pointeur d'adressage indirect
init1
        clrf    INDF                            ; effacer ram
        incf    FSR,f                           ; pointer sur suivant
        btfss   FSR,6                           ; tester si fin zone atteinte (>=40)
        goto    init1                           ; non, boucler
        btfss   FSR,4                           ; tester si fin zone atteinte (>=50)
        goto    init1                           ; non, boucler
                                ; initialiser ports
                                ; -----------------
        bcf             LED
        BANK0
        movlw           INTERMASK
        movwf           INTCON
```

```
                                ; Initialisations variables
                                ; ------------------------
        movlw    TIMEBASE
        movwf   cmpt
;*****************************************************************************
;                PROGRAMME PRINCIPAL                                          *
;*****************************************************************************
start
        goto start
        END
```

Exercice 5 : Tableau des carrés dans la mémoire programme.

```
carre
        addwf PCL , f                   ; ajouter w à PCL

        retlw  .0                       ; carré de 0 = 0
        retlw  .1                       ; carré de 1 = 1
        retlw  .4                       ; carré de 2 = 4
        retlw  .9                       ; carré de 3 = 9
        retlw  .16                      ; carré de 4 = 16
        retlw  .25                      ; carré de 5 = 25
        retlw  .36                      ; carré de 6 = 36
        retlw  .49                      ; carré de 7 = 49
        retlw  .64                      ; carré de 8 = 64
        retlw  .81                      ; carré de 9 = 81
        retlw  .100                     ; carré de 10 = 100
        retlw  .121                     ; carré de 11 = 121
        retlw  .144                     ; carré de 12 = 144
        retlw  .169                     ; carré de 13 = 169
        retlw  .196                     ; carré de 14 = 196
        retlw  .225                     ; carré de 15 = 225
;*****************************************************************
;               PROGRAMME PRINCIPAL                                *
;*****************************************************************
start
        clrf    nombre
loop
        movf    nombre,w
        call    carre
        incf    nombre,f
        btfss   nombre,4
        goto    loop
        goto    start
        End
```

Analyser le programme.