**PIC 16F84**

1. [Qu'est-ce qu'un PIC ? 4](#_TOC_250055)
2. [PIC 16F84 4](#_TOC_250054)
   1. [Brochage et fonction des pattes 5](#_TOC_250053)
   2. [Architecture générale 5](#_TOC_250052)
3. [Organisation de la mémoire 6](#_TOC_250051)
   1. [Mémoire de programme 6](#_TOC_250050)
   2. [Mémoire de données 7](#_TOC_250049)
      1. [Registres généraux 8](#_TOC_250048)
      2. [Registres spéciaux - SFRs 8](#_TOC_250047)
      3. [Mémoire EEPROM 10](#_TOC_250046)
4. [Jeu d'instructions 10](#_TOC_250045)
   1. [Format général 11](#_TOC_250044)
   2. [Exemple d'instruction – le transfert 11](#_TOC_250043)
   3. [Liste des instructions 13](#_TOC_250042)
   4. [Exécution d'un programme – notion de pipe-line 13](#_TOC_250041)
5. [Modes d'adressages 14](#_TOC_250040)
   1. [Adressage immédiat 14](#_TOC_250039)
   2. [Adressage direct 14](#_TOC_250038)
   3. [Adressage indirect 14](#_TOC_250037)
6. [Ports d'entrées/Sorties 15](#_TOC_250036)
   1. [Port A 15](#_TOC_250035)
   2. [Port B 16](#_TOC_250034)
7. [Compteur 17](#_TOC_250033)
   1. [Registre TMR0 17](#_TOC_250032)
   2. [Choix de l'horloge 18](#_TOC_250031)
   3. [Pré-diviseur 18](#_TOC_250030)
   4. [Fin de comptage et interruption 19](#_TOC_250029)
   5. [Registres utiles à la gestion de timer0 19](#_TOC_250028)
8. [Accès à la mémoire EEPORM 19](#_TOC_250027)
   1. [Registres utilisés 19](#_TOC_250026)
   2. [Lecture 20](#_TOC_250025)
   3. [Ecriture 20](#_TOC_250024)
9. [Interruptions 21](#_TOC_250023)
   1. Rappel - Notion de sous-programme 21
   2. [Mécanisme 21](#_TOC_250022)
   3. [Différentes sources d'interruption 22](#_TOC_250021)
   4. [Validation des interruptions 22](#_TOC_250020)
   5. [Séquence de détournement vers le sous-programme d'interruption 23](#_TOC_250019)
   6. [Sauvegarde et restitution du contexte 23](#_TOC_250018)
      1. [Où sauvegarder ces registres ? 24](#_TOC_250017)
      2. [Comment sauvegarder ces registres ? 24](#_TOC_250016)
      3. [Comment restituer ces registres ? 24](#_TOC_250015)
      4. [Résumé 24](#_TOC_250014)
   7. [Reconnaissance de l'interruption active 25](#_TOC_250013)
   8. [Retour au programme initial 25](#_TOC_250012)
10. [Chien de garde 25](#_TOC_250011)
    1. [Principe 25](#_TOC_250010)
    2. [Mise en service 26](#_TOC_250009)
    3. [Gestion 26](#_TOC_250008)
    4. [Choix de la durée 26](#_TOC_250007)
11. [Mode sommeil 26](#_TOC_250006)
    1. [Principe 26](#_TOC_250005)
    2. [Gestion 26](#_TOC_250004)
       1. [Mise en sommeil 26](#_TOC_250003)
       2. [Réveil 26](#_TOC_250002)
12. [Programmation sur site 27](#_TOC_250001)
13. [Bibliographie 29](#_TOC_250000)

# Qu'est-ce qu'un PIC ?

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

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

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

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

Il existe trois familles de PIC :

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

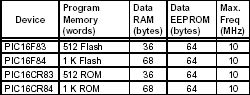
Un PIC est identifié par un numéro de la forme suivant : xx(L)XXyy –zz

* xx : Famille du composant (12, 14, 16, 17, 18)
* L : Tolérance plus importante de la plage de tension
* XX : Type de mémoire de programme C - EPROM ou EEPROM

CR - PROM F - FLASH

* yy : Identification
* zz : Vitesse maximum du quartz Nous utiliserons un PIC 16F84 –10, soit :
* 16 : Mid-Line
* F : FLASH
* 84 : Type
* 10 : Quartz à 10MHz au maximum

# PIC 16F84



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



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

Principales caractéristiques :

* + 35 instructions
  + Instructions codées sur 14 bits
  + Données sur 8 bits
  + 1 cycle machine par instruction, sauf pour les sauts (2 cycles machine)
  + Vitesse maximum 10 MHz soit une instruction en 400 ns (1 cycle machine = 4 cycles d'horloge)
  + 4 sources d'interruption
  + 1000 cycles d'effacement/écriture pour la mémoire flash, 10.000.000 pour la mémoire de donnée EEPROM

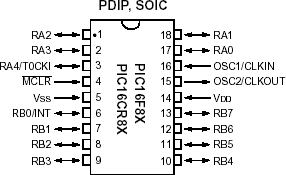
# Brochage et fonction des pattes

La Figure\* II.2 montre le brochage du circuit. Les fonctions des pattes sont les suivantes :

* + - VSS, VDD : Alimentation
    - OSC1,2 : Horloge
    - RA0-4 : Port A
    - RB0-7 : Port B
    - T0CKL : Entrée de comptage
    - INT : Entrée d'interruption
    - MCLR : Reset : 0V

