

## I.1 Introduction :

Les microcontrôleurs sont des composants nécessaires pour la conception des circuits électroniques. Dans ce chapitre nous allons expliquer les caractéristiques des différentes parties constituant un microcontrôleur PIC 16F84A (l'organisation de la mémoire, jeu d'instructions et comment programmer un pic...).

## I.2 LES MICROCONTRÔLEURS :

### I.2.1 Qu'est ce qu'un microcontrôleur ?

Les **Pics** sont des microcontrôleurs chez Microchip à architecture RISC (*Reduced Instruction Construction Set*), ou encore composants à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide; ce qui augmente la vitesse de fonctionnement du microcontrôleur.

La famille des PICs est subdivisée en 3 grandes familles :

- **La famille Basse - Line** : Les instructions sont codées sur 12 bits.
- **La famille Mid - Range** : Les instructions sont codées sur 14 bits (et dont font partie les 16F84A).
- **la famille High - End** : Les instructions sont codées sur 16 bits.

Les PICs sont des composants STATIQUES, ils peuvent fonctionner avec des fréquences d'horloge allant du continu jusqu'aux fréquences supérieures spécifiques à chaque circuit (*Le PIC16F84A par exemple peut fonctionner avec une horloge allant du continu jusqu'à 10 MHZ*). Nous limiterons dans ce chapitre à la famille Mid – Range et particulièrement PIC16F84A.

Ses caractéristiques principales sont :

- Séparation des mémoires de programme et de données (architecture Harvard). On obtient ainsi une meilleure bande passante et des instructions et des données pas forcément codées sur le même nombre de bits.
- Communication avec l'extérieur seulement par des ports : il ne possède pas de bus d'adresses, de bus de données et de bus de contrôle comme la plupart des microprocesseurs.
- Utilisation d'un jeu d'instructions réduit, d'où le nom de son architecture : RISC (*Reduced Instructions Set Construction*). Les instructions sont ainsi codées sur un nombre réduit de bits, ce qui accélère l'exécution (1 cycle machine par instruction sauf pour les sauts qui requièrent 2 cycles). En revanche, leur nombre limité oblige à se restreindre à des

instructions basiques, contrairement aux systèmes d'architecture CISC (Complex Instructions Set Construction) qui proposent plus d'instructions donc codées sur plus de bits mais réalisant des traitements plus complexes.

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

- **xx** : Famille du composant (12, 14, 16, 17, 18).
- **L** : Tolérance plus importante de la plage de tension.
- **XX** : Type de mémoire de programme:
  - C**: EPROM au EEPROM.
  - CR**: PROM.
  - F**: FLASH.
- **yy**: Identification.
- **zz** : Vitesse maximum du quartz.

Nous utiliserons un PIC 16F84 –10, soit :

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

## I.2.2 Intérêt des microcontrôleurs :

Les microcontrôleurs sont de taille tellement réduite qu'ils peuvent être sans difficulté implantés sur l'application même qu'ils sont censés piloter. Leurs prix et leurs performances simplifient énormément la conception de système électronique et informatique. L'utilisation des microcontrôleurs ne connaît de limite que l'ingéniosité des concepteurs, on les trouve dans nos cafetières, les magnétoscopes, les radios .....Une étude menée en l'an 2004 montre qu'en moyenne, un foyer américain héberge environ 240 microcontrôleurs.

## I.3 PRESENTATION GENERALE DU PIC 16F84A :

### I.3.1 Classification du PIC 16F84A :

Le PIC 16F84A est un microcontrôleur 8 bits. Il dispose donc d'un bus de données de huit bits. Puisqu'il traite des données de huit bits, il dispose d'une mémoire de donnée dans laquelle chaque emplacement (défini par une adresse) possède huit cases pouvant contenir chacune un bit.

### I.3.2 Architecture interne :

La structure générale du PIC 16F84A comporte 4 blocs comme le montre la figure (I.1):

- Mémoire de programme.
- Mémoire de données.
- Processeur.
- Ressources auxiliaires (périphériques).

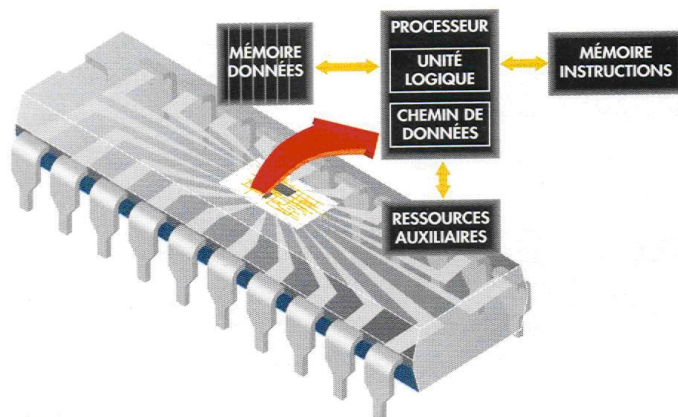


Figure (I.1): Architecture interne.

- La mémoire de programme contient les instructions pilotant l'application à laquelle le microcontrôleur est dédié. Il s'agit d'une mémoire non volatile (elle garde son contenu, même en l'absence de tension), elle est de type FLASH c'est à dire qu'elle peut être programmée et effacée par l'utilisateur via un programmeur et un PC. La technologie utilisée permet plus de 1000 cycles d'effacement et de programmation. Pour le PIC 16F84A cette mémoire est d'une taille de 1024\*14 bits, c'est à dire qu'elle dispose de 1024 emplacements (de 000h à 3FFh) contenant chacun 14 cases car dans le cas du PIC, les instructions sont codées sur 14 bits. On peut donc stocker 1024 instructions.
- La mémoire de donnée est séparée en deux parties :
  1. Une mémoire RAM de 68 octets puisque le bus de donnée est de huit bits. Cette RAM est volatile (les données sont perdues à chaque coupure de courant). On peut y lire et écrire des données.
  2. Une mémoire EEPROM de 64 octets dans laquelle on peut lire et écrire des données (de huit bits soit un octet) et qui possède l'avantage d'être non volatile (les données sont conservées même en l'absence de tension). La lecture et l'écriture dans cette

mémoire de données sont beaucoup plus lentes que dans la mémoire de données RAM.

- Le processeur est formé de deux parties:
  1. Une unité arithmétique et logique (UAL) chargée de faire des calculs.
  2. Un registre de travail noté W sur lequel travail l'UAL.
- Les ressources auxiliaires qui sont dans le cas du PIC16F84A :
  1. Ports d'entrées et de sorties.
  2. Temporisateur.
  3. Interruption.
  4. Chien de garde.
  5. Mode sommeil.

## I.4 STRUCTURE INTERNE DU PIC 16F84A:

### I.4.1 Brochage et caractéristiques principales :

Le PIC16F84A est un circuit intégré de 18 broches comme la figure (I.2):

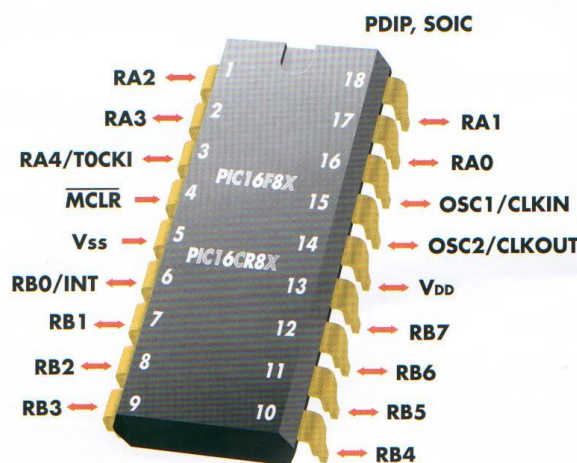


Figure (I.2): Brochage du PIC16F84A.

