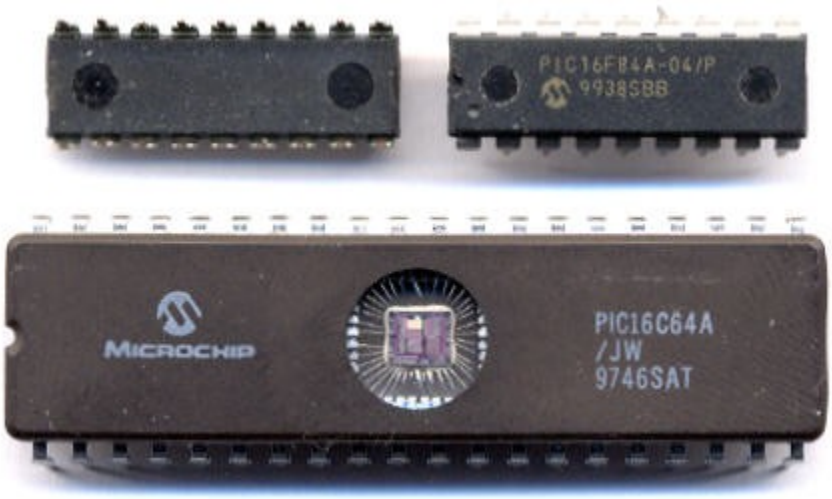
Les microcontrôleurs

PIC

de Microchip

Le 16F84



Sommaire

INTRODUCTION

Un PIC est un microcontrôleur, c'est une unité de traitement de l'information de type microprocesseur à laquelle on a ajouté des périphériques internes permettant de faciliter l'interfaçage avec le monde extérieur sans nécessiter l'ajout de composants externes.

Les PICs sont des composants RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus facile et plus rapide en est le décodage, et plus vite le composant fonctionne.

La famille des PICs est subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits, la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie la 16F84 et 16F876), et la famille **High-End**, qui utilise des mots de 16 bits.

Nous nous limiterons dans ce document à la famille Mid-Range et particulièrement au PIC 16F84, sachant que si on a tout assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur.

Pour identifier un PIC, on utilise simplement son numéro :

- Les 2 premiers chiffres indiquent la catégorie du PIC, 16 indique un PIC Mid-Range.
- Vient ensuite parfois une lettre L, celle-ci indique que le PIC peut fonctionner avec une plage de tension beaucoup plus tolérante.
- Vient en suite une ou deux lettres pour indiquer le type de mémoire programme :
  - **C** indique que la mémoire programme est une EPROM ou plus rarement une EEPROM
  - **CR** pour indiquer une mémoire de type ROM
  - **F** pour indiquer une mémoire de type FLASH.
- On trouve ensuite un nombre qui constitue la référence du PIC.
- On trouve ensuite un tiret suivi de deux chiffres indiquant la fréquence d'horloge maximale que le PIC peut recevoir.

Donc, un 16F84-04 est un PIC Mid-Range donc la mémoire programme est de type FLASH de référence 84 et capable d'accepter une fréquence d'horloge de 4MHz.

Notez que les PICs sont des composants STATIQUES, c'est à dire que la fréquence d'horloge peut être abaissée jusque l'arrêt complet sans perte de données et sans dysfonctionnement. Une version –10 peut donc toujours être employée sans problème en lieu et place d'une –04. Pas l'inverse, naturellement.

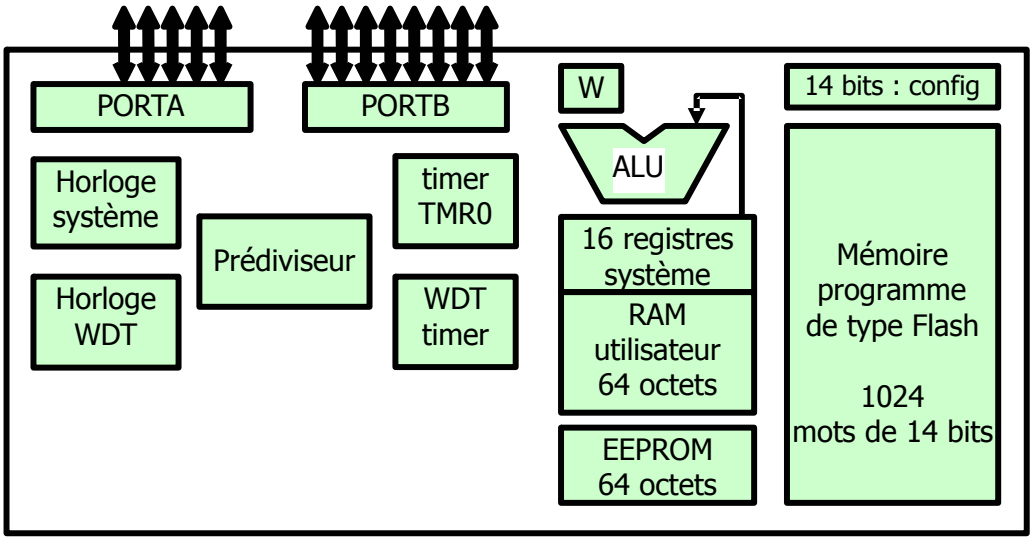
Pourquoi choisir un PIC ?

- Les performances sont identiques voir supérieurs à ses concurrents
- Les prix sont les plus bas du marché
- Très utilisé donc très disponible
- Les outils de développement sont gratuits et téléchargeables sur le WEB
- Le jeu d'instruction réduit est souple, puissant et facile à maîtriser
- Les versions avec mémoire flash présentent une souplesse d'utilisation et des avantages pratiques indéniables
- La communauté des utilisateurs des PICs est très présente sur le WEB. On trouve sur le net quasiment tout ce dont on a besoin, tutoriaux pour démarrer, documents plus approfondis, schémas de programmeurs avec les logiciels qui vont avec, librairies de routines, forums de discussion . . .