Choix du mode programmation : 12V - 14V

exécution : 4.5V - 5.5V

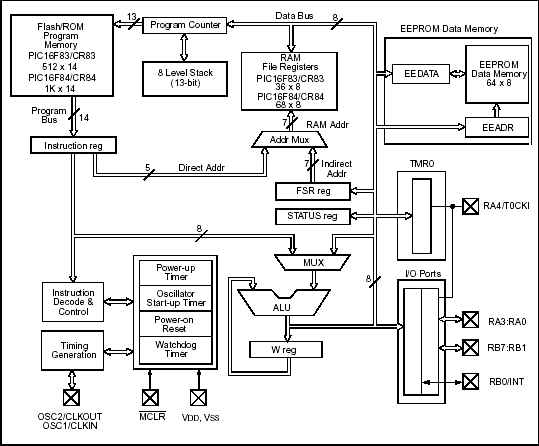


**Figure\* II.2 :** *Brochage du circuit.*

# Architecture générale

La Figure\* II.3 présente l'architecture générale du circuit. Il est constitué des éléments suivants :

* + - un système d'initialisation à la mise sous tension (power-up timer, …)
    - un système de génération d'horloge à partir du quartz externe (timing génération)
    - une unité arithmétique et logique (ALU)
    - une mémoire flash de programme de 1k "mots" de 14 bits (III.1 - XII)
    - un compteur de programme (program counter) et une pile (stack)
    - un bus spécifique pour le programme (program bus)
    - un registre contenant le code de l'instruction à exécuter
    - un bus spécifique pour les données (data bus)
    - une mémoire RAM contenant
      * les SFR (III.2.2)
      * 68 octets de données(III.2.1)
    - une mémoire EEPROM de 64 octets de données (VIII)
    - 2 ports d'entrées/sorties (VI)
    - un compteur (timer) (VII)
    - un chien de garde (watchdog) (X)



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

# Organisation de la mémoire

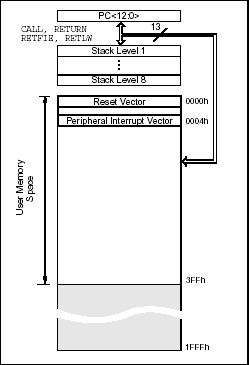
Le PIC contient de la mémoire de programme et de la mémoire de données. La structure Harvard des PICs fournit un accès séparé à chacune. Ainsi, un accès aux deux est possible pendant le même cycle machine.

# Mémoire de programme

C'est elle qui contient le programme à exécuter. Ce dernier est téléchargé par liaison série (voir

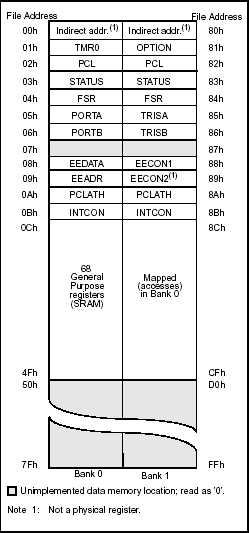
§ XII). La Figure\* III.1 montre l'organisation de cette mémoire. Elle contient 1k "mots" de 14 bits dans le cas du PIC 16F84, même si le compteur de programme (PC) de 13 bits peut en adresser 8k. Il faut se méfier des adresses images ! L'adresse 0000h contient le vecteur du reset, l'adresse 0004h l'unique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

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



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

# Mémoire de données



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

Elle se décompose en deux parties de RAM (Figure\* III.2) et une zone EEPROM. La première contient les SFRs (Special Function Registers) qui permettent de contrôler les opérations sur le circuit. La seconde contient des registres généraux, libres pour l'utilisateur. La dernière contient 64 octets.

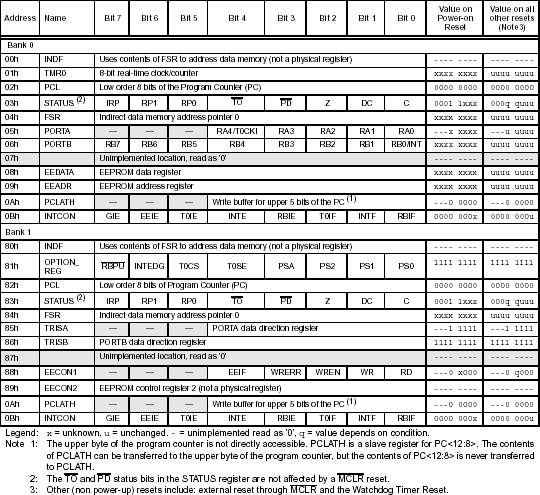
Comme nous le verrons dans le paragraphe IV, les instructions orientées octets ou bits contiennent une adresse sur 7 bits pour désigner l'octet avec lequel l'instruction doit travailler. D'après la Figure\* III.2, l'accès au registre TRISA d'adresse 85h, par exemple, est impossible avec une adresse sur 7 bits. C'est pourquoi le constructeur a défini deux banques. Le bit RP0 du registre d'état (STATUS.5) permet de choisir entre les deux. Ainsi, une adresse sur 8 bits est composée de RP0 en poids fort et des 7 bits provenant de l'instruction à exécuter.

## Registres généraux

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

## Registres spéciaux - SFRs

Ils permettent la gestion du circuit. Certains ont une fonction générale, d'autres un fonction spécifique attachée à un périphérique donné. La Figure\* III.3 donne la fonction de chacun des bits de ces registres. Ils sont situés de l'adresse 00h à l'adresse 0Bh dans la banque 0 et de l'adresse 80h à l'adresse 8Bh dans la banque 1. Les registres 07h et 87h n'existent pas.



**Figure\* III.3 :** *Description des SFR.*

INDF (00h - 80h) : Utilise le contenu de FSR pour l'accès indirect à la mémoire (V.3). TMR0 (01h) : Registre lié au compteur (VII).

PCL (02h - 82h) : Contient les poids faibles du compteur de programmes (PC). Le registre PCLATH (0Ah-8Ah) contient les poids forts.

STATUS (03h - 83h) : Il contient l'état de l'unité arithmétique et logique ainsi que les bits de sélection des banques (Figure\* III.4).

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

PORTA (05h) : Donne accès en lecture ou écriture au port A, 5 bits. Les sorties sont à drain ouvert. Le bit 4 peut être utilisé en entrée de comptage.

PORTB (06h) : Donne accès en lecture ou écriture au port B. Les sorties sont à drain ouvert.

Le bit 0 peut être utilisé en entrée d'interruption.