- L'alimentation du circuit est assurée par les pattes VDD et VSS. Elles permettent à l'ensemble des composants électroniques du PIC de fonctionner. Pour cela on relie VSS (patte 5) à la masse (0 Volt) et VDD (patte 14) à la borne positive de l'alimentation qui doit délivrer une tension continue comprise entre 3 et 6 Volts.
- Le microcontrôleur est un système qui exécute des instructions les unes après les autres à une vitesse (fréquence) qui est fixée par une horloge interne au circuit. Cette horloge doit

être stabilisée de manière externe au moyen d'un cristal de quartz connecté aux pattes OSC1/CLKIN (patte 16) et OSC2/CLKOUT (patte 15).

- La patte 4 est appelée MCLR. Elle permet lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur. C'est à dire que si un niveau bas (0 Volt) est appliqué sur MCLR le microcontrôleur s'arrête, place tout ses registres dans un état connu et se redirige vers le début de la mémoire de programme pour recommencer le programme au début (adresse dans la mémoire de programme : 0000). A la mise sous tension, la patte MCLR étant à zéro, le programme démarre donc à l'adresse 0000, (MCLR=Master Clear Reset).
- Les broches RB0 à RB7 et RA0 à RA4 sont les lignes d'entrées/sorties numériques. Elles sont au nombre de 13 et peuvent être configurées en entrée ou en sortie. Ce sont elles qui permettent au microcontrôleur de dialoguer avec le monde extérieur (périphériques). L'ensemble des lignes RB0 à RB7 forme le port B et les lignes RA0 à RA4 forment le port A. Certaines de ces broches ont aussi d'autres fonctions (interruption, timer).

### I.4.2 Structure interne :

La structure interne du PIC16F84A est donnée figure (I.3): (structure HARVARD: la mémoire de programme et la mémoire de données sont séparées contrairement à l'architecture Von Neumann qui caractérise d'autre fabricant de microcontrôleur).

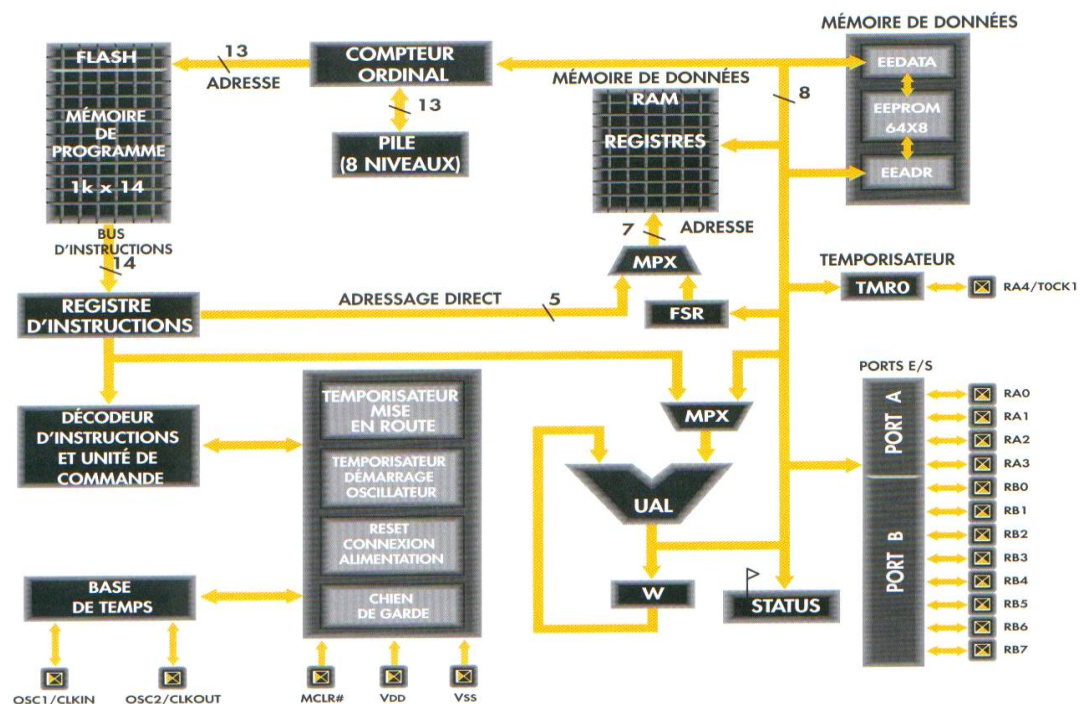


Figure (I.3): Structure interne du PIC16F84A.

On retrouve sur ce schéma la mémoire de programme, la mémoire RAM de données, la mémoire EEPROM, les ports A et B, ainsi que la partie processeur avec l'UAL et le registre de travail W (work). Nous allons étudier à présent plus en détail le fonctionnement du PIC.

### I.4.3 Principe de fonctionnement du PIC :

Un microcontrôleur exécute des instructions. On définit « le cycle instruction » comme le temps nécessaire à l'exécution d'une instruction. Attention de ne pas confondre cette notion avec le cycle d'horloge qui correspond au temps nécessaire à l'exécution d'une opération élémentaire (soit un coup d'horloge).

Une instruction est exécutée en deux phases :

- La phase de la recherche du code binaire de l'instruction stocké dans la mémoire de programme.
- La phase d'exécution ou le code de l'instruction est interprété par le processeur et exécuté.

Chaque phase dure 4 cycles d'horloge comme le montre la figure (I.4):

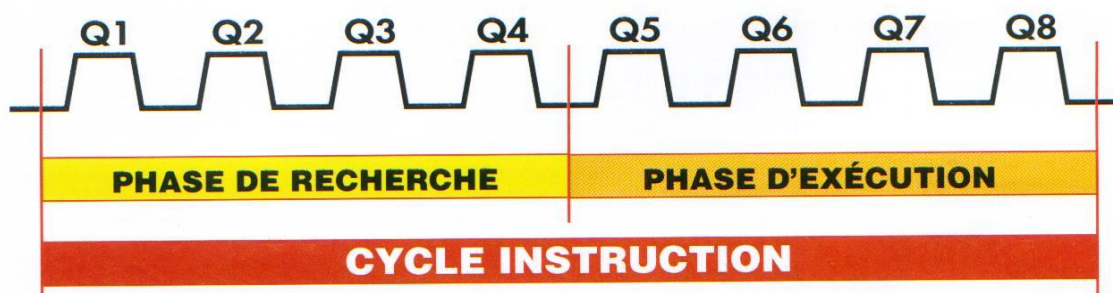


Figure (I.4): Le cycle instruction (4 cycles d'horloge).

On pourrait donc croire qu'un cycle instruction dure 8 cycles d'horloge mais l'architecture particulière du PIC lui permet de réduire ce temps par deux. En effet, comme les instructions issues de la mémoire de programme circulent sur un bus différent de celui sur lequel circulent les données, ainsi le processeur peut effectuer la phase de recherche d'une instruction pendant qu'il exécute l'instruction précédente (Voir la figure (I.5) et (I.6)).



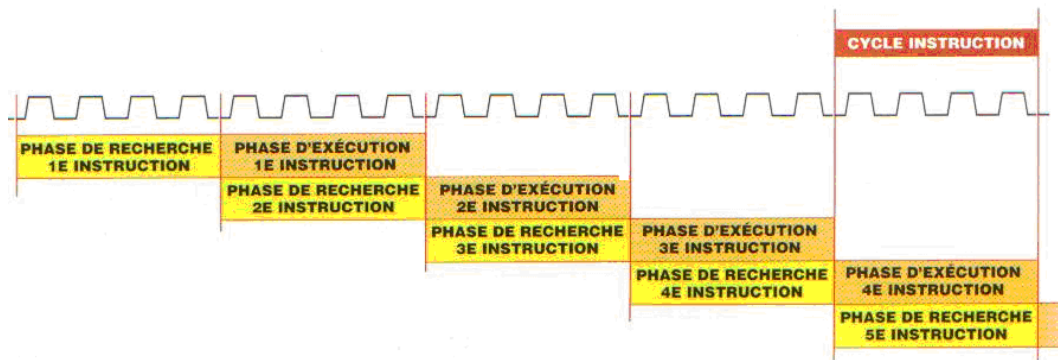


Figure (I.5): Le cycle instruction.