I LE PIC 16F84

Les caractéristiques principales du 16F84 sont :

- Une mémoire programme de type flash de 1K (1024) mots de 14 bits
- Une mémoire RAM constituée :
  - Des registres de control SFR (Special Function Registers)
  - 68 octets de RAM utilisateur appelés aussi GPR (General Propose Resisters)
- Une mémoire EEPROM de donnée de 64 octets
- Deux ports d'entrée sortie, un de 8 bits et un de 5 bits
- Un timer/Compteur cadencé par une horloge interne ou externe
- Un chien de garde / compteur qui est un timer particulier
- Un prédiviseur de fréquence programmable permettant d'étendre les possibilités du Timer TMR0 et du chien de garde WDT
- 4 sources d'interruption
- L'horloge peut être générée par 4 types d'oscillateurs sélectionnables
- Protection de code
- Fonctionnement en mode sleep pour réduction de la consommation
- Programmation par mode ICSP (In Circuit Serial Programming)



I.1 Aspect externe du 16F84

Le 16F84 est commercialisé dans un boîtier 18 broches classique

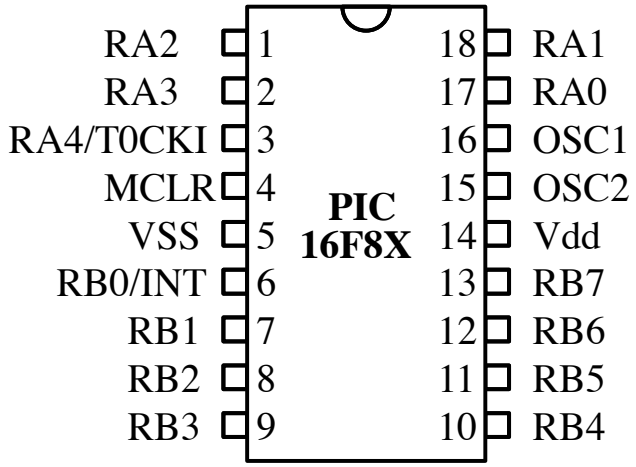


Fig. I-1 : brochage du 16 F84

I.2 La mémoire programme (flash)

Cette mémoire de 1024 mots stocke le programme. Elle est non volatile et reprogrammable à souhait. Chaque position de 14 bits contient une instruction. L'emplacement du programme peut se situer à n'importe quel endroit de la mémoire. Cependant il faut savoir que suite à un RESET ou lors de la mise sous tension, le PIC commence l'exécution à l'adresse 0000_H. De plus, lorsqu'il y a une interruption, le PIC va à l'adresse 0004_H. Il est donc conseillé de placer le début du programme après l'adresse 0004_H et de mettre un branchement au début du programme à l'adresse 0000_H et un branchement au début de la routine d'interruption s'il y en a une à l'adresse 0004_H. Le programme est implanté dans la flash à l'aide d'un programmateur (hard+soft) sur lequel nous reviendrons dans la suite de ce document.

I.3 La mémoire RAM - Rrgistres

La mémoire RAM est constituée de deux parties :

- Les registres SFR (Special Function Register), ce sont les registres de fonctionnement du PIC. L'ensemble de ces registres est souvent appelé fichier des registres. Nous reviendrons sur ces registres tout le long de ce document.

- Les registres GPR (General Propose Register) sont des positions mémoire que l'utilisateur peut utiliser pour stocker ses variables et ces données. On remarquera donc que, indépendamment de leur nature, les position de la RAM sont toujours appelé registres

La mémoire RAM est organisée en deux banks, pour accéder à un registre, il faut d'abord se placer dans le bank où il se trouve. Ceci est réalisé en positionnant le bit RP0 du registre STATUS. (RP0 = 0 → Bank 0, RP0 = 1 → Bank 1)

	bank 0	bank 1	
00	INDF	INDF	80
01	TMR0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C 4F	Mémoire utilisateur	Maped in bank0	8C . . . . CF

Registre STATUS

IRP	RP1	RP0		TO	PD	Z	DC	C		

Pour la mémoire utilisateur, l'utilisation des pages (Bank) n'est pas nécessaire puisque le Bank 1 est "mapped" avec le Bank0. Cela signifie qu'écrire une donnée à l'adresse $0C_H$ ou à l'adresse $8C_H$ revient au même.

I.4 L'ALU et le registre W

C'est une ALU 8 Bits qui réalise les opérations arithmétique et logique entre l'accumulateur W et n'importe quel autre registre 'F' ou constante K. L'accumulateur W est un registre de travail 8 bits, il n'a pas d'adresse comme les autres SFR. Pour les instructions à deux opérandes, c'est toujours lui qui contient un des deux opérandes. Pour les instructions à un opérande, celui-ci peut être soit W soit n'importe quel registre F. Le résultat de l'opération peut être placé soit dans le registre de travail W soit dans le registre F.

I.5 L'Horloge

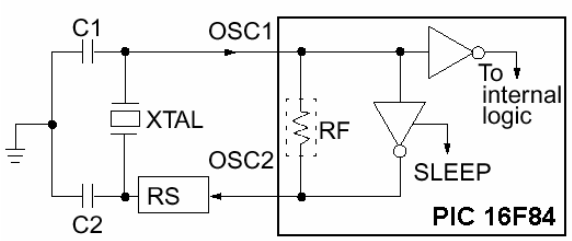
L'horloge peut être soit interne soit externe. L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC.

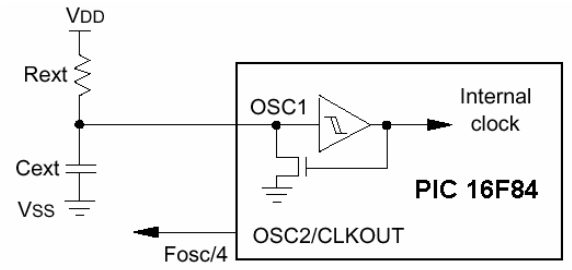
Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 4, 10 ou 20 MHz selon le type de µC. Le filtre passe bas RS, C2 limite les harmoniques dus à l'écrêtage et Réduit l'amplitude de l'oscillation. (il n'est pas obligatoire)

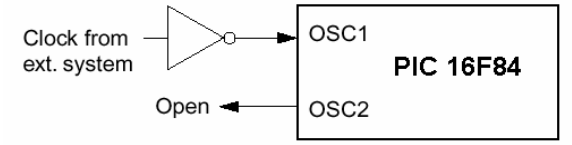
Avec un oscillateur RC, la fréquence de l'oscillation est fixée par Vdd, Rext et Cext. Elle peut varier légèrement d'un circuit à l'autre.

Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser le PIC sur un processus particulier.

Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Dans la suite de ce document on utilisera le terme Fosc/4 pour désigner l'horloge système.







Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs.

I.6 Le port d' E/S PORTA

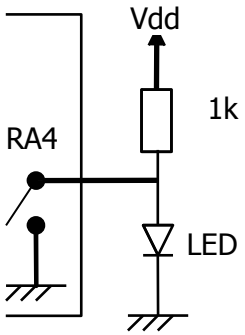
Le port A désigné par PORTA est un port de 5 bits (RA0 à RA4). Chaque E/S est compatible TTL. La configuration de direction pour chaque bit du port est déterminée avec le registre TRISA.

- Bit i de TRISA = 0 bit i de PORTA configuré en **sortie**
- Bit i de TRISA = 1 bit i de PORTA configuré en **entrée**

La broche RA4 est multiplexée avec l'entrée horloge du timer TMR0, elle peut donc être utilisée soit comme E/S normale du port A, soit comme entrée horloge pour le Timer TMR0, le choix se fait à l'aide du bit T0CS du registre OPTION_REG.

- T0CS = 0 RA4 est une E/S normale
- T0CS = 1 RA4 = horloge externe pour le timerTMR0

RA4 est une E/S à drain ouvert, si on veut l'utiliser comme sortie (pour allumer une LED par exemple), il ne faut pas oublier de mettre une résistance externe vers Vdd. Le schéma ci contre illustre (pour les non électronicien) le principe d'une sortie drain ouvert (ou collecteur ouvert) : si RA4 est positionnée à 0, l'interrupteur est fermé, la sortie est reliée à la masse. Si RA4 est placée à 1, l'interrupteur est ouvert, la sortie est déconnectée d'où la nécessite de la résistance externe pour amener le courant de l'alimentation vers la LED. (la valeur de 1k est donnée à titre indicatif, à vous d'ajuster selon votre application)



Registre OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	

I.7 Le port d' E/S PORTB

Le port port B désigné par PORTB est un port bidirectionnel de 8 bits (RB0 à RB7). Toutes les broches sont compatibles TTL. La configuration de direction se fait à l'aide du registre TRISB. (Voir PORTA / TRISA)
En entrée, la ligne RB0 appelée aussi INT peut déclencher l'interruption externe INT.
En entrée, une quelconque des lignes RB4 à RB7 peut déclencher l'interruption RBI.
Nous reviendrons là-dessus dans le paragraphe réservé aux interruptions.

I.8 Le Timer TMR0

C'est un compteur 8 bits ayant les caractéristiques suivantes :

- Il est incrémenté en permanence soit par l'horloge interne Fosc/4 (mode timer) soit par une horloge externe appliquée à la broche RA4 du port A (mode compteur). Le chois de l'horloge se fait à l'aide du bit T0CS du registre OPTION_REG
  - TOCS = 0 horloge interne
  - TOCS = 1 horloge externe appliquée à RA4

- Dans le cas de l'horloge externe, on peut choisir le front sur lequel le TIMER s'incrémente.
  - TOSE = 0 incrémentation sur fronts montants
  - TOSE = 1 incrémentation sur fronts descendants

- Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport est fixés par les bits PS0, PS1 et PS2 du registre OPTION_REG (tableau ci-contre). L'affectation ou non du prédiviseur se fait à l'aide du bit PSA du registre OPTION_REG
  - PSA = 0 on utilise le prédiviseur
  - PSA = 1 pas de prédiviseur (affecté au chien de garde)

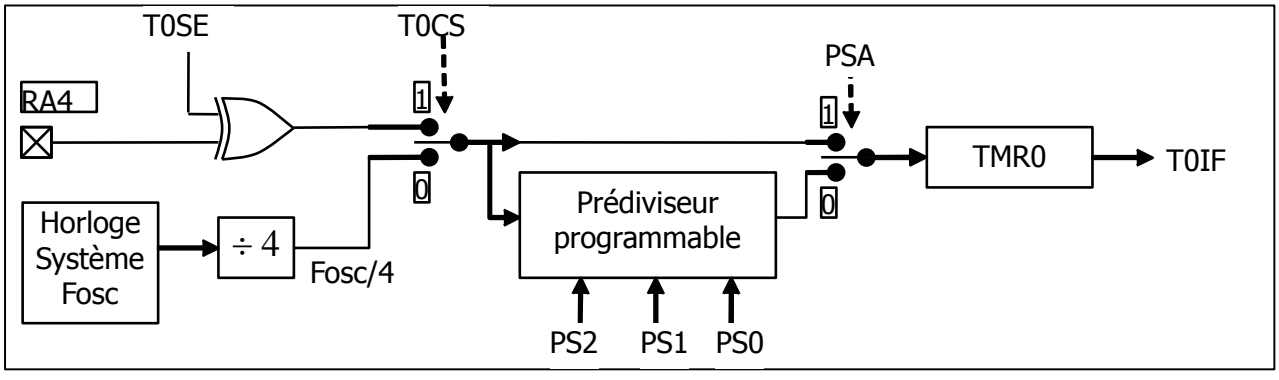
PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

- Le contenu du timer TMR0 est accessible par le registre qui porte le même nom. Il peut être lu ou écrit à n'importe quel moment. Après une écriture, l'incrémentation est inhibée pendant deux cycles instruction

- Au débordement de TMR0 (FF 00), le drapeau T0IF est placé à 1. Ceci peut déclencher l'interruption T0I si celle-ci est validée

Registre OPTION_REG

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	



I.9 Le WatchdogTimer WDT (Chien de garde)

C'est un compteur 8 bits incrémenté en permanence (même si le µC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur par un processus indésirable non contrôlé
- Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations

L'horloge du WDT est ajustée pour que Le Time-Out arrive toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passer le signal Time-Out dans un prédiviseur programmable (partagé avec le timer TMR0). l'affectation se fait à l'aide du bit PSA du registre OPTION_REG

