Pierre LOGLISCI

Le microcontrôleur PIC 16F84

Edition de l'Auteur

La ***bible pour désassembler à la main est une*** *nouveaté mondiale*, ***idée originale de l'Auteur***.

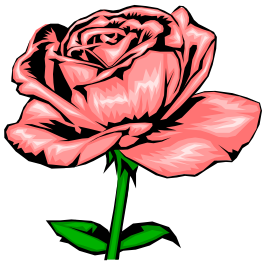
Dédicace

Lorsque ce travail n'était qu'un manuscrit, mon plus grand problème fut de savoir comment faire pour lui donner une forme dactylographique acceptable pour être lu par tous.

Je n'avais que très peu de connaissances en ce qui concerne l'utilisation du clavier, des logiciels de traitement de texte et d'images, et du scanner...
Je ne savais pas comment créer les indispensables tableaux, ni comment accéder aux caractères spéciaux....
Je n'avais aucune expérience de composition et de mise en page...
Bref : je ne savais pas comment m'y prendre pour faire en sorte qu'un tas de feuilles gribouillées à la main deviennent un livre...

Si j'y suis parvenu, c'est grâce à l'aide spontanée et constante que m'a apportée ma fille *Elodie* qui - comme un guide de haute montagne - m'a précédé dans la difficile ascension en m'ouvrant des chemins faciles et sûrs, toujours en se mettant à ma place, dosant les difficultés à la mesure de l'effort qu'elle savait que j'étais capable de fournir .
Sans jamais faire de concessions, elle s'est de nombreuses fois obligée à chercher - chez elle, sur son propre PC - les solutions pratiques aux problèmes que je lui soumettais, au fur et à mesure qu'ils se présentaient.

C’est à elle que je dédicace ce livre.



Pour son aide, bien sûr ; mais aussi pour les constants encouragements répétés qu’elle m’a donnés, l’enthousiasme qu’elle a partagé avec moi pendant tout le temps que j’ai consacré au projet, et son doux sourire, dont elle m’entoure encore.

Avant-propos

Bien qu'il existe une déjà abondante littérature sur les microcontrôleurs PIC *en général*, l'amateur qui veut s'attaquer à une réalisation personnelle utilisant le modèle 16F84 n'y arrive qu'au prix de gros efforts.
Ceci à cause du fait qu'il y a plusieurs modèles de PIC qui, tout en possédant la même philosophie, diffèrent sensiblement l'un de l'autre (par le nombre d'instructions, le nombre de pages mémoire, le nom des registres, la présence ou l'absence de ressources internes...) et déroutent celui qui en entreprend l'étude pour la première fois.

Or, si les ouvrages traitant les microcontrôleurs PIC *en général* sont nombreux, aucun n'aborde *le 16F84 en particulier*.

Dans ces conditions, le lecteur qui ne s'intéresse qu'à ce modèle exclusivement, doit se livrer à tout un travail pour séparer ce qui est important et nécessaire (parce que ça concerne le 16F84) de ce qui est superflu (parce que ça ne concerne pas le 16F84 mais se réfère à d'autres modèles).

Aussi j'ai condensé dans cet ouvrage les seules notions pratiques nécessaires pour aborder un montage personnel à base de 16F84.

Il y a plusieurs raisons à cela.

En premier lieu, ce modèle possédant une mémoire EEPROM effaçable électriquement, s'impose comme la solution idéale pour ceux qui veulent apprendre à utiliser un microcontrôleur PIC, du fait qu'il est reprogrammable jusqu'à plus de 1.000 fois (selon les spécifications du fabricant).

Associé à de simples organes périphériques, il représente l'outil d'apprentissage par excellence, car le lecteur peut tester tous les programmes avec le même microcontrôleur et revenir sur les erreurs, les corriger et rapidement re-tester l'application.

Ce microcontrôleur possède un fusible interne, accessible par programmation, qu'il faudra se garder de laisser intact. Car lorsque ce fusible a été brûlé, le microcontrôleur, s'il peut encore être effacé et reprogrammé, ne peut plus être lu correctement , car sa mémoire est restituée complètement désorganisée.
Bien utile pour ceux qui mettent au point une application commerciale qu'ils veulent protéger et mettre à l'abri de copies sauvages, ce fusible doit être ignoré pendant la durée de l'étude.

Cette accessibilité permanente de la mémoire représente l'aspect le plus original de tous les microcontrôleurs à mémoire flash, parmi lesquels trône le 16F84.

De plus, la capacité mémoire de ce modèle (ni trop petite ni trop grande) le prédestine comme le compromis idéal non seulement pour l'auto-apprentissage, mais aussi pour les premières applications personnelles que chacun aura envie d'inventer.