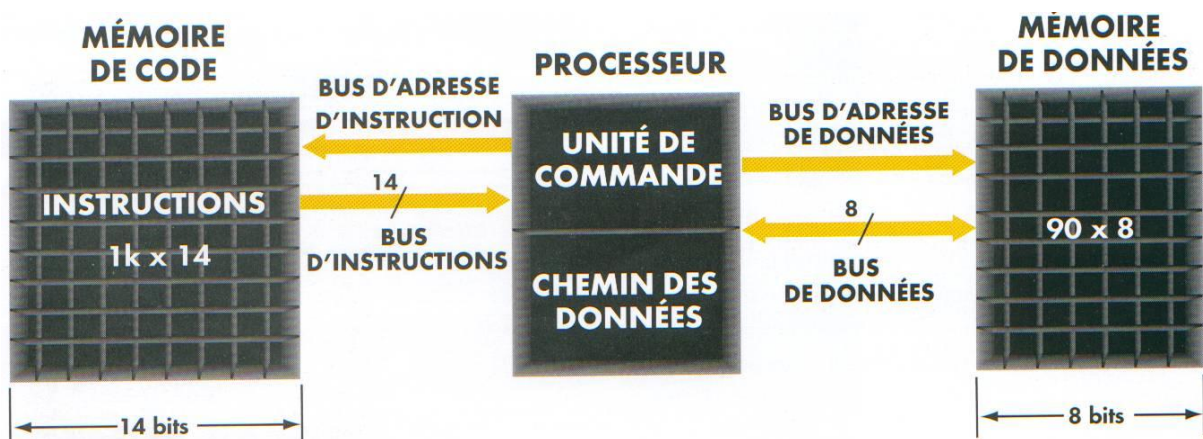


Figure (I.6): Le cycle instruction.

#### I.4.4 Déroulement d'un programme :

Le déroulement d'un programme s'effectue de façon très simple. A la mise sous tension, le processeur va chercher la première instruction qui se trouve à l'adresse 0000 de la mémoire de programme, l'exécute puis va chercher la deuxième instruction à l'adresse 0001 et ainsi de suite (sauf cas de saut ou d'appel de sous programme que nous allons voir plus loin). On parle de fonctionnement séquentiel. La figure (I.7) va nous permettre de mieux comprendre le fonctionnement :

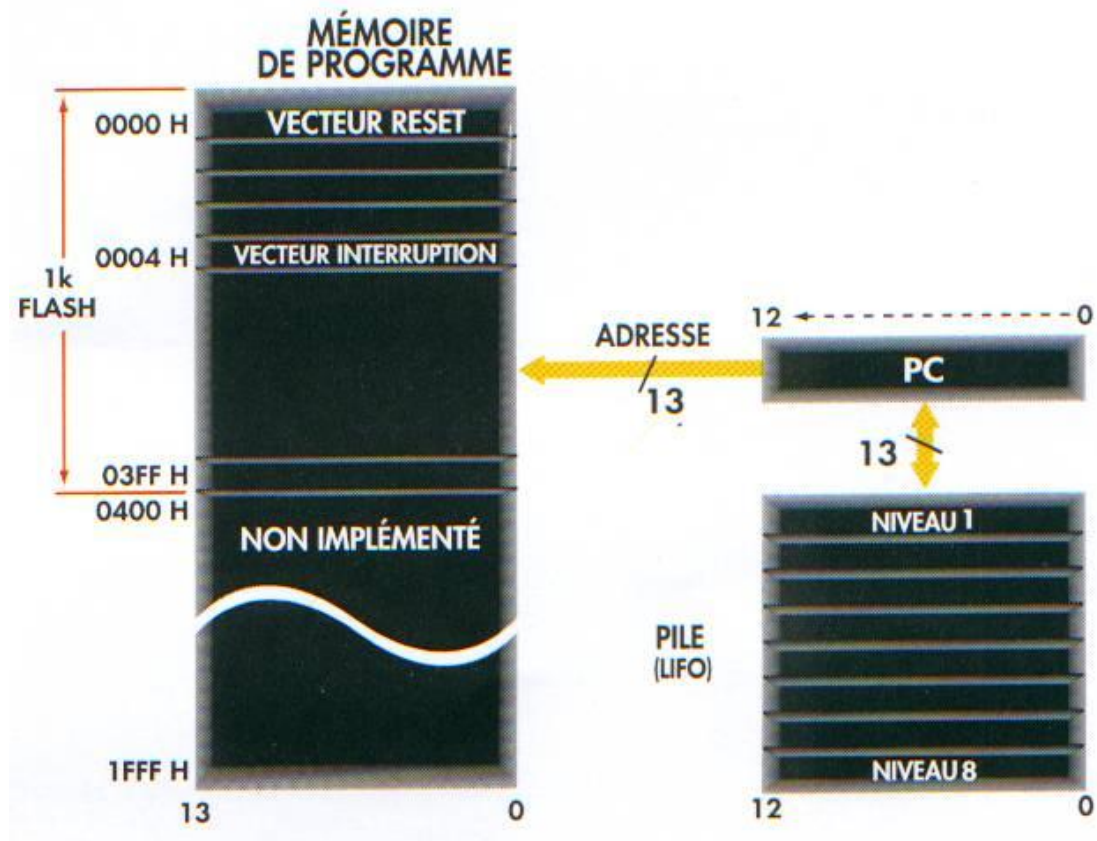


Figure (I.7): Déroulement d'un programme.

- On constate sur cette figure que la mémoire de programme contient 1024 emplacements (3FF en hexadécimale) contenant 14 bits (de 0 à 13). Une instruction occupe un emplacement qui est défini par une adresse. Le processeur peut alors sélectionner l'emplacement souhaité grâce au bus d'adresse et il peut lire son contenu (ici l'instruction) grâce à son bus d'instruction (voir la figure (I.8)). Cet adressage s'effectue à l'aide d'un compteur ordinal appelé PC qui lors de la mise sous tension démarre à zéro puis s'incrémente de 1 tous les quatre coups d'horloge, on exécute bien ainsi les instructions les unes à la suite des autres.
- Mais il arrive que dans un programme on fasse appel à un sous programme dont l'adresse de l'instruction ne se trouve pas juste après celle qui est en train d'être exécutée. C'est le rôle de la pile qui sert à emmagasiner de manière temporaire l'adresse d'une instruction. Elle est automatiquement utilisée chaque fois que l'on appelle un sous programme et elle permet une fois que l'exécution du sous programme est terminée de retourner dans le programme principal juste après l'endroit où l'on a appelé le sous programme. On constate que cette pile possède huit niveaux, cela signifie qu'il n'est pas possible d'imbriquer plus de huit sous programmes, car au-delà de huit, le processeur ne sera plus capable de retourner à l'adresse de base du programme principal.



- L'adresse 0000 est réservée au vecteur RESET, cela signifie que c'est à cette position que l'on accède chaque fois qu'il se produit une réinitialisation (0 volts sur la patte MCLR). C'est pour cette raison que le programme de fonctionnement du microcontrôleur doit toujours démarrer à cette adresse.
- L'adresse 0004 est assignée au vecteur d'interruption et fonctionne de manière similaire à celle du vecteur de Reset. Quand une interruption est produite et validée, le compteur ordinal PC se charge avec 0004 et l'instruction stockée à cet emplacement est exécutée.