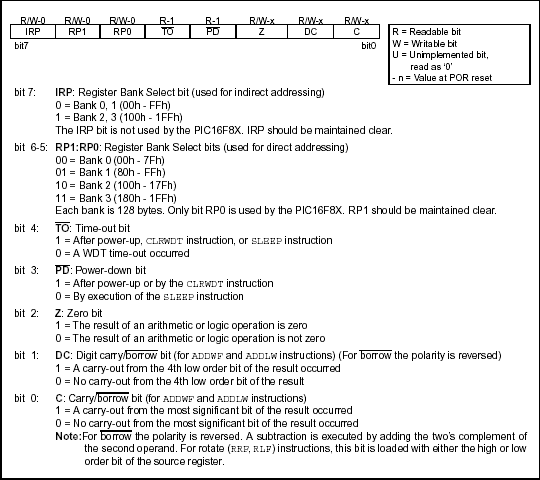
EEDATA (08h) : Permet l'accès aux données dans la mémoire EEPROM. EEADR (09h) : Permet l'accès aux adresses de la mémoire EEPROM.

PCLATCH (0Ah - 8Ah) : Donne accès en écriture aux bits de poids forts du compteur de programme.

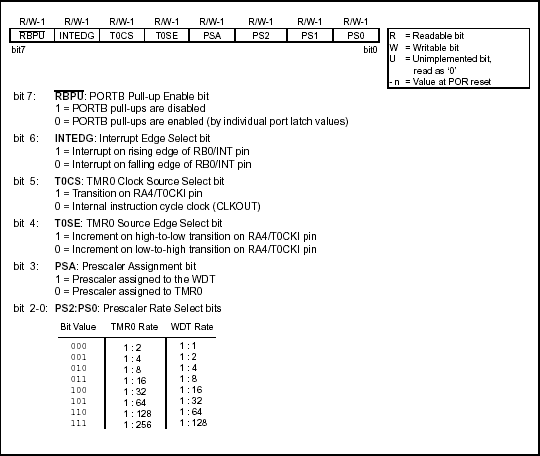
INTCON (0Bh - 8Bh) : Masque d'interruptions (VI).

OPTION\_REG (81h) : Contient des bits de configuration pour divers périphériques. TRISA (85h) : Indique la direction (entrée ou sortie) du port A.

TRISB (86h) : Indique la direction (entrée ou sortie) du port B.

EECON1 (88h) : Permet le contrôle d'accès à la mémoire EEPROM (VIII). EECON2 (89h) : Permet le contrôle d'accès à la mémoire EEPROM (VIII).

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



**Figure\* III.5 :** *Registre de configuration de périphériques - OPTION\_REG.*

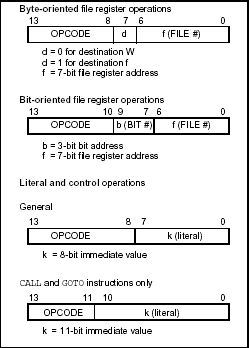
## Mémoire EEPROM

Le PIC possède une zone EEPROM de 64 octets accessibles en lecture et en écriture par le programme. On peut y sauvegarder des valeurs, qui seront conservées même si l'alimentation est éteinte, et les récupérer lors de la mise sous tension. Leur accès est spécifique et requiert l'utilisation de registres dédiés. La lecture et l'écriture ne peut s'exécuter que selon des séquences particulières décrite au paragraphe VIII.

# Jeu d'instructions

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

# Format général



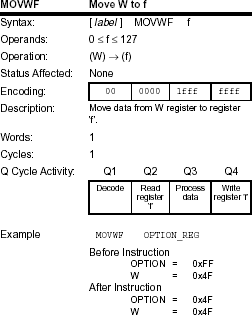
**Figure\* IV.1 :** *Format général d'une instruction.*

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

* + - Instructions orientées octets
    - Instructions orientées bits
    - Instructions de contrôle

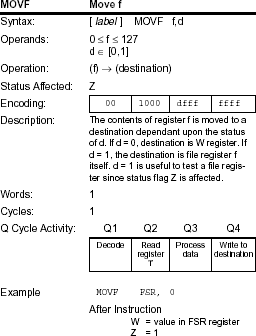
Le registre de travail W joue un rôle particulier dans un grand nombre d'instructions.

# Exemple d'instruction – le transfert



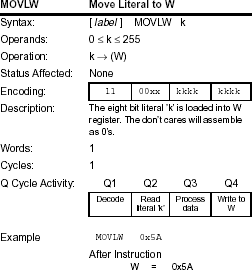
**Figure\* IV.2 :** *Transfert du registre W dans le registre f.*

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



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

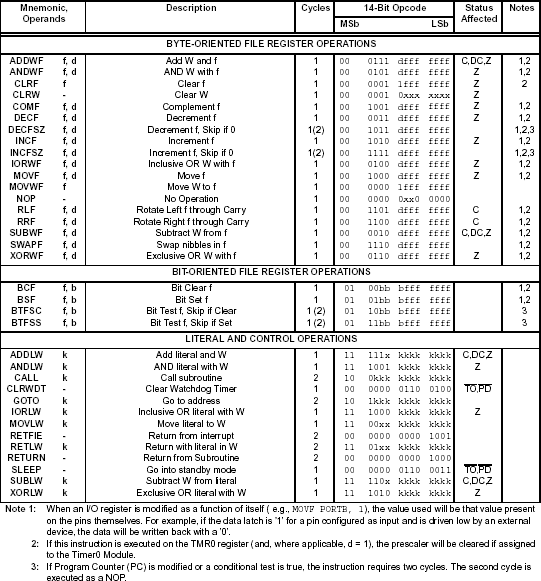
La seconde (Figure\* IV.3) permet de transférer une donnée contenue dans un registre f vers le registre W ou le registre f. Dans ce second cas, l'intérêt est de positionner le bit Z. On peut noter ici le bit 7 qui prend la valeur d fournie dans le code de l'instruction pour choisir la destination : W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