Car, s'il est incontestable qu'on peut parvenir à la réalisation d'un grand nombre de dispositifs en téléchargeant programmes et circuits imprimés à partir des nombreux sites consacrés aux microcontrôleurs, dans ce domaine particulier de la microélectronique seules les capacités personnelles comptent.

C'est pourquoi ce livre s'adresse tout particulièrement à qui veut vraiment prendre ... dans une main le PIC 16F84 ... et dans l'autre les indispensables outils de développement et... le fer à souder !

Les seules connaissances exigées pour en aborder la lecture sont les bases fondamentales de l'électronique générale et de l'électronique logique.

Un avertissement tout de même - s'il était nécessaire - consiste à rappeler que pour maîtriser la réalisation d'un montage incorporant un microcontrôleur 16F84 il faut disposer d'un ordinateur et d'un outil de développement (pouvant être soit une copie de l'assembleur MPLAB que Microchip distribue gratuitement sur son site Internet, soit un compilateur BASIC). Nous verrons ceci plus loin, dans la section traitant de la programmation.

Du *contrôleur* au *microcontrôleur*

Pour le dire avec des mots simples, un *contrôleur* est un dispositif qui - placé au cœur d'un processus - surveille l'évolution d'un événement et compare son état (ou sa valeur) à une donnée prédéterminée, pour intervenir dès que les limites préfixées sont atteintes.

De ce point de vue, un contrôleur n'est pas forcément électronique. Il peut être mécanique, pneumatique, thermique, etc..

Son travail consiste à surveiller (*lire*) la valeur d'une situation, et à la comparer en permanence à une valeur fixée d'avance.
Lorsqu'il y a une différence entre la valeur lue et celle fixée, le contrôleur génère une commande qui - envoyée à un endroit approprié du processus - réduit cette différence ou ramène les choses à la normale.

Aussi, un thermostat d'ambiance ou la valve de sécurité installée sur le couvercle d'une cocotte-minute, sont des exemples de contrôleurs simples.

Un contrôleur peut accomplir une ou plusieurs tâches à la suite.

Les plus souples de tous les contrôleurs sont évidemment les contrôleurs faisant appel à l'électronique, et plus particulièrement les *microcontrôleurs*.

La surveillance de la valeur d'une situation se fait alors au moyen d'une ou plusieurs lignes d'acquisition de données configurées en *entrées*, tandis que l'envoi de commandes se fait au moyen d'une ou plusieurs lignes configurées en *sorties*.

L'ensemble des tâches confiées à un microcontrôleur s'appelle *programme*.

Le microcontrôleur 16F84

Présentation générale

Ce modèle de PIC (*Programmable Interface Controler*) est un circuit de petite taille, fabriqué par la Société américaine Arizona MICROCHIP Technology.

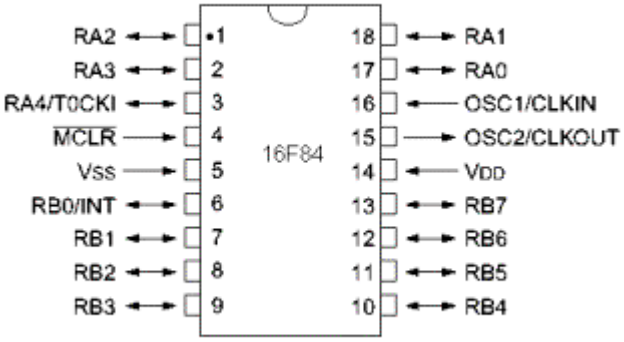
En le regardant pour la première fois, il fait davantage penser à un banal circuit intégré logique TTL ou MOS, plutôt qu'à un microcontrôleur. Son boîtier est un DIL (*Dual In Line*) de 2x9 pattes.

En dépit de sa petite taille, il est caractérisé par une architecture interne qui lui confère souplesse et vitesse incomparables.

Ses principales caractéristiques sont :

- 13 lignes d'entrées/sorties, réparties en un port de 5 lignes (Port A) et un port de 8 lignes (Port B)
- alimentation sous 5 Volts
- architecture interne révolutionnaire lui conférant une extraordinaire rapidité
- une mémoire de programme pouvant contenir 1.019 instructions de 14 bits chacune (allant de l'adresse 005 à l'adresse 3FF)
- une mémoire RAM utilisateur de 68 emplacements à 8 bits (de l'adresse 0C à l'adresse 4F)
- une mémoire RAM de 2x12 emplacements réservée aux registres spéciaux
- une mémoire EEPROM de 64 emplacements
- une horloge interne, avec pré diviseur et chien de garde
- possibilité d'être programmé *in-circuit*, c'est à dire sans qu'il soit nécessaire de le retirer du support de l'application
- vecteur de Reset situé à l'adresse 000
- un vecteur d'interruption, situé à l'adresse 004
- bus d'adresses de 13 lignes
- présence d'un code de protection permettant d'en empêcher la duplication
- facilité de programmation
- simplicité
- faible prix .