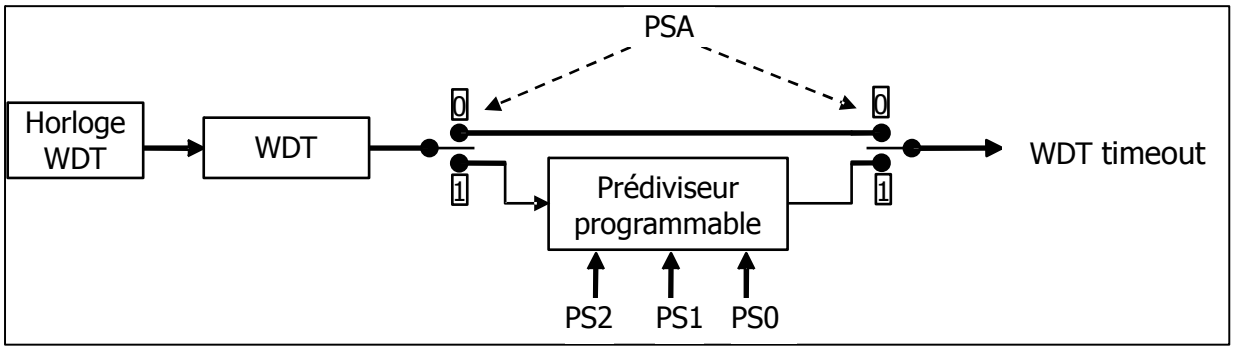
- PSA = 1 on utilise le prédiviseur
- PSA = 0 pas de prédiviseur (affecté à TMR0)

PS2	PS1	PS0	Div
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (voir tableau ci-contre)

L'utilisation du WDT doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un WDT timeOut lors de l'exécution d'un programme, on a deux possibilités :

- Inhiber le WDT d'une façon permanente en mettant à 0 le bit WDTE dans l'EEPROM de configuration
- Remettre le WDT à 0 périodiquement dans le programme à l'aide de l'instruction CLRWDT pour éviter qu'il ne déborde



I.10 Le mode SLEEP

Le PIC peut être placé en mode faible consommation à l'aide de l'instruction SLEEP. Dans ce mode, l'horloge système est arrêtée ce qui arrête l'exécution du programme.
Pour sortir du mode SLEEP, il faut provoquer un WAKE-UP, pour cela il y a 3 possibilités :

- RESET externe dû à l'initialisation du PIC en mettant l'entrée MCLR à 0. Le PIC reprend l'exécution du programme à partir du début.
- Timeout du chien de garde WDT si celui-ci est validé. Le PIC reprend le programme à partir de l'instruction qui suit l'instruction SLEEP
- Interruption INT (sur RB0) ou RBI (sur RB4-RB7) ou EEI (fin d'écriture en EEPROM de données). Le bit de validation de l'interruption en question doit être validé, par contre, le WAKE-UP a lieu quelque soit la position de bit de validation globale GIE. On a alors deux situations :

- GIE = 0, Le PIC reprend l'exécution du programme à partir de l'instruction qui suit l'instruction SLEEP, l'interruption n'est pas prise en compte
- GIE = 1, Le PIC exécute l'instruction qui se trouve juste après l'instruction SLEEP puis se branche à l'adresse 0004 pour exécuter la procédure d'interruption. Dans le cas où l'instruction suivant SLEEP n'est pas désirée, il faut utiliser l'instruction NOP.

L'utilisation des interruptions pour réaliser un WAKE-UP doit être utilisée avec précaution. Voir le data sheet[1][2] du 16F84 pour plus de précisions.

I.11 La mémoire EEPROM de configuration

Pendant la phase d'implantation d'un programme dans la mémoire programme du PIC, on programme aussi une EEPROM de configuration constituée de 5 mots de 14 bits :

- **4 mots d'identification** (ID) à partir de l'adresse 0x2000 pouvant contenir un repérage quelconque que nous n'utiliserons pas,

- **1 mot de configuration** (adresse 0x2007) qui permet :
  - de choisir le type de l'oscillateur pour l'horloge
  - de valider ou non le WDT timer
  - d'interdire la lecture des mémoires EEPROM de programme et de données.

- bits 1:0 **FOSC1**:**FOSC0** Sélection du type d'oscillateur pour l'horloge
  11 : Oscillateur RC
  10 : Oscillateur HS (High speed) : quartz haute fréquence (jusqu'à 10 MHz)
  01 : Oscillateur XT, c'est le mode le plus utilisé, quartz jusqu'à 4 MHz
  00 : Oscillateur LP (Low power), consommation réduite, jusqu'à 200 kHz

- bit 2 **WDTE** validation du timer WDT (chien de garde)
  1 : WDT validé
  0 : WDT inhibé

- Bit 3 **PWRTE** validation d'une temporisation à la mise sous tension
  1 : temporisation inhibée
  0 : temporisation validée

- Bit 13:4 **CP** Protection en lecture du code programme
  1 : pas de protection
  0 : protection activée

I.12 La mémoire EEPROM de données

La mémoire EEPROM de données est constituée de 64 octets commençant à l'adresse 0x2100 que l'on peut lire et écrire depuis un programme. Ces octets sont conservés après une coupure de courant et sont très utiles pour conserver des paramètres semi permanents.

On y accède à l'aide des registres EEADR et EEDATA : toute lecture écriture dans le registre EEDATA se fait dans la position mémoire pointée par EEADR. En fait EEADR contient l'adresse relative par rapport à la page qui commence en 0x2100, autrement dit, l'adresse va de 0 à 63.

Deux registres de contrôle (EECON1 et EECON2) sont associés à la mémoire EEMROM.

La durée d'écriture d'un octet est de l'ordre de 10 ms, la fin de chaque écriture réussie est annoncé par le drapeau EEIF et la remise à zéro du bit RW du registre EECON1. Le drapeau EEIF peut déclencher l'interruption EEI si elle a été validée.