### **I.4.5 La mémoire de données RAM:**

Si l'on regarde la mémoire de donnée RAM, on s'aperçoit que celle-ci est un peu particulière comme le montre la figure (I.8):

On constate en effet que cette mémoire est séparée en deux pages (page 0 et page 1). De plus, on remarque que tant pour la page 0 que pour la page 1, les premiers octets sont réservés (SFR pour Special File Register). Ces emplacements sont en effet utilisés par le microcontrôleur pour configurer l'ensemble de son fonctionnement.

Le bus d'adresse qui permet d'adresser la RAM est composé de 7 fils ce qui veut dire qu'il est capable d'adresser 128 emplacements différents. Or, chaque page de la RAM est composée de 128 octets, le bus d'adresse ne peut donc pas accéder aux deux pages, c'est pourquoi on utilise une astuce de programmation qui permet de diriger le bus d'adresse soit sur la page 0, soit sur la page 1. Cela est réalisé grâce à un bit d'un registre spécifique (le bit RP0 du registre STATUS) dont nous verrons le fonctionnement plus loin.

La RAM de données proprement dite se réduit donc à la zone notée GPR (Registre à usage générale) qui s'étend de l'adresse 0Ch (12 en décimale) jusqu'à 4Fh (79 en décimale) soit au total 68 registres en page 0 et autant en page 1, mais on constate que les données écrites en page 1 sont redirigées en page 0 cela signifie qu'au final l'utilisateur dispose uniquement de 68 registres (donc 68 octets de mémoire vive) dans lesquels il peut écrire et lire à volonté en sachant qu'à la mise hors tension, ces données seront perdues.

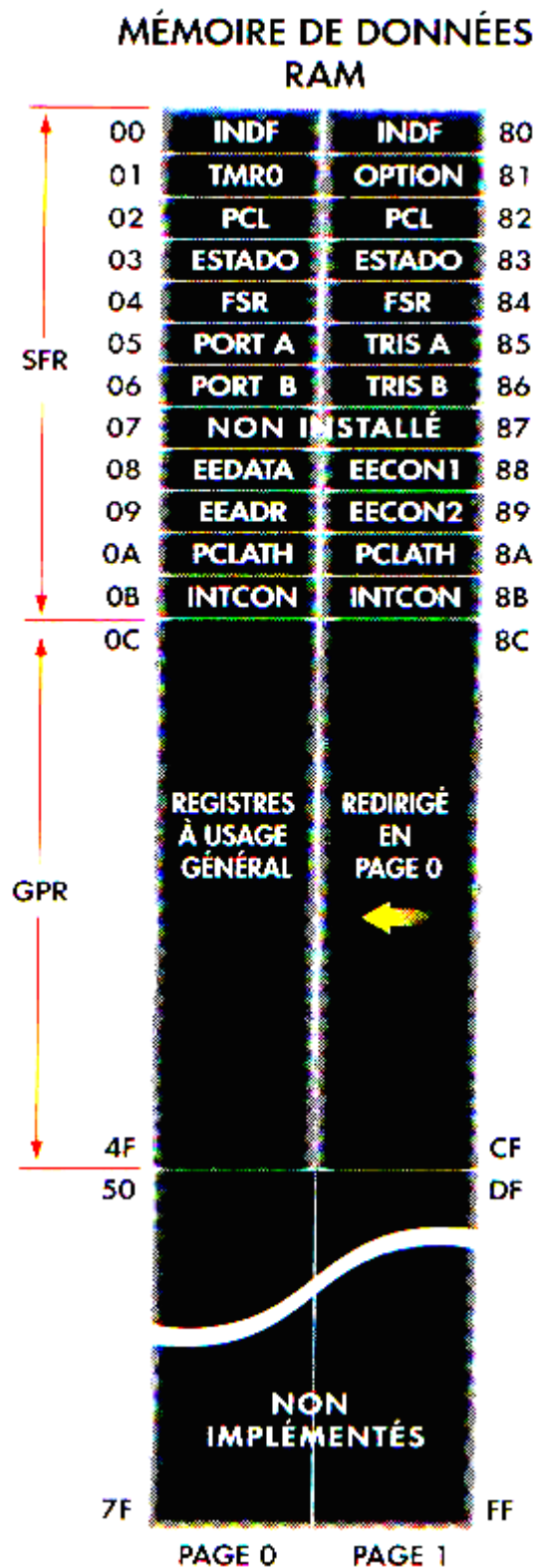


Figure (I.8): La mémoire de données (RAM).

### I.4.6) Les registres:

Nous avons vu dans le paragraphe précédent que la mémoire de données RAM contenait des registres spécifiques qui permettent de configurer le PIC, nous allons les détailler un à un et voir comment on peut accéder à la page 0 ou la page 1. Afin de faciliter la compréhension, les registres les plus utilisés sont encadrés.

- **Adresse 00 et 80, INDF:** Cette adresse ne contient pas de registre physique, elle sert pour l'adressage indirect.
- **Adresse 01, TMR0:** Contenu du Timer (8 bits). Il peut être incrémenté par l'horloge ( $f_{osc}/4$ ) c'est à dire tous les 4 coups d'horloge ou par la broche RA4.
- **Adresse 02 et 82, PCL:** 8 bits de poids faibles du compteur ordinal PC. Les 5 (13-8) bits de poids forts sont dans PCLATH.
- **Adresse 03 et 83, STATUS Registre d'état:**

Les cinq bits de poids faible de ce registre sont en lecture seule, ce sont des témoins (drapeaux ou flag en anglais) caractérisant le résultat de l'opération réalisée par l'UAL. Le bit RP0 est lui en lecture / écriture et c'est lui qui permet de sélectionner la page dans la mémoire RAM.

Si RP0=0 on accède à la page 0 et si RP0=1 on accède à la page 1.

RP0	TO/	PD/	Z	DC	C
-----	-----	-----	---	----	---

Au reset, seul le bit RP0 de sélection de page est fixé (RP0=0 : page 0).

TO/ (Time Out): débordement du timer WDT.

PD/ (Power Down): caractérise l'activité du chien de garde WDT.

Z (zéro): résultat nul pour une opération arithmétique et logique.

DC (digit carry): retenue sur un quartet (4 bits).

C (carry): retenue sur un octet (8 bits).

- **Adresse 04 et 84, FSR (Registre de sélection de registre):** contient l'adresse d'un autre registre (adressage indirect).
- **Adresse 05, PORTA:** Ce registre contient l'état des lignes du port A.
- **Adresse 06, PORTB:** Ce registre contient l'état des lignes du port B.

- **Adresse 08, EEDATA:** Contient un octet lu ou à écrire dans l'EEPROM de données.
- **Adresse 09, EEADR:** Contient l'adresse de la donnée lue ou écrite dans l'EEPROM de données.
- **Adresse 0A et 8A, PCLATH:** Voir l'adresse 02 PCL.
- **Adresse 0B et 8B:** INTCON.

Contrôle des 4 interruptions.

GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
-----	------	------	------	------	------	------	------

### Masque :

**GIE** (Global Interrupt Enable): masque global d'inter.

**EEIE** (EEPROM Interrupt Enable): autorise l'interruption venant de l'EEPROM.

**TOIE** (Timer 0 Interrupt Enable): autorise l'interruption provoquée par le débordement du TIMER0.

**INTE** (Interrupt Enable): autorise l'interruption provoquée par un changement d'état sur broche RB0/INT.

**RBIE** (RB Interrupt Enable) : autorise les interruptions provoquées par un changement d'états sur l'une des broches RB4 à RB7.

Si ces bits sont mis à 1, ils autorisent les interruptions pour lesquels ils sont dédiés.