Brochage du PIC 16F84
(µC vu de dessus)



Le cortège des invariants

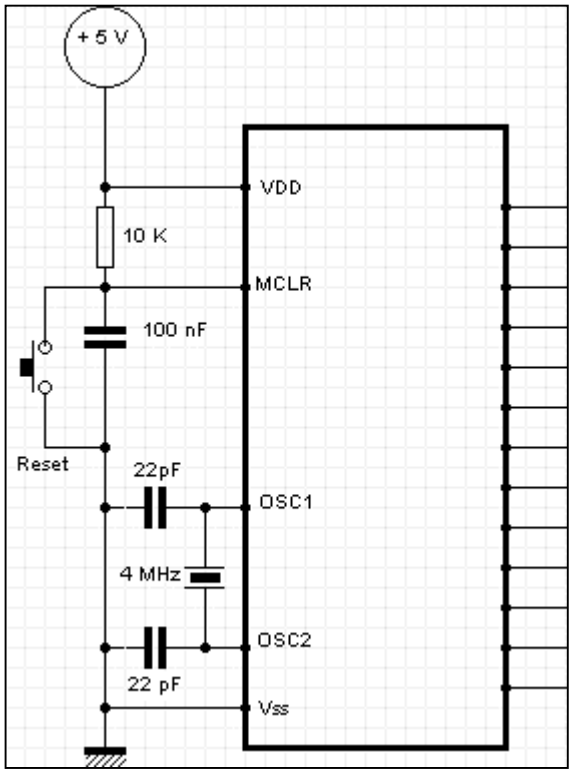
Indépendamment de ce qu'on veut faire de ses 13 lignes (que l'on définit par lignes d'entrée/sortie) et quelle que soit l'application à laquelle on le destine, un microcontrôleur PIC 16F84, pour pouvoir fonctionner, a nécessairement besoin de :

- une alimentation de 5 Volts ;
- un quartz et deux condensateurs (si un pilotage précis par base de temps à quartz est nécessaire), ou une résistance et un condensateur (pour une base de temps de type RC, économique, utilisable dans les cas ne demandant pas une extrême précision de cadencement) ;
- un condensateur de découplage (pour réduire les transitoires se formant inévitablement dans tout système impulsionnel) ;
- un bouton poussoir et une résistance, pour la mise en place d'une commande de Reset.

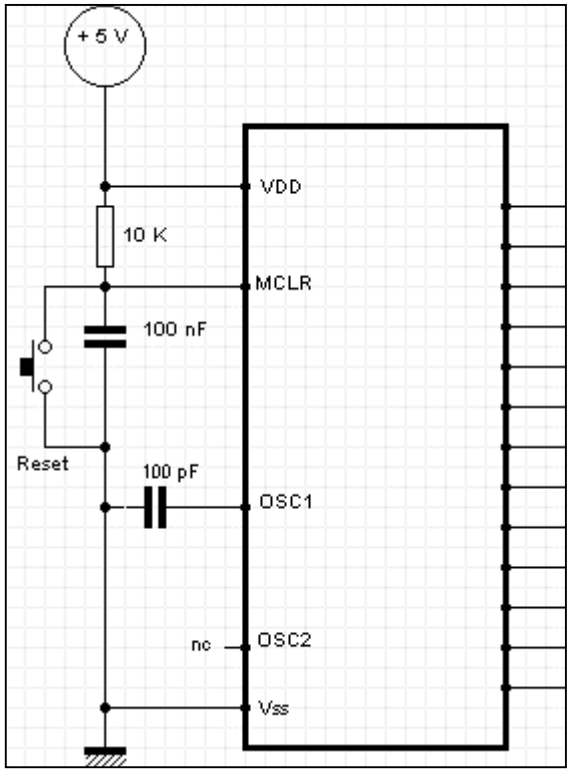
Ces éléments - qu'il convient de considérer comme des invariants devant nécessairement figurer dans tout montage - représentent le cortège obligatoire de tout microcontrôleur PIC 16F84, de la même façon - pourrais-je dire - qu'un transistor demande, pour fonctionner, une résistance de Base et une résistance de Collecteur.

Les applications type sont celles des deux pages suivantes :

1) Pilotage par quartz



2) Pilotage par oscillateur RC



Les Entrées/Sorties

A part les cinq pins réservées au cortège des invariants devant nécessairement figurer dans tout montage, les treize autres pins du 16F84 servent d'*entrées/sorties*.

Elles sont regroupées en deux ports : *Port A* et *Port B*.