I.12.1 Procédure de lecture dans l'EEPROM de données

- Placer l'adresse relative dans EEADR
- Mettre le bit RD de EECON1 à 1
- Lire le contenu du registre EEDATA

I.12.2 Procédure d'écriture dans l'EEPROM de données

1. L'écriture dans L'EEPROM doit être autorisée : bit WREN = 1
2. Placer l'adresse relative dans EEADR
3. Placer la donnée à écrire dans EEDATA
4. Placer 0x55 dans EECON2
5. Placer 0xAA dans EECON2
6. Démarrer l'écriture en positionnant le bit WR
7. Attendre la fin de l'écriture, (10 ms) (EEIF=1 ou WR=0)
8. recommencer au point 2 si on a d'autres données à écrire

Le drapeau WRERR est positionné si une erreur d'écriture intervient

EECON1 -

	-	-	EEIF	WRERR	WREN	WR	RD

EECON2 n'en est pas véritablement un Registre. Microchip l'utilise en tant que registre de commande. L'écriture de valeurs spécifiques dans EECON2 provoque l'exécution d'une commande spécifique dans l'électronique interne du PIC.

I.13 Les interruptions

Une interruption provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption. A la fin de cette procédure, le microcontrôleur reprend le programme à l'endroit où il s'était arrêté. Le PIC16F84 possède 4 sources d'interruption. A chaque interruption sont associés deux bits: un bit de validation et un drapeau. Le premier permet d'autoriser ou non l'interruption, le second permet au programmeur de savoir de quelle interruption il s'agit. Tous ces bits sont dans le registre INTCON à part le drapeau EEIF de l'interruption EEI qui se trouve dans le registre EECON1.

I.13.1 Déroulement d'une interruption

Lorsque l'événement déclencheur d'une interruption intervient, alors son drapeau est positionné à un (levé). Si l'interruption correspondante a été validée, elle est alors déclenchée : le programme arrête ce qu'il est en train de faire et va exécuter la procédure d'interruption qui se trouve à l'adresse 4 en exécutant les étapes suivantes :

- l'adresse contenue dans le PC (Program Counter) est sauvegardée dans la pile, puis remplacée par la valeur 0004 (adresse de la routine d'interruption).
- Le bit GIE est placé "0" pour inhiber toutes les interruptions (afin qu'on ne soit pas dérangés pendant l'exécution de la procédure d'interruption)
- A la fin de la procédure d'interruption (instruction RETFIE) :
  - le bit GIE est replacé à l'état haut (autorisant ainsi un autre événement)
  - le contenu du PC est rechargé à partir de la pile ce qui permet au programme de reprendre là où il s'est arrêté

Deux remarques importantes sont à faire :

Le drapeau reste à l'état haut même après le traitement de l'interruption. Par conséquent, il faut toujours le remettre à "0" à la fin de la routine d'interruption sinon l'interruption sera déclenchée de nouveau juste après l'instruction RETFI

Seul le PC est empilé automatiquement. Si cela est nécessaire, les registres W et STATUS doivent être sauvegardés en RAM puis restaurés à la fin de la routine pour que le microcontrôleur puisse reprendre le programme dans les mêmes conditions où il l'a laissé.

I.13.2 L'interruption INT (Entrée RB0 du port B)

Cette interruption est provoquée par un changement d'état sur l'entrée RB0 du port B quand elle est programmée en entrée. Elle est gérée par les bits :

- INTE : bit de validation (1=oui, 0=non)
- INTF : drapeau
- INTEDG : front de déclenchement, 1=montant, 0=descendant (registre OPTION_REG)

I.13.3 L'interruption RBI (RB4 A RB7 du port B)

Cette interruption est provoquée par un changement d'état sur l'une des entrées RB4 à RB7 du port B, Le front n'a pas d'importance. Les bits associés sont RBIE (validation) et RBIF (drapeau)

I.13.4 L'interruption T0I : Débordement du Timer TMR0

Cette interruption est provoquée par le débordement du timer TMR0. Les bits associés sont T0IE (validation) et T0IF (drapeau)

I.13.5 L'interruption EEI : Fin d'écriture dans l'EEPROM

Cette interruption est déclenchée à la fin d'une écriture réussie dans l'EEPROM.
Les bits associés sont EEIE (validation) et EEIF (drapeau).

INTCON GIE		EEIE	T0IF	INTE	RBIE	T0IF	INTF	RBIF
EECON1 -		-	-	EEIF	WRERR	WREN	WR RD	
OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA PS2		PS1	PS0

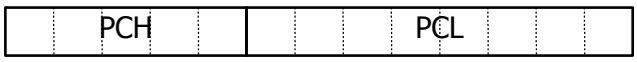
GIE : ce bit permet de valider ou d'interdire (globalement) toutes les interruptions

I.14 L'adressage indirect

L'adressage indirect se fait par l'intermédiaire des registres FSR et INDF. Le registre INDF n'est pas un vrai registre mais représente la case mémoire pointée par le registre d'index FSR. Pour lire ou écrire dans une case mémoire en utilisant l'adressage indirect, on commence par placer l'adresse dans le registre FSR, ensuite on lit/écrit dans le registre INDF

I.15 Le conteur programme

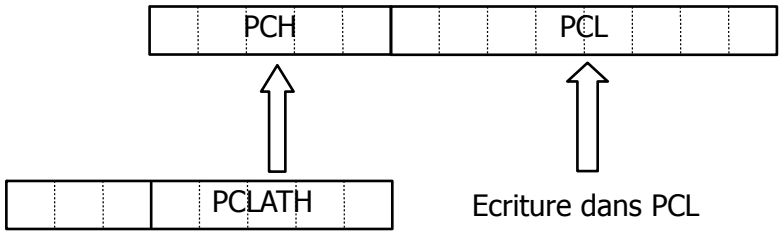
Le Program Counter est un registre de 13 bits qui s'incrémente automatiquement lors de l'exécution du programme. On peut toutefois le modifier par programme pour réaliser ce qu'on appelle un goto calculé. On y accède par les registres PCL et PCLATH



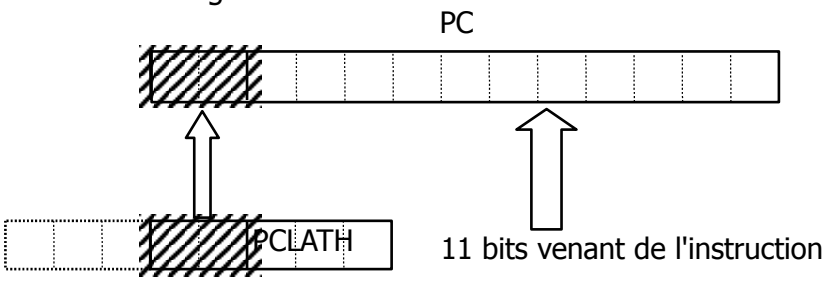
- **PCL** (8 bits) est la partie basse de PC, il est accessible en lecture écriture
- **PCH** (5 bits) est la partie haute de PC, il n'est pas accessible directement. On peut toutefois le modifier indirectement à l'aide du registre PCLATH qui est une registre SFR accessible en lecture écriture et où seuls 5 bits sont utilisés.