### Drapeaux :

**TOIF** (Timer 0 Interrupt Flag): débordement du Timer.

**INTF** (Interrupt Flag): interruption provoquée par la broche RB0/INT.

**RBIF** (RB Interrupt Flag): interruption provoquée par les broches RB4-RB7.

### ➤ Adresse 81, OPTION:

8 bits (tous à 1 au RESET) affectant le comportement des E/S et des Timers.

RBPU/	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0
-------	--------	-----	-----	-----	-----	-----	-----

RBPU/ (RB Pull Up) Résistances de tirage à Vdd des entrées du port B. Si RBPU/=0 les résistances de pull-up sont connectées en interne sur l'ensemble du port B.

**INTEDG** (Interrupt Edge): sélection du front actif de l'interruption sur RB0/INT (1 pour front montant et 0 pour front descendant).

**RTS** (Real Timer Source): sélection du signal alimentant le timer 0 : 0 pour horloge interne, 1 pour RA4/T0CLK.

**RTE** (Real Timer Edge): sélection du front actif du signal timer (0 pour front montant).

**PSA** (Prescaler assignment): 0 pour Timer 0 et 1 pour chien de garde WDT.

**PS2...0** (Prescaler 210): sélection de la valeur du diviseur de fréquence pour les timers.

- **Adresse 85, TRISA:** Direction des données pour le port A : 0 pour sortir et 1 pour entrer.
- **Adresse 86, TRISB:** Direction des données pour le port B : 0 pour sortir et 1 pour entrer.
- **Adresse 88, EECON1:** Contrôle le comportement de l'EEPROM de données.

<b>EEIF</b>	<b>WRERR</b>	<b>WREN</b>	<b>WR</b>	<b>RD</b>
-------------	--------------	-------------	-----------	-----------

**EEIF** (EEPROM Interrupt Flag): passe à 1 quand l'écriture est terminée.

**WRERR** (Write Error): 1 si erreur d'écriture.

**WREN** (Write Enable): 0 pour interdire l'écriture en EEPROM de données.

**WR** (Write): 1 pour écrire une donnée. Bit remis automatiquement à 0.

**RD** (Read): 1 pour lire une donnée. Bit remis automatiquement à 0.

- **Adresse 89, EECON2:** Registre de sécurité d'écriture en EEPROM de données.

Une donnée ne peut être écrite qu'après avoir écrit successivement 0x55 et 0xAA dans ce registre.

#### I.4.7) Les ports d'entrées/sorties:

Le PIC16F84A est équipé de 13 lignes d'entrées/sorties réparties en deux ports :

-le port A : RA0 à RA4.

-le port B : RB0 à RB7.

Chaque ligne peut être configurée soit en entrée, soit en sortie, et ceci indépendamment l'une de l'autre. Pour cela on utilise les registres TRISA et TRISB. Le bit de poids faible (b0) du registre TRISA correspond à la ligne RA0, le bit b1 de TRISA correspond à RA1 et ainsi de

suite. Il en est de même pour le port B et le registre TRISB (b0 de TRISB correspond à RB0 → b7 correspond à RB7). Si l'on veut placer une ligne en sortie il suffit de mettre le bit correspondant dans TRISA ou TRISB à 0 (retenez 0 comme Output=sortie). Si l'on veut placer une ligne en entrée, il suffit de placer le bit correspondant dans TRISA ou TRISB à 1 (retenez 1 comme Input=entrée).

Les bits des deux registres PORTA et PORTB permettent soit de lire l'état d'une ligne si celle-ci est en entrée, soit de définir le niveau logique d'une ligne si celle-ci est en sortie.

Lors d'un RESET, toutes les lignes sont configurées en entrées.

➤ Particularité du PORTA: les bits b7 à b5 des registres TRISA et PORTA ne correspondent à rien car il n'y a que 5 lignes (b0 à b4). RA4 est une ligne à collecteur ouvert, cela veut dire que configurée en sortie cette broche assure 0Volt à l'état bas, mais qu'à l'état haut, il est nécessaire de fixer la valeur de la tension grâce à une résistance de tirage (pull up en anglais).

➤ Particularité du PORTB: Il est possible de connecter de façon interne sur chaque ligne une résistance de tirage (pull up) dont le rôle consiste à fixer la tension de la patte (configuré en entrée) à un niveau haut lorsque qu'aucun signal n'est appliqué sur la patte en question. Pour connecter ces résistances, il suffit de placer le bit RBPU/ du registre OPTION à 0.

### I.4.8 Le Timer:

Dans la majeure partie des applications, il est nécessaire de contrôler le temps; afin de ne pas occuper le microcontrôleur qu'à cette tâche (boucle de comptage qui monopolise le micro), on le décharge en utilisant un timer. Le pic 16F84A dispose de deux timers, un à usage général (le TMR0) et un autre utilisé pour le chien de garde (Watch dog WDG).

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 figure (I.9):



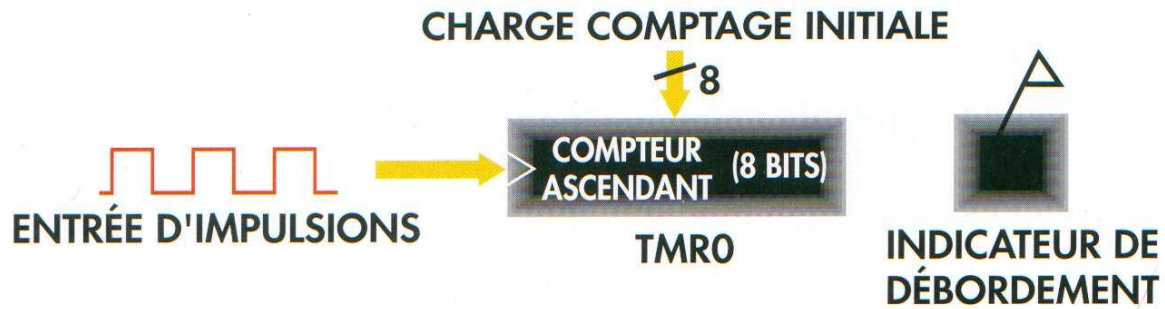


Figure (I.9): Le principe de Timer.

Le TMR0 peut remplir deux fonctions:

- Temporisateur ou contrôle du temps. Son entrée d'incrémentation est alors l'horloge qui correspond au cycle instruction ( $F_{osc}/4$ ). Il est possible d'utiliser un pré-diviseur de fréquence que nous verrons plus loin.
- 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 RTS du registre OPTION.

Le pic 16F84A dispose d'un diviseur de fréquence qui peut être assigné soit au chien de garde, soit au TMR0 (uniquement un à la fois). L'assignation du pré diviseur se fait grâce au bit PSA du registre OPTION. La structure interne du TMR0 est donc la suivante (figure (I.10)):

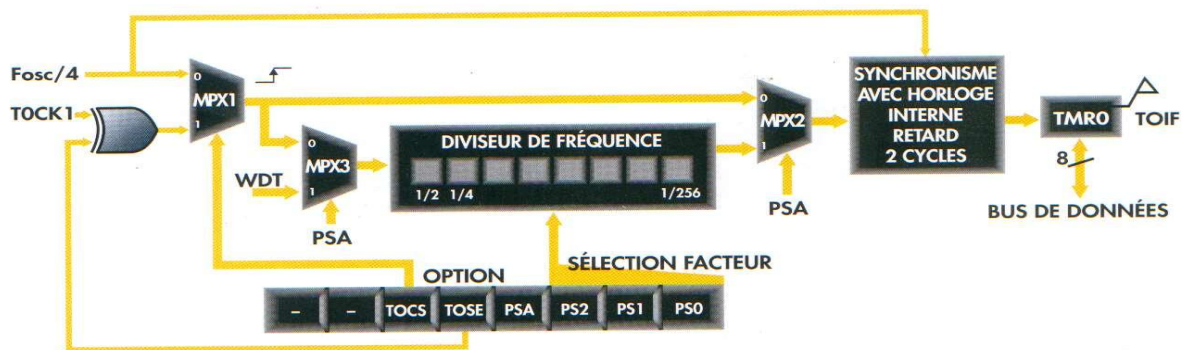


Figure (I.10): Structure interne de TIMER.