Le *Port A* possède 5 lignes, nommées:
RA0.....pin 17
RA1.....pin 18
RA2.....pin 1
RA3.....pin 2
RA4.....pin 3 (RA4/T0CKI)
(NB : RA = Register A)

Le *Port B* possède 8 lignes, nommées:
RB0.....pin 6 (RB0/INT)
RB1.....pin 7
RB2.....pin 8
RB3.....pin 9
RB4.....pin 10
RB5.....pin 11
RB6.....pin 12
RB7.....pin 13
(NB : RB = Register B)

A remarquer que RB0 (pin 6) et RA4 (pin 3), outre qu'à pouvoir servir d'entrées/sorties, selon la façon dont on les programme peuvent respectivement servir l'une comme entrée d'interruption et l'autre comme entrée d'horloge externe pour le pilotage du timer (TMR0).

Organisation de la mémoire du PIC 16F84

La mémoire du PIC 16F84 est répartie en trois espaces, logés sur la même pastille de silicium :

1) Une mémoire EEPROM de type flash, de 1 K mots de 14 bits, allant de l'adresse 000 à l'adresse 3FF.
Cet espace est dénommé ***mémoire de programme***, dont le plan est le suivant :

5 adresses réservées au μC (adresses que je conseille de sauter)

000 Vecteur de Reset
001
002
003
004 Vecteur d'Interruption

005 Début du programme utilisateur

.
.

1019 adresses restantes, disponibles pour y loger les instructions de votre programme

.
.
.
.

3FF Fin de l'espace mémoire disponible

Cette mémoire est celle dans laquelle le programmateur écrit les instructions du programme.

Dans cet espace mémoire, les cinq premières adresses (000, 001, 002, 003, et 004) sont réservées au microcontrôleur.

Certaines d'entre elles sont particulièrement remarquables :

a) l'adresse 000 correspond au *vecteur de Reset*.
A la mise sous tension, ou à chaque fois que des instructions spécifiques l'obligent, le Program Counter (PC) se rend à cette adresse et c'est là que le système trouve la première instruction à exécuter.
C'est une case devant obligatoirement être remplie et contenir l'origine du programme (ORG).
Si cette adresse était vide, le microcontrôleur ne ferait rien, car aucun programme ne serait exécuté.

b) l'adresse 004 correspond au *vecteur d'interruption*.
C'est l'adresse « point de rencontre » définie par le fabricant, à laquelle système et utilisateur se rendent lorsqu'un problème surgit, pour se dire ce qu'il se passe et quel sont les remèdes d'urgence à apporter.

2) une *mémoire de données* (Data Memory) *EEPROM* flash, de 64 emplacements à 8 bits, allant de l'adresse 00 à l'adresse 3F, auxquels on accède uniquement par l'intermédiaire de quatre registres spéciaux:

- -EEADR (EEprom ADRess) pour ce qui concerne les adresses
- EEDATA (EEprom DATA) pour ce qui concerne les données
- EECON1 (EEprom CONtrol) permettant de définir le
- EECON2 mode de fonctionnement de cette mémoire.

<u>Pour lire</u> dans cette mémoire, les étapes à suivre sont les suivantes :
1) on écrit l'adresse dans le registre EEADR ;
2) on met à 1 le bit 0 (RD : Read Data) du registre EECON1 (ce qui provoque le transfert de la donnée dans le registre EEDATA) ;
3) on lit la donnée dans le registre EEDATA où elle est devenue disponible.

Exemple : on veut ***lire*** le contenu de l'emplacement mémoire 03 :

```
BCF     STATUS,RP0
MOVLW   03
MOVWF   EEADR
BSF     STATUS,RP0
BSF     EECON1,0
BCF     STATUS,RP0
```

A partir de ce moment, ayant autorisé le mode « *lecture* », la donnée contenue à l'adresse 03 est disponible dans le registre EEDATA, et on peut l'utiliser comme on veut.

Ainsi, par exemple, on veut lire une donnée en EEPROM et la porter dans le registre W :

```
BCF     STATUS,RP0
MOVLW   adresse dont on veut lire le contenu
MOVWF   EEADR
BSF     STATUS,RP0
BSF     EECON1,0
BCF     STATUS,RP0
MOVF    EEDATA,W
```

Voyons maintenant *comment* ***écrire*** *une donnée en EEPROM* :

```
BCF     STATUS,RP0
BCF     INTCON,7
MOVLW   donnée que l'on veut écrire
MOVWF   EEDATA
MOVLW   adresse
MOVWF   EEADR
BSF     STATUS,RP0
BCF     EECON1,4
BSF     EECON1,2
MOVLW   55
MOVWF   EECON2
MOVLW   AA
MOVWF   EECON2
```

```
BSF     EECON1,1
BCF     EECON1,2
BSF     INTCON,7
BCF     STATUS,RP0
```

Cette séquence montre que pour **écrire** dans cette mémoire, les étapes à suivre sont un peu plus complexes, car on est obligé de passer d'abord par EECON2 avant de confirmer la donnée par EECON1 :
1) on interdit les interruptions ;
2) on écrit la donnée dans le registre EEDATA ;
3) on écrit l'adresse dans le registre EEADR ;
4) on configure le registre EECON1
5) on envoie la séquence définie par Microchip (55 et AA)
6) on reconfigure les registres EECON1, INTCON et STATUS.