I.15.1 GOTO calculé

Si on veut modifier le Program Counter pour réaliser un saut, il faut d'abord placer la partie haute dans le registre PCLATH, ensuite on écrit la partie basse dans PCL. Au moment de l'écriture dans PCL, le contenu de PCLATH est recopié automatiquement dans PCH



Dans les instructions de branchement, l'adresse de destination est codée sur 11 bits. Lors de l'exécution de telles instruction, les 11 bits sont copiés dans PC les deux bits manquants sont pris dans PCLATH. Pour le 16F84, On n'aura pas besoin de ces bits car pour adresser 1024 lignes de programme, seuls 10 bits du Programme Counter sont utilisés.



I.16 Les indicateurs

Les indicateurs C, DC, et Z sont des bits qui nous informent sur le résultat d'une instruction. Ils sont situés dans le registre STATUS :

- **C (Carry) :** ce bit Il passe à "1" lorsque le résultat d'une opération dépasse la valeur FF ou si le résultat est négatif.
- **DC (Digital Carry) :** ce bit passe à "1" lorsque une retenue s'est produite entre les bit 3 et 4.
- **Z (Zero) :** Ce bit passe à "1", pour indiquer que le résultat de l'opération est nul.

I.17 Les instructions du 16F84

Tous les PICs Mid-Range ont un jeu de 35 instructions. Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (OC) ainsi que l'opérande. A part les instructions de saut, toutes les instructions sont exécutées en un cycle d'horloge. Sachant que l'horloge fournie au PIC est prédivisée par 4, si on utilise par exemple un quartz de 4MHz, on obtient donc 1000000 cycles/seconde, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d' Instructions Par Seconde). Avec une horloge de 20MHz, on obtient une vitesse de traitement plus qu'honorable.

I.17.1 Les instructions « orientées octet » (adressage direct)

Ce sont des instructions qui manipulent les données sous forme d'octets. Elles sont codées de la manière suivante :

- 6 bits pour l'instruction : logique, car comme il y a 35 instructions, il faut 6 bits pour pouvoir les coder toutes

- 1 bit (d) pour indiquer si le résultat obtenu doit être conservé dans le registre de travail (accumulateur) W de l'unité de calcul (W pour Work) ou sauvé dans un registre F (F pour File).

- Reste 7 bits pour encoder l'adresse de l'opérande (128 positions au total)

Problème ! 7 bits ne donnent pas accès à la mémoire RAM totale, donc voici l'explication de la division de la RAM en deux banks. Pour remplacer le bit manquant, on utilise le bit RP0 du registre STATUS.

Bien qu'on ne l'utilise pas sur le 16F84, le bit RP1 est aussi réservé pour le changement de bank, le 16F876 par exemple possède 4 banks.

I.17.2 Les instructions « orientées bits »

Ce sont des instructions destinées à manipuler directement les bits d'un registre d'une case mémoire. Elles sont codées de la manière suivante :

- 4 bits pour l'instruction
- 3 bits pour indiquer le numéro du bit à manipuler (de 0 à 7)
- 7 bits pour indiquer l'opérande.

I.17.3 Les instructions opérant sur une donnée (adressage immédiat)

Ce sont les instructions qui manipulent des données qui sont codées dans l'instruction directement. Elles sont codées de la manière suivante :

- L'instruction est codée sur 6 bits
- Elle est suivie d'une valeur IMMEDIATE codée sur 8 bits (donc de 0 à 255).

I.17.4 Les instructions de saut et appel de procédures

Ce sont les instructions qui provoquent une rupture dans la séquence de déroulement du programme. Elles sont codées de la manière suivante :

- Les instructions sont codées sur 3 bits
- La destination est codée sur 11 bits

Nous pouvons déjà en déduire que les sauts ne donnent accès qu'à 2K de mémoire programme (2^{11}). Pas de problème pour le 16F84 qui ne possède que 1k de mémoire programme.

I.17.5 Exemples d'instruction

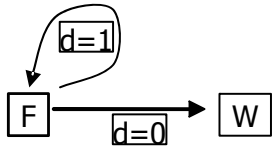
MOVWF F ; recopie W dans le registre d'adresse F : [W] → [F]

F (File) désigne l'adresse de n'importe quel registre SFR ou GPR. Pour les registres SFR, on peut utiliser leurs noms à condition d'inclure le fichier p16F84.inc dans le programme

```
MOVWF    0x2C      ; recopie W dans la case mémoire d'adresse 2C_h
MOVWF    EEDATA    ; recopie W dans le registre EEDATA
MOVF     0x08      ; recopie W dans le registre EEDATA
```

MOVF F,d ; recopie le registre F soit dans W soit dans lui-même

Recopier un registre sur lui-même peut paraître sans intérêt, mais comme l'instruction positionne les indicateurs, cela peut s'avérer intéressant



I.17.6 Le jeu d'instructions

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

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

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

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

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

I.17.7 Etat de quelque registre à l'initialisation

STATUS IRP		RP1	RP0	TO	PD	Z	DC	C	000x xxxx
OPTION_REG RBPU		INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111
INTCON GIE		EEIE	T0IF	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
EECON1 -		-	-	EEIF	WRERR	WREN	WR	RD	- - -0 x000

TRISA	- - -1 1111
TRISB	1111 1111
PORTA	- - -x xxxx
PORTB xxxx	xxxx

II LES OUTILS DE DEVELOPPEMENT

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

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

Microchip propose gratuitement l'outil de développement **MPLAB** qui regroupe l'éditeur de texte, le compilateur MPASM, un outil de simulation et le logiciel de programmation. Le programmateur lui-même, n'est malheureusement pas gratuit.