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

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

# Liste des instructions

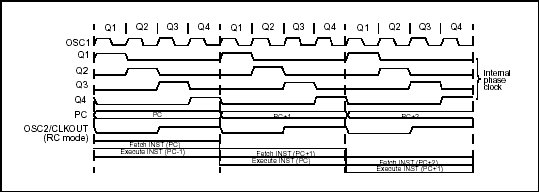


**Figure\* IV.5 :** *Liste des instructions.*

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

# Exécution d'un programme – notion de pipe-line

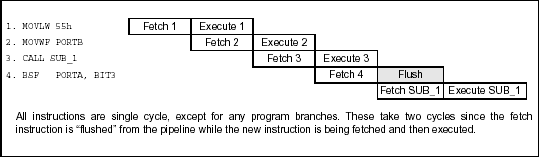
La Figure\* IV.6 montre l'enchaînement des instructions tous les 4 cycles d'horloge. Pendant un premier cycle machine, l'instruction à exécuterest stockée en mémoire RAM. Le cycle suivant, elle est exécutée. Chaque instruction dure donc 2 cycles machine.



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

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

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



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

# Modes d'adressages

On ne peut pas concevoir un programme qui ne manipule pas de données. Il existe trois grands types d'accès à une donnée ou modes d'adressage :

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

pointeur.

# Adressage immédiat

La donnée est contenue dans l'instruction.

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

# Adressage direct

La donnée est contenue dans un registre. Ce dernier peut être par un nom (par exemple W) ou une adresse mémoire.

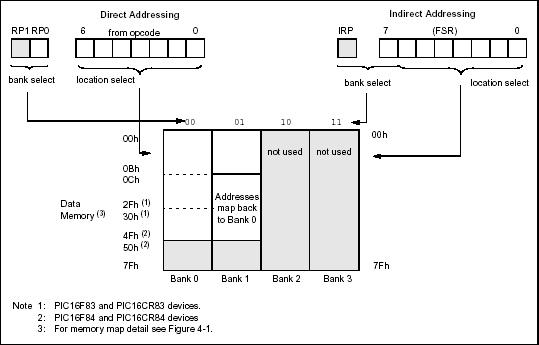
Exemple : movf 0x2B, 0 ; **Transfert dans W** la valeur **contenue à l'adresse 0x2B.**

**!**

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

# Adressage indirect

L'adresse de la donnée est contenue dans un pointeur. Dans les PIC, un seul pointeur est disponible pour l'adressage indirect : FSR. Contenu à l'adresse 04h dans les deux banques, il est donc accessible indépendamment du numéro de banque. En utilisant l'adressage direct, on peut écrire dans FSR l'adresse du registre à atteindre. FSR contenant 8 bits, on peut atteindre les deux banques du PIC 16F84. Pour les PIC contenant quatre banques, il faut positionner le bit IRP du registre d'état qui sert alors de 9ème bit d'adresse (Figure\* V.1).



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

L'accès au registre d'adresse contenue dans FSR se fait en utilisant le registre INDF. Il se trouve à l'adresse 0 dans les deux banques. Il ne s'agit pas d'un registre physique. On peut le voir comme un autre nom de FSR, utilisé pour accéder à la donnée elle-même, FSR servant à choisir l'adresse.

|  |  |  |
| --- | --- | --- |
| Exemple : movlw | 0x1A | ; Charge 1Ah dans W |
| movwf | FSR | ; Charge W, contenant 1Ah, dans FSR |
| movf | INDF, 0 | ; Charge la valeur contenue à l'adresse 1Ah dans W |

# Ports d'entrées/Sorties

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

# Port A

Il comporte 5 pattes d'entrée/sortie bi-directionnelles, notées RAx avec x={0,1,2,3,4} sur le brochage du circuit (Figure\* II.2). Le registre PORTA, d'adresse 05h dans la banque 0, permet d'y accéder en lecture ou en écriture. Le registre TRISA, d'adresse 85h dans la banque 1, permet de choisir le sens de chaque patte (entrée ou sortie) : un bit à 1 positionne le port en entrée, un bit à 0 positionne le port en sortie.

La Figure\* VI.1 donne le câblage interne d'une patte du port A :

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

En lecture : Bloqué.

* + - Transistor P : Permet d'alimenter la sortie.



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

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

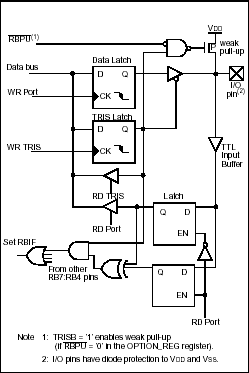
# Port B

Il comporte 8 pattes d'entrée/sortie bi-directionnelles, notées RBx avec x={0,1,2,3,4,5,6,7} sur le brochage du circuit (Figure\* II.2). Le registre PORTB, d'adresse 06h dans la banque 0, permet d'y accéder en lecture ou en écriture. Le registre TRISB, d'adresse 86h dans la banque 1, permet de choisir le sens de chaque patte (entrée ou sortie) : un bit à 1 positionne le port en entrée, un bit à 0 positionne le port en sortie.

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

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

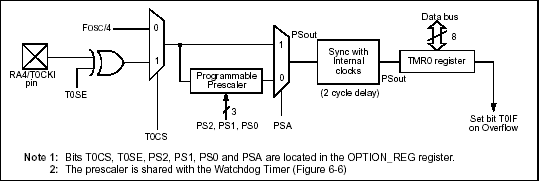
RB0 peut aussi servir d'entrée d'interruption externe.



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

# Compteur

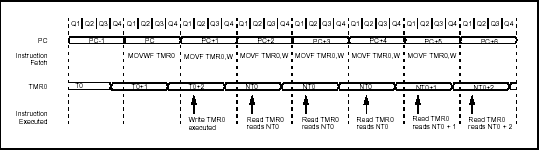
Le PIC 16F84 est doté d'un compteur 8 bits. La Figure\* VII.1 en donne l'organigramme.