Voyons les choses plus en détail au moyen d'un autre exemple montrant l'écriture d'une donnée (par exemple : 13) à une certaine adresse (par exemple : 1F) :

```
EEDATA  EQU     08
EEADR   EQU     09
EECON1  EQU     88
EECON2  EQU     89
INTCON  EQU     0B
        BCF     INTCON,7
        MOVLW   13
        MOVWF   EEDATA
        MOVLW   1F
        MOVWF   EEADR
        BSF     STATUS,RP0
        BCF     EECON1,4
        BSF     EECON1,2
        MOVLW   55
        MOVWF   EECON2
        MOVLW   AA
        MOVWF   EECON2
        BSF     EECON1,1
        BCF     EECON1,2
        BSF     INTCON,7
        BCF     STATUS,RP0
```

3) Une mémoire RAM à 8 bits, que Microchip appelle *Register File*, réservée aux données.
A plus proprement parler, il s'agit d'une RAM statique (SRAM).

Cet espace est à son tour réparti en deux zones :

a) une zone RAM de 24 emplacements à 8 bits réservée aux *registres spéciaux*, dont 12 situés en *Page 0* (adresses 00 à 0B) et 12 situés en *Page 1* (adresses 80 à 8B) selon la mappe suivante :

Page 0		**Page 1**	
00	Adressage indirect	80	Adressage indirect
01	TMR0	81	OPTION
02	PCL	82	PCL
03	STATUS	83	STATUS
04	FSR	84	FSR
05	PORT A	85	TRIS A
06	PORT B	86	TRIS B
07		87	
08	EEDATA	88	EECON1
09	EEADR	89	EECON2
0A	PCLATH	8A	PCLATH
0B	INTCON	8B	INTCON

Ces registres - auxquels on accède en programmant le bit 5 (RP0) du registre STATUS - servent à contrôler le fonctionnement de nombreux organes internes au PIC. Nous y reviendrons plus en détail;

b) une zone RAM de données, constituée de 68 emplacements à 8 bits (adresses de 0C à 4F) situés juste au dessous des registres spéciaux, formant la *RAM utilisateur* proprement dite, selon la mappe détaillée ci-après :

0C
0D
0E
0F
10

11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
20
21
22
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
30
32
33
34
35
36
37
38
39
3A

3B
3C
3D
3E
3F
40
41
42
43
44
45
46
47
48
49
4A
4B
4C
4D
4E
4F

Lors de la programmation il faut toujours indiquer l'adresse de la zone RAM à partir de laquelle le μC doit commencer à écrire, ainsi que le nombre d'emplacements à réserver pour chaque variable.

Comme ceci, par exemple :

```
            ORG     OC
Compteur    RES     3
```

Ce qui revient à dire : réserve trois emplacements à la variable Compteur, dans l'ordre suivant :

Compteur à l'adresse 0C
Compteur+1 à l'adresse 0D
Compteur+2 à l'adresse 0E .

Ainsi, par exemple :

pour effacer les données de l'adresse OE, on écrira :

```
CLRF Compteur+2.
```

4) et enfin une toute petite mémoire EEPROM, contenant seulement 8 cases, de l'adresse 2000 à l'adresse 2007, réservées au microcontrôleur.

Les adresses 2000, 2001, 2002 et 2003 correspondent aux emplacements dans lesquels l'utilisateur peut stocker un code d'identification (en n'utilisant que les quatre bits de poids faible de chacun de ces mots à 14 bits).

L'adresse 2007 correspond au *registre de configuration* du microcontrôleur.