**Exemple:**

On veut que le timer nous indique par la mise à un du drapeau T0IF l'écoulement d'une durée de 20ms (la fréquence d'horloge étant de 4MHz) d'où  $F_{osc}/4=1\mu s$ . Si on choisit un pré division de 256, on aura donc  $20000 \mu s / 256 = 78$ . Il ne faut pas charger le TMR0 avec 78 mais avec le complément à deux de cette valeur (car le timer compte et ne décompte pas) d'où  $256-78=178$ . Soit en hexadécimale la valeur **B2h** à charger dans le registre TMR0.

**I.4.9 Mise en œuvre:**

L'utilisation et la mise en œuvre très simple des PICs les a rendus extrêmement populaire au point que la société qui les fabrique (MICROCHIP) est en passe de devenir le leader mondial dans le domaine des microcontrôleurs devant MOTOROLA et INTEL.

Il suffit d'alimenter le circuit par ses deux broches VDD et VSS, de fixer sa vitesse de fonctionnement à l'aide d'un quartz (figure (I.11)) et d'élaborer un petit système pour permettre de réinitialiser le microcontrôleur sans avoir à couper l'alimentation (figure (I.12)).

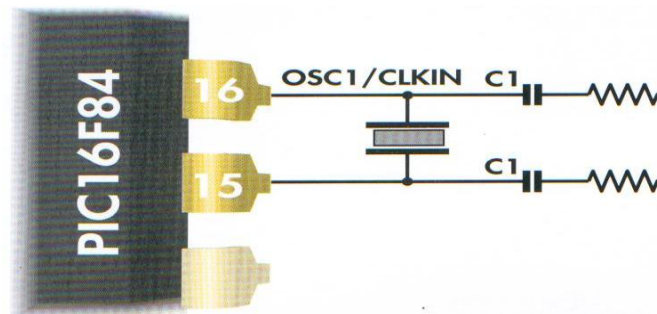


Figure (I.11) : Alimentation du pic 16F84A.

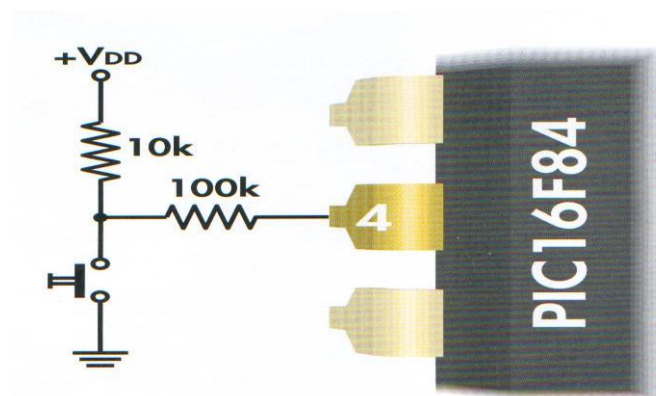


Figure (I.12): Alimentation du pic 16F84A.

Il suffit ensuite d'écrire le programme en langage assembleur ou en C sur un ordinateur grâce au logiciel MPLAB de MICROCHIP (logiciel gratuit) puis de le compiler pour le transformer en langage machine et le transférer dans le PIC grâce à un programmeur.

Il ne nous reste plus qu'à voir le jeu d'instruction de programmation en assembleur du PIC et c'est là que réside tout l'intérêt puisqu'il ne dispose que de 35 instructions qui lui permettent de réaliser toutes les tâches.

## I.5 JEU D'INSTRUCTIONS :

Afin de comprendre la fonction de chaque instruction, la notation adoptée pour les données et adresses manipulées par les instructions est fort simple et est la suivante :

- **f** représente un registre.
- **b** représente un numéro de bit en sachant que 0 correspond toujours au bit de poids faible (le plus à droite dans le registre).
- **k** représente une donnée aussi appelée littérale.

Un certain nombre d'instructions (ADDWF, ANDWF, etc..) utilise une notation spéciale présentée sous la forme :

**ADDWF      f, d ;**      Où **f** indique le registre et où **d** peut prendre deux valeurs (0 ou 1), ce qui change le comportement de l'instruction. Si **d** est à 0, le résultat est placé dans le registre de travail **W**, la valeur dans le registre **f** est alors inchangée, alors que si **d** est à 1, le résultat est placé dans le registre **f**.

Un autre type d'instruction mérite quelques éclaircissements, ce sont les instructions de branchement conditionnel. Prenons comme exemple:

**BTFSC      f, b ;**      Qui va vouloir dire (Bit Test File Skip if Clear) qui signifie que l'on va tester le bit **b** du registre **f** (**b** peut prendre une valeur de 0 à 7 pour un registre 8 bits). Il peut alors y avoir deux solutions:

- Soit le bit testé est à 1, donc la condition testée n'est pas réalisée, le programme continue alors son déroulement normalement en séquence avec l'instruction juste en dessous.

- Soit le bit testé vaut 0, donc la condition testée est réalisée et le programme saute l'instruction qui suit le **BTFSC** dans le programme.

Cette façon de programmer peut paraître étrange, mais avec de l'habitude, elle s'avère très pratique et permet de réaliser des programmes compacts et performants.

Les 35 instructions sont donc les suivantes :

### 1) ADDLW (Add Literal to W):

Syntaxe :     ADDLW     k

Opération :     $W+k \rightarrow W$ .

Bits d'état du registre STATUS affectée : C, DC, Z.

On ajoute au registre de travail la valeur k et on place le résultat dans le registre de travail W.

Durée : 1 cycle instruction (4 cycles d'horloge).

### 2) ADDWF (Add W to F):

Syntaxe :     ADDWF     f, d

Opération :     $W+f \rightarrow f$  si d=1 ou  $W+f \rightarrow W$  si d=0.

Bits d'état du registre STATUS affectés : C, DC, Z.

On ajoute le contenu de W et le contenu de f et on place le résultat dans f si d=1 ou dans W si d=0.

Durée : 1 cycle instruction (4 cycles d'horloge).

### 3) ANDLW (And Literal and W):

Syntaxe : ANDLW    k

Opération :     $W \text{ ET } k \rightarrow W$ .

Bit d'état du registre STATUS affecté: Z.

On effectue un ET logique entre le contenu de W et le littéral k, on place le résultat dans W.

Durée: 1 cycle instruction (4 cycles d'horloge).

### 4) ANDWF (And W with F):

Syntaxe:     ANDWF     f, d

Opération :     $W \text{ ET } f \rightarrow f$  si d=1 ou  $W \text{ ET } f \rightarrow W$  si d=0.

Bit d'état du registre STATUS affecté : Z.

On effectue un ET logique entre le contenu de W et le contenu de f, on place le résultat dans W si d=0 ou dans f si d=1.

Durée: Cycle instruction (4 cycles d'horloge).

#### 5) BCF (Bit Clear F):

Syntaxe:        BCF            f, b

Opération:      $0 \rightarrow b(f)$ .

Bits d'état du registre STATUS affectés : aucun.

On met à 0 le bit b du registre f.

Durée: 1 cycle instruction (4 cycles d'horloge).

#### 6) BSF (Bit Set F):

Syntaxes:       BSF            f, b

Opération:      $1 \rightarrow b(f)$ .

Bits d'état du registre STATUS affectés: aucun.

On met à 1 le bit b du registre f.

Durée: 1 cycle instruction (4 cycles d'horloge).