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

# Registre TMR0

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



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

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

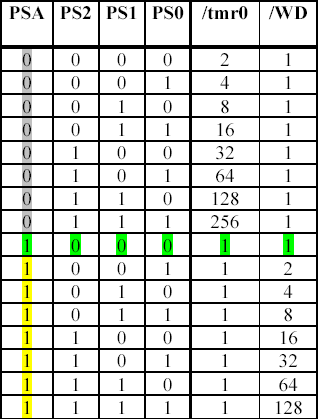
# Choix de l'horloge

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

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

# Pré-diviseur

En plus des deux horloges, un pré-diviseur, partagé avec le chien de garde, est disponible. La période de l'horloge d'entrée est divisée par une valeur comprise entre 2 et 256 suivant les bits PS2, PS1 et PS0 (respectivement OPTION\_REG.2, .1 et .0) (Figure\*\* VII.3). Le bit PSA (OPTION\_REG.3) permet de choisir entre la pré-division de timer0 (PSA=0) ou du chien de garde (PSA=1).



**Figure\*\* VII.3 :** *Valeurs du pré-diviseur en fonction de PSA, PS2, PS2 et PS0.*

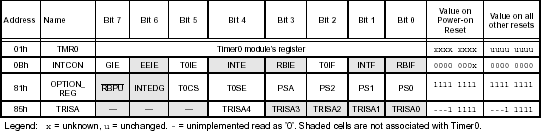
# Fin de comptage et interruption

Le bit T0IF (INTCON.2) est mis à 1 chaque fois que le registre TMR0 passe de FFh à 00h. On peut donc tester ce bit pour connaître la fin de comptage. Pour compter 50 événements, il faut donc charger TMR0 avec la valeur 256-50=206 et attendre le passage de T0IF à 1. Cette méthode est simple mais bloque le processeur dans une boucle d'attente.

On peut aussi repérer la fin du comptage grâce à l'interruption que peut générer T0IF en passant à 1 (§ VII). Le processeur est ainsi libre de travailler en attendant cet événement.

# Registres utiles à la gestion de timer0

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



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

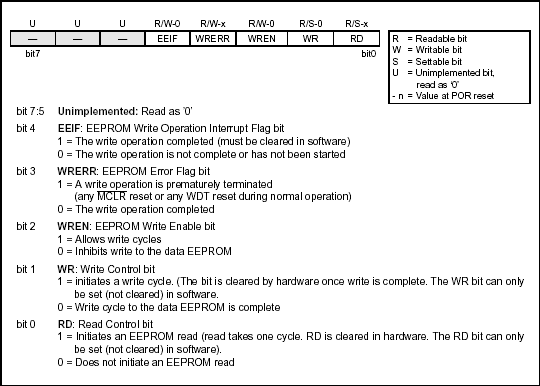
# Accès à la mémoire EEPORM

Le PIC possède une zone EEPROM de 64 octets accessibles en lecture et en écriture par le programme. On peut y sauvegarder des valeurs, qui seront conservées même si l'alimentation est éteinte, et les récupérer lors de la mise sous tension. Leur accès est spécifique et requiert l'utilisation de registres dédiés. La lecture et l'écriture ne peut s'exécuter que selon des séquences particulières.

# Registres utilisés

Quatre registres sont utilisés pour l'accès à la mémoire eeprom du PIC :

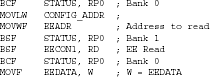
* EEDATA contient la donnée.
* EEADR contient l'adresse.
* EECON1 (Figure\* VIII.1) est le registre de contrôle de l'accès à l'eeprom. Cinq bits permettent un cet accès :
  + RD et WR initient la lecture ou l'écriture. Ils sont mis à 1 par le programme pour initier l'accès et mis à zéro par le système à la fin de l'accès.
  + WREN autorise (1) ou non (0) l'accès en écriture.
  + WRERR est mis à 1 par le système quand une opération d'écriture est interrompu par MCLR, reset ou le chien de garde.
  + EEIF est un drapeau d'interruption signalant la fin de l'écriture physique dans la mémoire eeprom. Il doit être mis à 0 par programme.



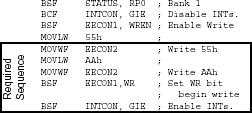
**Figure\* VIII.1 :** *Registre EECON1.*

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

# Lecture

Pour lire une donnée dans la mémoire eeprom, il faut mettre l'adresse dans EEADR et positionner RD à 1. La valeur lue est alors disponible dans EEDATA au cycle machine suivant. Le programme ci-dessous donne un exemple de lecture dans la mémoire eeprom.

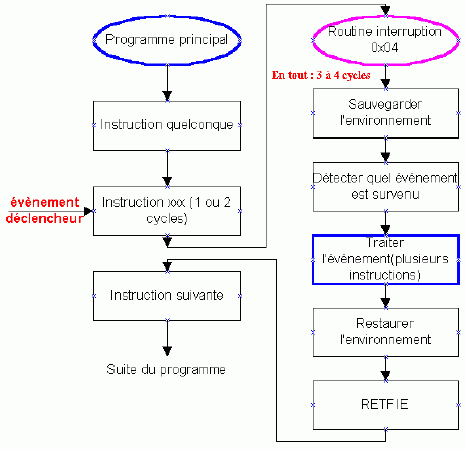
# Ecriture

Pour écrire une donnée dans la mémoire eeprom, il faut d'abord mettre l'adresse dans EEADR et la donnée dans EEDATA. Un cycle bien spécifique doit ensuite être respecter pour que l'écriture ait lieu. L'exemple suivant donne le cycle :

# Interruptions

* 1. **Rappel - Notion de sous-programme A développer**

# Mécanisme



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

L'interruption est un mécanisme fondamental de tout processeur. Il permet de prendre en compte des événements extérieurs au processeur et de leur associer un traitement spécifique. La Figure IX.1 donne le déroulement du programme lors d'une interruption. Il faut noter que l'exécution d'une instruction n'est jamais interrompue ; c'est à la fin de l'instruction en cours lors de l'arrivée de l'événement que le sous-programme d'interruption est exécuté.