Lui aussi mot de 14 bits, dont les cinq premiers seulement sont utilisables :

4	3	2	1	0
CP	PWRTE	WDTE	$FOSC_1$	$FOSC_0$

- Bit 0 - $FOSC_0$ (OSCillateur zéro) et
- Bit 1 - $FOSC_1$ (OSCillateur un)

sont à programmer en fonction du type d'oscillateur utilisé, conformément aux spécifications du tableau suivant:

$FOSC_1$	$FOSC_0$	Type d'oscillateur	Caractéristiques
0	0	LP	(Low Power) Quartz jusqu'à 200 KHz
0	1	XT	Quartz (XT ou 4) MHz
1	0	HS	(High Speed) jusqu'à 20 MHz
1	1	RC	RC jusqu'à 4 MHz

- Bit 2 - WDTE (Watch-Dog Timer Enable)
  1 = autorise le chien de garde
  0 = n'autorise pas le chien de garde
- Bit 3 - PWRTE (PoWeR Timer Enable)
  Le μC possède un timer permettant de retarder de 72 ms le lancement du programme après la mise sous tension du circuit.
  1 = le μC attend 72 ms
  0 = le μC démarre tout de suite

- Bit 4 - CP (Code Protection)
  1 = pas de protection (le μC pourra être lu correctement)
  0 = avec protection (le μC ne pourra plus être lu correctement.
  Le contenu de la mémoire sera désorganisé).

Les registres spéciaux

Nous avons dit que dans l'espace mémoire RAM que Microchip appelle Register File, une zone est réservée aux *registres spéciaux*.

Le mot *registre* est utilisé ici pour désigner un emplacement mémoire, tandis que le mot *file* signifie groupement.

Certains de ces registres sont situés en Page 0 entre les adresses 00 et 0B, et d'autres sont situés en Page 1 entre les adresses 80 et 8B. Quelques-uns d'entre eux figurent même dans les deux pages (Page 0 et Page 1) pour en faciliter l'accès.

Ils ont des noms et des usages spécifiques, et servent à commander le microcontrôleur.

Il y en a 16 en tout, et sont si importants qu'ils conditionnent véritablement la programmation.

Ils sont utilisés constamment, et constamment tenus présents dans la tête du programmeur.

Celui qui veut écrire ne fût-ce qu'un petit programme de quelques lignes, ne peut pas les ignorer.

C'est pourquoi ils doivent être étudiés et connus à fond.

Examinons-les en détail, un par un, par ordre alphabétique.

EEADR (EEprom ADRess)

Registre dans lequel on écrit l'adresse de la mémoire de données EEPROM (mémoire flash de 64 octets, allant de l'adresse 00 à l'adresse 3F) à laquelle on veut accéder pour y lire ou pour y écrire.

Contrairement à l'EEPROM de programme qui - en plus de la tension d'alimentation du microcontrôleur - nécessite une tension externe pour la programmation, cette EEPROM fonctionne avec la seule tension d'alimentation, dans toute sa plage.

EECON1 (EEprom CONtrol 1)

Registre de contrôle permettant de définir le mode de fonctionnement de la mémoire de données EEPROM (mémoire flash de 64 octets, allant de l'adresse 00 à l'adresse 3F).
Registre à 8 bits, mais dont 5 seulement sont utilisés:

7	6	5	4	3	2	1	0
			EEIF	WRERR	WREN	WR	RD

Bit 0 : RD (ReaD)
Normalement à 0. Il se met dans cet état de lui-même.
Le programmeur ne peut y écrire que un 1.
N'accepte pas d'être programmé à zéro.

Bit 1 : WR (WRite)
Normalement à 0. Il se met dans cet état de lui-même.
Le programmeur ne peut écrire que un 1.
N'accepte pas d'être programmé à zéro.

Bit 2 : WREN (WRite ENable)
Mis à zéro, interdit toute écriture en mémoire.
Mis à 1, autorise une écriture en mémoire.

Bit 3 : WRERR (WRite ERRor)
Flag d'erreur. Normalement à zéro.
Passe à 1 pour signaler qu'une erreur s'est produite juste au moment où une écriture était en cours (Celle-ci n'a pu aboutir parce qu'un événement inopiné s'est produit ; par exemple un Reset).

Bit 4 : EEIF (EEprom Interrupt Flag)
Flag d'interruption.
Il est automatiquement mis à 1 lorsque la programmation de l'EEPROM de données est terminée.
Doit être mis à zéro par programmation.

EECON2 (EEprom CONtrol 2)

Registre n'ayant aucune consistance physique, et dont le seul rôle consiste à obliger le programmeur à vérifier les données qu'il envoie dans l'EEPROM.

EEDATA (EEprom DATA)

- Pendant une opération de lecture : registre dans lequel est disponible la donnée qu'on est allé chercher à une certaine adresse de la mémoire EEPROM.

- Pendant une opération d'écriture : registre dans lequel on place la donnée qu'on veut y écrire.

FSR (File Select Register)

Sert à sélectionner la mémoire de données, pour pouvoir y accéder.

INTCON (INTerrupt CONtrol)

Est le registre qui préside au fonctionnement des interruptions.