#### 7) BTFSC (Bit Test , Skip if Clear):

Syntaxe:        BTFSC        f, b

Opération: saut de l'instruction qui suit si  $b(f)=0$ .

Bits d'état du registre STATUS affectés: aucun.

Si le bit b de f est nul, l'instruction qui suit celle-ci est ignorée et traitée comme un NOP. Dans ce cas et dans ce cas seulement, l'instruction BTFSC demande deux cycles pour s'exécuter.

Durée: 1 cycle instruction (4 cycles d'horloge) ou 2 cycles.

#### 8) BTFSS (Bit Test , Skip if Set):

Syntaxe :        BTFSS        f, b

Opération :     saut de l'instruction qui suit si  $b(f)=1$ .

Bits d'état du registre STATUS affectés: aucun.

Si le bit b de f est à 1, l'instruction qui suit celle-ci est ignorée et traitée comme un NOP. Dans ce cas et dans ce cas seulement, l'instruction BTFSS demande deux cycles pour s'exécuter.

Durée: 1 cycle instruction (4 cycles d'horloge) ou 2 cycles.

#### 9) CALL (subroutine Call):

Syntaxes:       CALL           label

Bits d'état du registre STATUS affectés: aucun.

On sauvegarde l'adresse de retour dans la pile puis on appelle le sous programme définit avec l'étiquette label.

Durée: 2 cycles instruction (8 cycles d'horloge).

**10) CLRF (Clear F):**

Syntaxes: CLRF f

Opération :  $0 \rightarrow F$ .

Bit d'état du registre STATUS affecté : Z.

On met le contenu du registre f à 0 et on positionne Z.

Durée: 1 cycle instruction (4 cycles d'horloge).

**11) CLRW (Clear W):**

Syntaxes: CLRW

Opération :  $0 \rightarrow W$ .

Bit d'état du registre STATUS affecté: Z.

On met le contenu du registre W à 0 et on positionne Z.

Durée: 1 cycle instruction (4 cycles d'horloge).

**12) CLRWD (Clear Watchdog Timer):**

Syntaxes: CLRWD

Opération:  $0 \rightarrow WDT$  et  $0 \rightarrow$  pré diviseur du Timer.

On met le contenu du registre du timer chien de garde à 0 ainsi que le pré diviseur.

Durée: 1 cycle instruction (4 cycles d'horloge).

**13) COMF (Complement F):**

Syntaxe: COMF f, d

Opération:  $f \rightarrow f$  si  $d=1$  ou  $f \rightarrow W$  si  $d=0$ .

Bit d'état du registre STATUS affecté: Z.

On complémente le contenu du registre f bit à bit, le résultat est placé dans f si  $d=1$ , dans W si  $d=0$ .

Durée: 1 cycle instruction (4 cycles d'horloge).

**14) DECF (Decrement F):**

Syntaxe: DECF f, d

Opération:  $f-1 \rightarrow f$  si  $d=1$  ou  $f-1 \rightarrow W$  si  $d=0$ .

Bit d'état du registre STATUS affecté : Z.

On diminue le contenu du registre f d'une unité, le résultat est placé dans f si  $d=1$ , dans W si  $d=0$ .

Durée: 1 cycle instruction (4 cycles d'horloge).

**15) DECFSZ (Decrement F ,Skip if Zero):**

Syntaxe: DECFSZ f, d

Opération:  $f-1 \rightarrow f$  si  $d=1$  ou  $f-1 \rightarrow W$  si  $d=0$  et saut si  $f-1=0$ .

Bit d'état du registre STATUS affecté: aucun.



On diminue le contenu du registre f d'une unité, le résultat est placé dans f si d=1, dans W si d=0. Si le résultat est nul, l'instruction suivante est ignorée et dans ce cas, cette instruction dure deux cycles.

Durée: 1 cycle instruction (4 cycles d'horloge) ou 2 cycles.

#### 16) GOTO (branchement inconditionnel):

Syntaxe:       GOTO       label

Bit d'état du registre STATUS affecté: aucun.

On effectue un saut dans le programme pour aller à l'adresse pointé par le label précisé dans GOTO.

Durée: 2 cycles (8 cycles d'horloge).

#### 17) INCF (Increment F):

Syntaxe :       INCF       f, d

Opération :    f+1→f si d=1 ou f+1→W si d=0.

Bit d'état du registre STATUS affecté: Z.

On augmente le contenu du registre f d'une unité, le résultat est placé dans f si d=1, dans W si d=0.

Durée: 1 cycle instruction (4 cycles d'horloge).

#### 18) INCFSZ (Increment F, Skip if Zero):

Syntaxe:       INCFSZ       f, d

Opération:    f+1→f si d=1 ou f+1→W si d=0 et saut si f-1=0.

Bit d'état du registre STATUS affecté: aucun.

On augmente le contenu du registre f d'une unité, le résultat est placé dans f si d=1, dans W si d=0. Si le résultat est nul, l'instruction suivante est ignorée et dans ce cas, cette instruction dure deux cycles.

Durée: 1 cycle instruction (4 cycles d'horloge) ou 2 cycles.

#### 19) IORLW ( Inclusive Or Literal with W ):

Syntaxe:       IORLW       k

Opération:    W OU k → W.

Bit d'état du registre STATUS affecté : Z.

On effectue un OU logique entre le contenu de W et le littéral k, le résultat est placé dans W.

Durée : 1 cycle instruction (4 cycles d'horloge).

#### 20) IORWF ( Inclusive Or W with F ):

Syntaxe:       IORWF       f, d

Opération:    W OU f→f si d=1 ou W OU f→W si d=0.

Bit d'état du registre STATUS affecté: Z.

On effectue un OU entre le contenu de W et le contenu de f, on place le résultat dans f si d=1, dans W si d=0.

Durée: 1 cycle instruction (4 cycles d'horloge).

### 21) MOVF ( Move F ):

Syntaxes:      MOVF          f, d

Opération:     $f \rightarrow f$  si d=1 ou  $f \rightarrow W$  si d=0.

Bit d'état du registre STATUS affecté: Z.

On déplace le contenu de f dans f si d=1 ou de f dans W si d=0. Attention, le déplacement de f dans f semble à priori inutile, mais il permet en fait de tester le contenu de f par rapport à 0 et de positionner le bit Z.

Durée: 1 cycle instruction (4 cycles d'horloge).

### 22) MOVLW ( Move Literal to W ):

Syntaxe:      MOVLW      k

Opération:     $k \rightarrow W$ .

Bit d'état du registre STATUS affecté: aucun.

On charge le contenu de W avec le littéral k.

Durée: 1 cycle instruction (4 cycles d'horloge).

### 23) MOVWF ( Move W to F ):

Syntaxe:      MOVWF      f

Opération:     $W \rightarrow f$ .

Bit d'état du registre STATUS affecté: aucun.

On charge le contenu de f avec le contenu de W.

Durée: 1 cycle instruction (4 cycles d'horloge).

### 24) NOP (No Operation):

Syntaxe:      NOP

Opération:    néant.

Bit d'état du registre STATUS affecté: aucun.

On ne fait que consommer du temps machine (un cycle dans ce cas).

Durée: 1 cycle instruction (4 cycles d'horloge).

### 25) RETFIE ( Return From Interrupt ):

Syntaxes:      RETFIE

Opération:    Pile  $\rightarrow$  PC.

Bit d'état du registre STATUS affecté: aucun.

On charge le compteur ordinal avec la valeur qui se trouve au sommet de la pile pour revenir au programme principal lorsque l'exécution du sous programme est terminée.

Durée : 2 cycles instruction (8 cycles d'horloge).

## 26) RETLW ( Return Literal to W ):

Syntaxe: RETLW k

Opération:  $k \rightarrow W$ , Pile  $\rightarrow PC$ .

Bit d'état du registre STATUS affecté : aucun.