La séquence classique de fonctionnement d'une interruption est la suivante : 1- Détection de l'événement déclencheur - IX.3, IX.4

1. Fin de l'instruction en cours
2. Sauvegarde de l'adresse de retour - IX.5
3. Déroutement vers la routine d'interruption - IX.5 5- Sauvegarde du contexte - IX.6

6- Identification de l'événement survenu - IX.6 7- Traitement de l'interruption correspondante 8- Restauration du contexte - IX.6

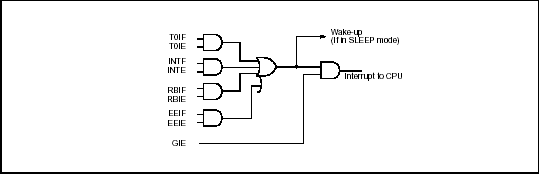
9- Retour au programme initial - IX.8

 Figure prise dans le cours de Bigonoff

# Différentes sources d'interruption

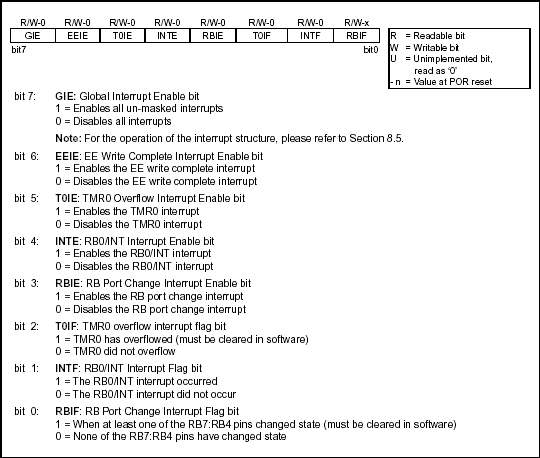
Dans le cas du PIC 16F84, il existe 4 sources d'interruption (Figure\* IX.2) :

* + - INT : Interruption externe, broche RB0/INT
    - TMR0 : Fin de comptage
    - PORTB : Changement d'état du port B (RB7-RB4)
    - EEPROM : Fin d'écriture en EEPROM



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

# Validation des interruptions



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

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

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

# Séquence de détournement vers le sous-programme d'interruption

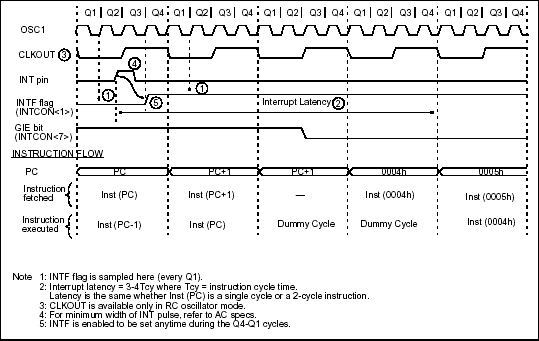
Par construction, l'interruption survient n'importe quand pendant l'exécution du programme. Avant l'exécution du sous-programme d'interruption, il faut donc sauvegarder l'adresse de l'instruction suivant celle en cours pour l'exécuter après le sous-programme d'interruption. L'adresse de retour est stockée dans la pile (Figure\* III.1). Cette opération est gérée automatiquement par le processeur.

Une fois l'adresse de retour sauvegardée, le compteur de programme peut être chargé avec l'adresse du sous-programme à exécuter, ici 0004h.

Dans le cas du PIC, à cause de la faible taille de la pile, une interruption n'est pas interruptible. Le bit GIE de validation générale est donc mis à 0 au début du sous-programme d'interruption. Cette opération est gérée automatiquement par le processeur.

La Figure\* IX.4 montre l'enchaînement des ces opérations. Cinq étapes sont alors utiles pour commencer l'interruption :

* Apparition d'un événement, sans perturber le déroulement normal des instructions
* Prise en compte de l'événement, exécution de l'instruction en cours (PC)
* Cycle d'attente, sauvegarde de l'adresse PC+1 dans la pile
* Chargement de l'adresse 0004h dans le PC
* Exécution de l'instruction d'adresse 0004h et chargement de l'instruction suivante



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

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

# Sauvegarde et restitution du contexte

C'est un point important pour tous les sous-programme qui devient capital pour les sous- programmes d'interruption. En effet, beaucoup d'instructions modifient le registre STATUS et/ou utilisent le registre W. Afin de les rendre dans le même état à la fin du sous-programme d'interruption qu'au début, il faut les sauvegarder au début et les recopier à la fin. Si d'autres

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

## Où sauvegarder ces registres ?

Classiquement dans la pile. Dans le cas des PICs, elle est très petite et non accessible pour l'utilisateur. Il faut donc définir une zone de sauvegarde dans la RAM. Pour définir une variable, on peut utiliser les directives CBLOCK et ENDC :

CBLOCK 0x0C ; début de la zone de stockage

Sauve\_W : 1 ; 1 octet réservé pour la sauvegarde de W Sauve\_Status :1 ; 1 octet réservé pour la sauvegarde de STAUTS

ENDC

## Comment sauvegarder ces registres ?

Pour W, c'est simple : il suffit d'utiliser l'instruction :

movwf Sauve\_W ; Sauvegarde de W

Pour STATUS, c'est plus compliqué. En effet, il n'existe pas d'instruction de transfert d'un registre vers un autre. Il faut donc passer par W. Première difficulté : il faut penser à sauvegarder W avant de l'utiliser pour sauvegarder STATUS. On pourrait alors utiliser la séquence suivante :

movf STATUS, 0 ; Ecrit la valeur de STATUS dans W movwf Sauve\_STATUS; ; Sauvegarde de STATUS

Malheureusement, cette séquence ne fonctionne pas. En effet, l'instruction movf modifie le bit Z du registre STATUS. L'astuce consiste à utiliser l'instruction swapf qui intervertit les digits de poids fort et de poids faible d'un registre, sans modifier le registre STATUS. La séquence suivante permet de sauvegarder STATUS "swapé" :

swapf STATUS, 0 ; Ecrit STATUS "swapé" dans W movwf Sauve\_STATUS ; Sauvegarde de STATUS "swapé"