Dans le 16F84 il y a quatre sources possibles d'interruptions. Chaque fois que l'une d'elles surgit, le microcontrôleur (après avoir noté dans la pile l'adresse de retour) abandonne momentanément (interrompt) le programme qu'il avait en cours d'exécution et saute à l'adresse 004 (adresse prédéfinie par le fabricant, de la même façon que l'adresse 000 a été prédéfinie pour la fonction Reset).

En lisant le contenu de ce registre, on peut déterminer la provenance de la demande d'interruption et aiguiller le programme de manière à y répondre de façon adéquate.

L'interruption peut être commandée soit par un flanc montant, soit par un flanc descendant : cela dépend de la façon dont on a préalablement programmé le bit 6 (INTEDG) du registre OPTION :

1 = l'interruption est générée à l'apparition d'un front montant ;
0 = l'interruption est générée à l'apparition d'un front descendant.

Les quatre sources d'interruption possibles sont :

1) la fin d'une programmation de l'EEPROM de données ;

2) le débordement du timer interne ;

3) une commande externe appliquée sur la pin 6 (RB0/INT) ;

4) un changement d'état sur l'une des pins 10, 11, 12 ou 13 (respectivement RB4, RB5, RB6, RB7).
Dans ce cas, seule une configuration des lignes en *entrée* peut donner lieu à une éventuelle demande d'interruption.

Examinons un par un chacun des bits de ce registre :

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

Bit 7 : GIE (Global Interrupt Enable)

Le chef suprême du registre INTCON.
Le roi de toutes les interruptions !
Mis à 1, il autorise la prise en compte de tous les autres bits de l'octet ; tandis que mis à zéro, il les *masque* (même s'ils sont positionnés).
Sous l'action d'un Reset il est automatiquement mis à zéro.
Pour interdire la prise en compte de toute éventuelle demande d'interruption pouvant surgir pendant que le microcontrôleur est déjà occupé à en traiter une, ce bit (GIE) passe automatiquement à zéro jusqu'à ce que dans le programme apparaît l'instruction RETFIE, qui le repositionne à 1.
Au cas où l'utilisation de ce bit soit nécessaire, il ne faut pas oublier que:
après l'avoir activé (pour ouvrir l'accès au(x) bit(s) concerné(s),
et après l'instruction RETFIE, c'est à dire à la fin d'un sous-programme d'interruption,
lorsque son utilisation n'est plus nécessaire, il faut le mettre à zéro, au risque de placer les demandes d'interruption dans un cycle qui les ferait revenir continuellement.

Bit 6 : EEIE (EEprom Interrupt Enable)

Mis à 1, autorise l'interruption que l'EEPROM génère à la fin de la programmation.

7	6	5	4	3	2	1	0
	EEIE						

Bit 5 : T0IE (Timer zero Interrupt Enable)

7	6	5	4	3	2	1	0
		TOIE					

Mis à 1, il autorise les interruptions provoquées par le
débordement du timer interne (passage de FF à 00).

Bit 4 : INTE (INTerrupt Enable)

7	6	5	4	3	2	1	0
			INTE				

Mis à 1, il autorise les demandes d'interruption
provenant de l'extérieur, appliquées sur la pin 6
(RB_0 /INT).
NB : ces demandes peuvent se déclencher soit à l'apparition d'un
Front montant, soit à l'apparition d'un front descendant, selon la façon
Dont on a programmé le bit 6 du registre OPTION (INTEDG)
1 = sur front montant
0 = sur front descendant

Bit 3 : RBIE (Register B Interrupt Enable)

7	6	5	4	3	2	1	0
				RBIE			

Mis à 1, il autorise les interruptions provoquées
par un changement d'état sur les lignes du port B
(RB_4, BR_5, RB_6, RB_7).

Bit 2 : T0IF (Timer zero Interrupt Flag)

7	6	5	4	3	2	1	0
					TOIF		

Le fonctionnement de ce flag est conditionné par l'état du bit 5. Il ne fonctionne que si le bit 5 a préalablement été mis à 1.
Dans la mesure où le bit 5 est à 1, ce flag passe à 1 chaque fois que le timer TMR0 déborde (passage de FF à 00).

Bit 1 : INTF (INTerrupt Flag)

7	6	5	4	3	2	1	0
						INTF	

Le fonctionnement de ce flag est conditionné par l'état du bit 4. Il ne fonctionne que si le bit 4 a préalablement été mis à 1.
Dans la mesure où le bit 4 est à 1, ce flag passe à 1 chaque fois qu'une demande d'interruption surgit, provenant de l'extérieur, appliquée sur la pin 6 du boîtier (RB0/INT).