On charge le contenu de W avec le littéral k puis on charge le compteur ordinal PC avec la valeur qui se trouve au sommet de la pile effectuent ainsi un retour de sous programme.

Durée: 2 cycles instruction (8 cycles d'horloge).

## 27) RETURN (Return from subroutine):

Syntaxes: RETURN

Opération: Pile  $\rightarrow PC$ .

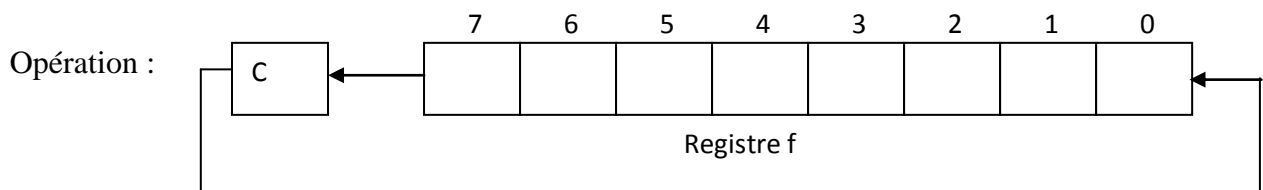
Bit d'état du registre STATUS affecté: aucun.

On charge le compteur ordinal PC avec la valeur qui se trouve au sommet de la pile effectuent ainsi un retour de sous programme. C'est un RETLW simplifié.

Durée: 2 cycles instruction (8 cycles d'horloge).

## 28) RLF (Rotate Left F through carry):

Syntaxe: RLF f, d



Bit d'état du registre STATUS affecté: C.

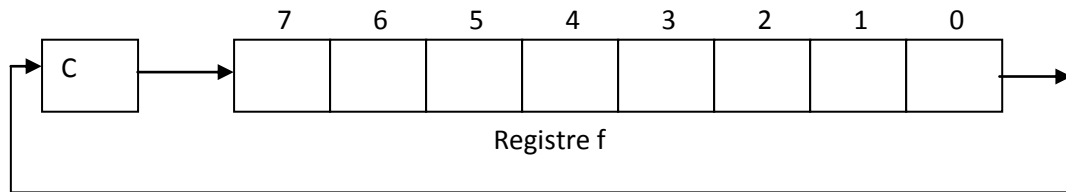
On effectue une rotation à gauche de un bit du contenu du registre f en passant par le bit de retenu C Si d=1 le résultat est placé dans f, si d=0, le résultat est placé dans W.

Durée: 1 cycle instruction (4 cycles d'horloge).

**29) RRF (Rotate Right F through carry):**

Syntaxe:      RRF                  f, d

Opération:



Bit d'état du registre STATUS affecté: C.

On effectue une rotation à droite de un bit du contenu du registre f en passant par le bit de retenu C. Si d=1 le résultat est placé dans f, si d=0, le résultat est placé dans W.

Durée : 1 cycle instruction (4 cycles d'horloge).

**30) SLEEP (Sleep):**

Syntaxe:      SLEEP

Opération:    0→PD, 1→T0, 0→WDT, 0→ pré diviseur.

On place le circuit en mode sommeil avec arrêt de l'oscillateur. Cette commande est à utiliser avec précaution, elle nécessite la connaissance du mode sommeil.

**31) SUBLW ( Subtract W from Literal ):**

Syntaxe:      SUBLW                  k

Opération:    k-W→W.

Bits d'état du registre STATUS affectés: C, DC, Z.

On soustrait le contenu du registre W du littéral k et on place le résultat dans W (soustraction par la méthode du complément à 2).

Durée: 1 cycle instruction (4 cycles d'horloge).

**32) SUBWF ( Subtract W from F ):**

Syntaxe:      SUBWF                  f, d

Opération:    f-W→W si d=0 ou f-W→f si d=1.

Bits d'état du registre STATUS affectés: C, DC, Z.

On soustrait le contenu du registre W du contenu du registre f et on place le résultat dans W si d=0, ou dans f si d=1.

Durée: 1 cycle instruction (4 cycles d'horloge).

**33) SWAPF (Swap F):**

Syntaxe:      SWAPF                  f, d

Opération:    f (0-3) →f (4-7) et f (4-7) →f (0-3) résultat dans W ou f selon d.

Bit d'état du registre STATUS affecté: aucun.

On échange les quatre bits de poids forts avec les quatre bits de poids faibles et on place le résultat dans W si d=0, ou dans f si d=1.

Durée: 1 cycle instruction (4 cycles d'horloge).

#### **34) XORLW ( Exclusive Or Literal with W ):**

Syntaxe:       XORLW               k

Opération:     W OU EXCLUSIF k→W.

Bit d'état du registre STATUS affecté: Z.

On effectue un OU Exclusif entre W et le littéral k, le résultat est placé dans W.

Durée: 1 cycle instruction (4 cycles d'horloge).

#### **35) XORWF (Exclusive Or W with F):**

Syntaxe:       XORWF               f, d

Opération:     W OU EXCLUSIF f→W si d=0 ou W OU EXCLUSIF f→f si d=1.

Bit d'état du registre STATUS affecté: Z.

On effectue un OU Exclusif entre W et le contenu de f, le résultat est placé dans W si d=0, sinon il est placé dans f.

Durée: 1 cycle instruction (4 cycles d'horloge).

## **I.6 Programmation du PIC16F84A:**

Pour programmer n'importe quel type de Pic il faut maîtriser au minimum un langage de programmation développé au microcontrôleur.

De nos jours il existe plusieurs langages faciles à maîtriser, nous allons citer quelques uns:

### **I.6.1 Le langage C:**

Ce langage est universel, rigoureux et efficace. Il permet de tout faire quelque soit le support. Il existe de très nombreux compilateurs pour tous les matériels. Ce langage a pour seul défaut d'être d'une approche difficile pour le débutant qui peut avoir du mal pour décoller. Un programme en C sera transportable facilement, quasiment sans rien changer sur des dizaines de plates formes, il n'y a pas à se préoccuper du processeur qui fera tourner l'application, c'est le compilateur qui fera le travail de traduction (comme d'ailleurs en basic).

## I.6.2 L'assembleur:

Il effraie le débutant par son approche complexe. C'est toutefois le plus proche de la machine, il produit un code compact et rapide mais tout doit être défini à la main. Il est parfait pour les applications en temps réel qui demandent la maîtrise de timing très court. Il est très simulant intellectuellement pour celui qui veut s'impliquer dans des projets complexes.

Il sera souvent utilisé pour créer des primitives très efficaces qui seront appelées par des routines en C. Le C combiné à l'assembleur sera l'outil de base incontournable du programmeur performant. Pour programmer avec l'assembleur il faut connaître l'architecture interne du microprocesseur et pendant la programmation il faut toujours garder sous le coude le jeu d'instruction détaillé. Il est très difficile de travailler efficacement en parallèle sur des familles différentes en assembleur.

Et pour la programmation il faut avoir sous la main un IDE (INTEGRATED DEVELOPPEMENT ENVIRONNEMENT) facile à manipuler comme :

- Le MPLAB intégrant l'assembleur et comme option d'autres langages ainsi que des programmeurs MICROCHIP ou autres comme le célèbre PICSTART.
- Ou bien, des compilateurs C ou basic par exemples.

## I.7 Conclusion:

Ainsi, comme nous avons pu le voir tout au long de ce chapitre, le PIC16F84A, bien que vieillissant un peu, dispose encore de nombreuses ressources. Difficile à appréhender en premier lieu, celui-ci s'impose comme l'élément indispensable pour de nombreux montages, tant par son rapport qualité/prix, que par ses capacités ; son principal handicap étant le nombre limité de ses entrées/sorties.

Il est programmable avec un minimum de difficultés, ce qui fait de lui le microcontrôleur parfait pour les amateurs débutants.