## Comment restituer ces registres ?

Il faut commencer par restituer STATUS sans le modifier. En effet, on doit pour cela utiliser W qu'il est donc inutile de restituer avant. Comme STATUS a été sauvegardé "swapé", la séquence suivante convient :

swapf Sauve\_STATUS, 0 ; Ecrit Sauve\_Status "sawpé" dans W movwf STATUS ; Restitue STATUS, "swapé" deux fois

Pour restituer W, on pourrait tout simplement utiliser la séquence suivante : movf Sauve\_W, 0 ; Ecrit Sauve\_W dans W

Malheureusement, l'instrction movf modifie le bit Z du registre STATUS, déjà restitué. Il faut donc encore une fois passer par l'instruction swap, à exécuter deux fois :

swapf Sauve\_W, 1 ; Ecrit Sauve\_W "swapé" dans lui-même swapf Sauve\_W, 0 ; Restitue W, "swapé" deux fois.

## Résumé

; Création des variables de sauvegarde

CBLOCK 0x0C ; début de la zone de stockage

Sauve\_W : 1 ; 1 octet réservé pour la sauvegarde de W Sauve\_Status :1 ; 1 octet réservé pour la sauvegarde de STAUTS

ENDC

;----------------------------

; Routine d'interruption

;----------------------------

; Sauvegarde du contexte

movwf Sauve\_W ; Sauvegarde de W

swapf STATUS, 0 ; Ecrit STATUS "swapé" dans W movwf Sauve\_STATUS ; Sauvegarde de STATUS "swapé"

; Traitement de l'interruption

; …

; Restitution du contexte

|  |  |  |
| --- | --- | --- |
| swapf | Sauve\_STATUS, 0 | ; Ecrit Sauve\_Status "sawpé" dans W |
| movwf | STATUS | ; Restitue STATUS, "swapé" deux fois |
| swapf | Sauve\_W, 1 | ; Ecrit Sauve\_W "swapé" dans lui-même |
| swapf | Sauve\_W, 0 | ; Restitue W, "swapé" deux fois. |

# Reconnaissance de l'interruption active

En revanche, il n'existe qu'une adresse d'interruption, 0004h, pour les différentes sources. Les bits 0 à 2 du registre INTCON (Figure\* IX.3) et le bit 4 du registre EECON1 (Figure\* VIII.1) permettent de savoir quel événement extérieur a déclenché une interruption. Ainsi, au début du programme d'interruption, si plusieurs sources ont été validées, il faut impérativement aller tester ces différents bits pour connaître la source active et dérouler le programme correspondant. On utilise pour cela l'instruction btfsc qui exécute l'instruction suivante si le bit testé vaut 1, la saute sinon. On peut donc écrire la séquence suivante après la sauvegarde du contexte où Int\_xxx correspond aux différents sous-programmes de gestion des divers événements :

|  |  |  |
| --- | --- | --- |
| btfsc | INTCON, 0 | ; Test du bit RBIF |
| call | Int\_PB | ; Appel sous-programme si RBIF=1 |
| btfsc | INTCON, 1 | ; Test du bit INTF |
| call | Int\_Ext | ; Appel sous-programme si INTF=1 |
| btfsc | INTCON, 2 | ; Test de bit T0IF |
| call | Int\_Timer | ; Appel sous-programme si T0IF=1 |
| btfsc | EECON1, 4 | ; Test de bit EEIF |
| call | Int\_Timer | ; Appel sous-programme si EEIF=1 |

# Retour au programme initial

Une fois le sous-programme d'interruption terminé, après la restitution du contexte, il faut revenir au programme initial. C'est l'instruction retfie qui le permet. Elle commence par revalider les interruptions (GIE=1) puis elle revient au programme initial grâce à la valeur du compteur de programme empilée.

# Chien de garde

# Principe

C'est un système de protection contre un blocage du programme. Par exemple, si le programme attend le résultat d'un système extérieur (conversion analogique numérique par exemple) et qu'il n'y a pas de réponse, il peut rester bloquer. Pour en sortir on utilise un chien de garde. Il s'agit d'un compteur qui, lorsqu'il arrive en fin de comptage, permet de redémarrer le programme. Il est lancé au début du programme. En fonctionnement normal, il est remis à zéro régulièrement dans une branche du programme qui s'exécute régulièrement. Si le programme est bloqué, il ne passe plus

dans la branche de remise à zéro et le comptage va jusqu'au bout, déclenche le chien de garde qui relance le programme.

# Mise en service

Elle se décide lors de la programmation physique du PIC. Elle ne peut pas être suspendue pendant l'exécution d'un programme. Elle est définitive jusqu'à une nouvelle programmation de la puce.

La directive de programmation \_CONFIG permet de valider (option \_WDT\_ON) ou non (option

\_WDT\_OFF). La mise en service peut aussi être réalisée directement par le programmateur. L'inconvénient de cette seconde solution est que le code du programme ne contient pas l'information ; la mise en service du chien de grade peut être oubliée lors du téléchargement et générer un fonctionnement incorrect du programme en cas de blocage.

# Gestion

Une fois le chien de garde mis en service, il faut remettre le comptage à zéro régulièrement. Cette opération est réalisée par l'instruction *clrwdt*. Tant que le programme se déroule normalement, cette instruction est exécutée régulièrement et le chien de garde ne s'active pas. Si un blocage apparaît, la remise à zéro n'a pas lieu et le chien de garde est activé. Le PIC redémarre alors à l'adresse 0000h et le bit TO (STATUS.4) est mis 0. Le test de ce bit au début du programme permet de savoir si le système vient d'être mis sous tension (TO=1) ou si le chien de garde vient de s'activer (TO=0).