Pour ce qui nous concerne, nous utiliseront MPLAB pour écrire, compiler et éventuellement simuler nos programmes, ensuite nous utiliserons un programmateur fait maison pour implanter les programmes dans la mémoire flash du PIC. Moi j'utilise le programmateur JDM avec le logiciel ICPROG, les deux sont disponibles gratuitement sur le Web.

II.1 Deux mot sur MPLAB

MPLAB peut être trouvé sur les CD distribués par Microchip ou téléchargé directement du site Web http://www.microchip.com

Nous allons réaliser un tout petit programme sans grand intérêt pour voir la procédure de fonctionnement (avec MPLAB 6.30)

- Debugger → Select tool → MPLAB SIM (à faire une fois après installation de MPLAB)
- Configure → Select Device → PIC16F64A
- Ouvrir une nouvelle fenêtre (de l'éditeur) pour commencer à écrire un programme : **file new** ou cliquez sur l'icône feuille blanche

- Taper le petit programme ci-dessous dans la fenêtre qui vient de s'ouvrir. Ce programme incrémente sans fin la position mémoire (RAM) $0C_H$

```
loop incf    0x0C,1
 goto         looop
 end
```

- Sauvegarder (file save) ce programme dans la directory de votre chois sous le nom bidon.asm

- Lancer la compilation du programme à l'aide de la commande **project Quikbuild**
  Apparemment il y a un problème, le compilateur nous dit qu'il y une erreur à la ligne 2 :
  Error[113] C:\...\BIDON.ASM 2 : Symbol not previously defined (looop)
  Evidemment, le label loop définit dans la ligne précédente prend seulement deux o. Corrigez et recommencez. Cette fois ça a l'air d'aller. On peut vérifier que le compilateur à crée le fichier bidon.hex dans la même directory où se trouve bidon.asm. Les fichiers bidon.cod, bidon.err et bidon.lst ne nous servent à rien pour l'instant on peut les détruire.

- Nous pouvons maintenant exécuter notre programme en simulation pour voir s'il réalise bien la tache demandée :

- Ouvrez la fenêtre qui visualise la mémoire RAM : view FileRegisters. La case mémoire 0x0C se trouve sur la première ligne (ligne:0000, colonne:0C)

- Exécuter maintenant le programme PAS à PAS en cliquant à chaque fois sur le bouton Step Into { } en observant la case mémoire 0C . (on dirait que ça marche).

- On peut aussi exécuter en continu en cliquant sur le bouton animate , pour arrêter, il faut cliquer sur le bouton halt

Pour plus de détail, consulter le manuel d'utilisation de MPLAB

II.2 Les directives de MPASM

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

II.2.1 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 certain 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. ça peut être une constante s'il s'agit d'une instruction avec adressage immédiat, ou d'une adresse s'il s'agit d'une instruction avec adressage direct.

- **#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
  ```

- **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 :
                                          RTLW 'P'
                                          RTLW 'r'
                                          RTLW 'o'
                                          . . .
                                          RTLW 'C'
                                          RTLW 23
                                          RTLW 0x47
```

- **END** : indique la fin du programme

Pour plus de détail sur les directives de MPASM, voir "MPASM USER'S GUIDE"

II.3 Format des nombres

L'assembleur reconnaît les nombres en décimal, hexadécimal, binaire ou octal. Pour préciser la base il faut utiliser les préfixes précisés dans le tableau ci-dessous :

On peut à l'aide de la directive LIST ou RADIX définir un format par défaut. Si par exemple on place une des instructions suivantes au début du programme, tous les nombres sans préfix seront interprétés en décimal :

```
LIST r = dec
RADIX dec
```

(les radix valables sont dec, hex ou oct)

Base	Préfixe	Exemple (36)
Décimal	D'nnn'	D'36'
	.nnn	**.**36
Hexadécimal	H'nn'	H'24'
	0xnn	0x24
	nnh	24h
Binaire	B'….'	B'00100100'
Octal	O'nnn'	O'44'

II.4 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 l'instructions
  - la colonne de droite contient des commentaires

- Il existe différentes écoles indiquant comment doit être organisé un programme. Voici un exemple d'organisation :

1) Quelques lignes de commentaire précisant la fonction du programme,

2) Configuration, exemple :

```
LIST        p=16f84, f=inhx8m, r = dec
INCLUDE     "p16f84.inc"
__CONFIG H'3FF9'
```

3) Définition des constantes et des variables, exemple :

```
led equ 0
x equ               0x0C
cblock 0x0D
   y,z
   u,v,w
endc
```

4) Si le programme utilise des interruptions, mettre à l'adresse 0000 (adresse du RESET) une instruction de branchement au début du programme principal :

```
org 0
goto debut
```

5) Ecrire la routine d'interruption à l'adresse 4

```
ORG 4
écrire la routine d'interruption ici
RETFIE
```

Si le programme est configuré pour interdire les interruptions, on peut se passer des étapes 4) et 5),

6) Ecrire les sous programmes (s'il y en a). Chaque procédure commence par une étiquette qui représente son nom, et se termine par l'instruction RETURN

7) Ecrire le programme principal (commençant par l'étiquette début: si les étapes 4 et 5 sont présentes)

8) terminer avec la directive END

II.5 Exemples 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, f=inhx8m, r = dec
                INCLUDE     "p16f84A.inc"
                __CONFIG    _CP_OFF  &  _XT_OSC &   _PWRTE_OFF   & _WDT_OFF

                bsf         STATUS,RP0      ; bank 1
 movlw                      B'00000001'
                movwf       TRISB           ; pour configurer RB0 en entrée
                bcf         STATUS,RP0      ; bank 0

tst btfss                   PORTB,0
 goto                       off
 bsf                        PORTB,1
 goto                       tst
off bcf                     PORTB,1
 goto                       tst

 end

;*****************************************************************************************************************************************
;  programme led-tmr0-1.asm
;  faire clignoter une  LED connectée sur une sortie du port B, la temporisation permettant d'ajuster la fréquence
; est obtenue par  scrutation des débordement du timer TMR0
;*****************************************************************************************************************************************
                LIST                        p=16f84A, f = inhx8m, r = dec
                __CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF
 INCLUDE                    "p16f84A.inc"