# Choix de la durée

Le chien de garde possède sa propre horloge. Sa période de base est de 18ms. Le pré-diviseur de fréquence utilisé par le compteur est partagé avec le chien de garde (Figure\*\* VII.3). Si le bit PSA (OPTION\_REG.3) est à 1, le pré-diviseur est assigné au chien de garde. 8 valeurs de 1 à 128 sont disponibles, ce qui permet d'aller jusqu'à 128\*18ms=2.3s avant le déclenchement du chien de garde.

# Mode sommeil

# Principe

Lorsque le PIC n'a rien à faire (par exemple lors de l'attente d'une mesure extérieure), ce mode est utilisé pour limiter sa consommation : le PIC est mis en sommeil (le programme s'arrête) jusqu'à son réveil (le programme repart). Ce mode est principalement utilisé pour les systèmes embarqués fonctionnant sur pile.

# Gestion

## Mise en sommeil

La mise en sommeil est réalisée grâce à l'instruction *sleep*. La séquence suivante est exécutée :

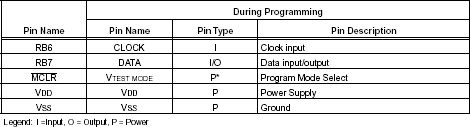
* + - * Le chien de garde est remis à 0 (équivalent à *clrwdt*)
      * Le bit TO (STATUS.4) est mis à 1
      * Le bit PD (STATUS.3) est mis à 0
      * L'oscillateur est arrêté ; le PIC n'exécute plus d'instruction

## Réveil

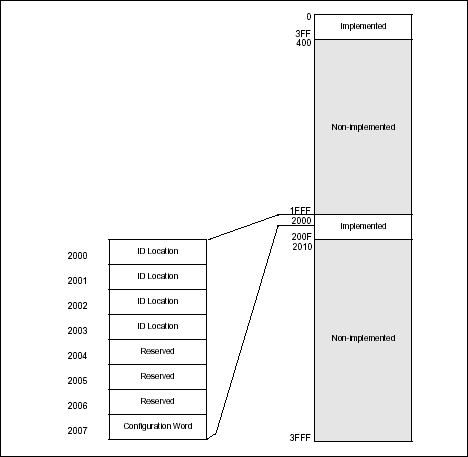
Ce mode n'est intéressant que si l'on peut en sortir pour relancer le programme. Trois événements permettent de sortir le PIC du sommeil :

* *Application d'un niveau 0 sur l'entrée MCLR (broche numéro 4)*. Le PIC effectue alors un reset et relance le programme à partir de l'adresse 0000h. Les bits TO (STATUS.4) et PD (STATUS.3) permettent à l'utilisateur de savoir quel événement à lancer le programme (mise sous tension, reset, chien de garde).
* *Activation du chien de garde*. Le programme reprend à l'instruction suivant le sleep.
* *Apparition d'une interruption (RB0/INT, RB ou EEPROM)*. Il faut pour cela que les bits de validation spécifique des interruption concernées soient positionnés. Si le bit de validation générale des interruptions (GIE) est à 0 (pas de validation des interruptions), le programme reprend après l'instruction sleep comme pour le chien de garde. Si le bit de validation générale des interruptions (GIE) est à 1 (validation des interruptions), l'instruction suivant le sleep est exécutée et la fonction d'interruption liée à l'événement qui a réveillé le PIC est exécutée.

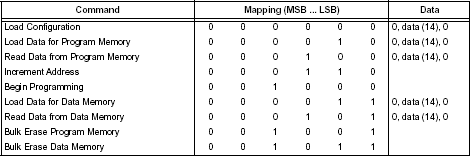
# Programmation sur site



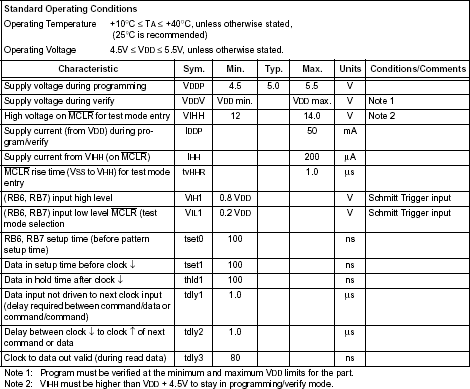
**Figure\* XII.1 :** *Pattes utilisées pour la programmation sur site.*



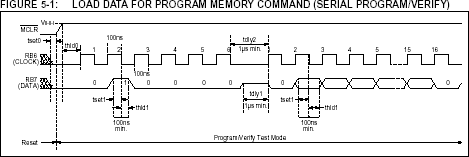
**Figure\* XII.2 :** *Adresses de la mémoire de programme.*



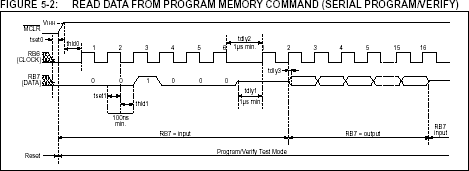
**Figure\* XII.3 :** *???.*



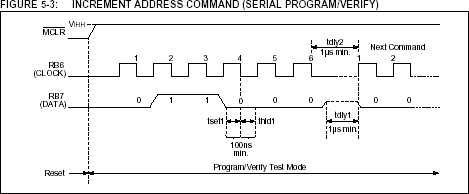
**Figure\* XII.4 :** *Caractéristiques électriques.*



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



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



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

# Bibliographie

1. Bigonoff (bigocours@hotmail.com) : "La programmation des PICs" - <http://fribotte.free.fr/bdtech/cours/pic16f84/>

\*\* : Figures prises dans ce document

1. Lycée Jacquard : "Le PIC 16FXX" - <http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm>
2. Lycée Jacquard : "MPLAB" - <http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm>
3. Microchip : "PIC16F8X – 18-pin Flash/EEPROM 8-bit micro-controllers" – DS30430C, <http://www.microchip.com/1010/suppdoc/>

\* : Figures prises dans ce document

1. Microchip : "PIC16F8X – EEPROM memory programming specification" – DS30262E, <http://www.microchip.com/1010/suppdoc/>