CTR equ                     0x0C

                bsf         STATUS,RP0      ; bank 1 (pour TRISB et OPTION_REG)
                clrf        TRISB           ; PORTB en sortie
 movlw                      B'00000111'
                movwf       OPTION_REG      ; PSA=0, prédiviseur affecté à TMR0, PS1 PS2 PS3 = 111, div = 256
                                            ; T0CS=0, horloge TMR0 = fosc/4/div
                bcf         STATUS,RP0      ; retour à bank 0

encore:         comf        PORTB,f         ; complémenter PORTB
                call        delay           ; attendre un peu
 goto                       encore          ; recommencer

delay:          movlw       5               ; pour attendre que TMR0 déborde 5 fois
                movwf       CTR             ; ce qui donne  5 x 256 x 256 µs
tst: btfss                  INTCON,T0IF     ; attendre que TMR0 déborde
 goto                       $-1
                bcf         INTCON,T0IF     ; baisser le drapeau
                decfsz      CTR,f           ; pour recommencer CTR fois
 goto                       tst
 return

 end
```

```
;*****************************************************************************************************************
;  programme led-tmr0-2.asm
;  faire clignoter une  LED connectée sur une sortie du port B. La temporisation permettant d'ajuster la
;  fréquence est obtenue en comptant les débordements du timer TMR0 à l'interieur de l'interruption T0I
;  TMR0 est utilisé en timer avec un prédiviseur de 256. En comptant 5 débordement on obtient une
;  temporisation de 4 x 256 x 256 µs
;*****************************************************************************************************************

 list            p=16f84,f=inhx8m,r=dec
            __config  _PWRTE_OFF & _CP_OFF & _WDT_OFF & _XT_OSC
 #include          "p16f84.inc"

CTR         equ    0x0C                ; varible de comptage

; ================= démarrage sur RESET
            org    0
            goto   start

;================= procedure d'interruption
 org                4
            bcf     INTCON,T0IF     ; baisser le drapeau levé par l'interruption
 decfsz             CTR,f
 retfie
            comf    PORTB,f         ; changer l'état de la LED
            movlw   5               ; initializer compteur
            movwf   CTR
 retfie

;================= Programme principal
start       bsf     STATUS,RP0      ; select bank1
            clrf    TRISB           ; programme tous les bits du bort B en sortie
            movlw   B'00000111'     ; mode timer, prédiviseur pour TMR0, div=256
 movwf              OPTION_REG
            bcf     STATUS,RP0      ; select bank0
            movlw   B'10100000'     ; autorisation Interruption T0I
 movwf              INTCON
            movlw   5               ; initialise CTR pour le premier passage
            movwf   CTR
Loop        goto    Loop            ; le PIC reste planté ici et n'en sort que pour aller
                                    ; executer une interuption due au débordement de TMR0
 end
```

```
;*****************************************************************************************************************
;
; Clignotement d'une LED reliée à la sortie 0 du port B. Les autres bits du port B ne sont pas affectés
; La temporisation est réalisée à l'aide du Watchdog timer
;*****************************************************************************************************************
;
            list p=16f84, f=inhx8m, r = dec
            __config  _PWRTE_OFF & _CP_OFF & _WDT_ON & _XT_OSC
#include              "p16f84A.inc"

            bsf       STATUS,RP0      ; select bank1
            bcf       TRISB,0           ; RB0 en sortie
            movlw   B'00001101'       ; prescaler affecté au WDT, prescaler = 101 = 32
            movwf     OPTION_REG      ; débordement du WDT tous les 32 x 18ms = 0.576 s
            bcf       STATUS,RP0      ; select bank0
            movlw     1                 ; bit 0 de W à 1, le autres à 0,

Loop:       sleep                       ; passe en mode sleep, réveil dans 0.576 s
            xorwf     PORTB,f           ; complémente le bit 0 de PORTB
            goto    Loop                ; recommence la loupe indéfinitivement

 end


;*************************************************************************
;
; Clignotement d'une LED reliée à n'importe quelle sortie du port B. La temporisation est réalisée à l'aide d'une boucle de
; retard à base de 3 boucles imbriquées N1 (256), N2 (256), N3 (à préciser)
; T1 : (N1-1) x (1+2) + 2 => N1=0=256 => T1 = 767 µs
; T2 : T1 + (N2-1) x (1+2+T1) + 2, N2=0 => T2 = 196352 µs
; T3 : T2+(N3-1) x (1+2+T2) + 2 + 2 = N3 x 196355 + 1 µs  (je croix)
;*************************************************************************
;
            list p=16f84, f=inhx8m, r=dec
#include              "p16f84A.inc"
            __config _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF

N1          equ       0x0D            ; N1,N2,N3 = compteurs temporisateur
N2 equ                0x0E
N3 equ                0x0F
;================================Programme principal
            bsf       STATUS,RP0      ; select bank1
            clrf      TRISB             ; programme tous les bits du bort B en sortie
            bcf       STATUS,RP0      ; select bank0

loop:       comf      PORTB,f           ; complémente PORTB
            movlw     3                 ; 3 donne une tempo voisine de 0.6 s
            call      tempo
            goto      loop              ; recommence la loupe indéfinitivement

;================================Procédure de temporisation
tempo:      movwf     N3                ; copier W dans N3
tmp         decfsz    N1,f              ; boucle intérieure
 goto                 tmp
            decfsz    N2,f              ; boucle médiane
 goto                 tmp
            decfsz    N3,f              ; boucle extérieure
 goto                 tmp
 return

 end
```

II.6 Références

[1] PIC16F8X, document DS30430C, www.microchip.com
[2] PIC16F84a, document DS35007A, www.microchip.com
[3] Programmation des PIC, Première partie-PIC16F84-Révision 5, par BIGONOFF, http://www.abcelectronique.com/bigonoff/organisation.php?2654c