Bit 0 : RBIF (Register B Interrupt Flag)

7	6	5	4	3	2	1	0
							RBIF

Le fonctionnement de ce flag est conditionné par l'état du bit 3. Il ne fonctionne que si le bit 3 a préalablement été mis à 1.
Dans la mesure où le bit 3 est à 1, ce bit passe à 1 chaque fois qu'il y a un changement d'état sur l'une des lignes du port B (RB4, RB5, RB6 ou RB7) par rapport à la dernière opération de lecture du port B (dans la mesure, évidemment, où les lignes de ce port sont configurées en *entrée*).

OPTION

Est le registre qui préside au fonctionnement de l'horloge interne du microcontrôleur (TMR0) :

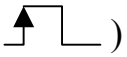
7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

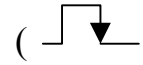
Bit 7 : RBPU (Register B Pull Up)

Mis à zéro (actif à l'état bas) valide les résistances de pull-up présentes, à l'intérieur du boîtier, sur les lignes du port B.

Bit 6 : INTEDG (INTerrupt EDGe)

Détermine le front du signal d'horloge sur lequel on veut que soit prise en compte une demande d'interruption provenant de l'extérieur (commande appliquée sur la pin 6 : RB0/INT). Car on peut faire agir une telle demande soit à l'apparition d'un front montant (passage de zéro à 1), soit à l'apparition d'un front descendant (passage de 1 à zéro).

1 = interruption programmée pour se déclencher sur un front montant ()

0 = interruption programmée pour se déclencher sur un front descendant ().

Bit 5 : T0CS (Timer zero Clock Source)

Sert à choisir la provenance du signal qu'on souhaite utiliser comme clock pour piloter l'horloge interne. Il existe deux choix possibles : soit utiliser l'horloge interne utilisant le quartz pilote du microcontrôleur et fournissant un signal dont la fréquence est celle du quartz divisée par 4, soit utiliser un signal externe prélevé sur la pin RA4 (bit 4 du port A).

0 = le timer est piloté par l'horloge interne

1 = le timer est piloté par un signal externe.

Bit 4 : T0SE (Timer zero Signal Edge)

Sert à déterminer si l'horloge doit avancer sur front montant ou sur front descendant.

0 = l'horloge avance sur front montant ()

1 = l'horloge avance sur front descendant () .

Bit 3 : PSA (Pre-Scaler Assignment)

Sert à affecter le prédiviseur soit au timer TMR0 soit au Watch-Dog.

0 = le pré diviseur est affecté au timer TMR0

1 = le pré diviseur est affecté au Watch-Dog.

Bits 2 – 1 – 0 : PS2 – PS1 – PS0 (Pre-Scaler rate)

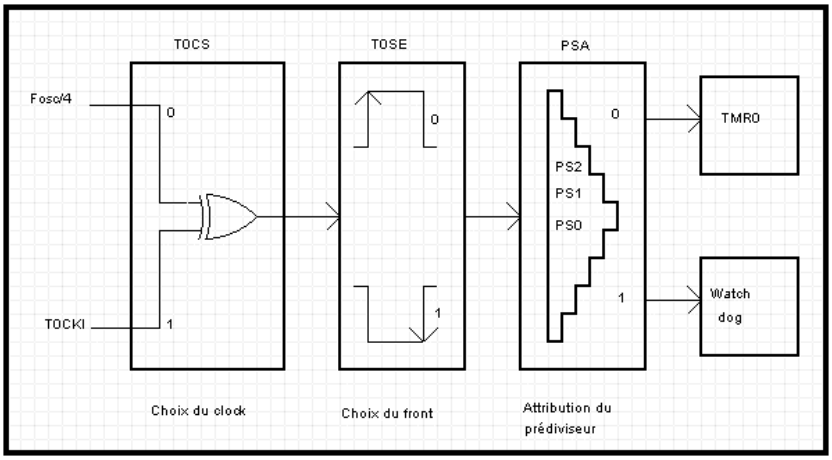
Ces trois bits servent à programmer le facteur de division qu'on veut assigner au prédiviseur dans le but d'avoir des signaux plus lents.

A remarquer que le facteur de division n'est pas le même selon que le pré diviseur soit affecté au TMR0 (timer) ou au Watch-Dog (chien de garde) :

PS2	PS1	PS0	Facteur de division	
			Pour le TMR0	Pour le Watch-Dog
0	0	0	2	1
0	0	1	4	2
0	1	0	8	4
0	1	1	16	8
1	0	0	32	16
1	0	1	64	32
1	1	0	128	64
1	1	1	256	128

Pour écrire dans ce registre on utilise soit l'instruction BSF,bit Soit l'instruction BCF,bit.

Représentation imagée du fonctionnement du timer TMR0 et du Watch-Dog (chien de garde) :



PCL (Program Counter Low)

Il s'agit du compteur qui fournit au programme la partie basse de l'adresse.

Dans les microcontrôleurs de Microchip les lignes d'adresses sont réparties en deux bytes : le PCL (fourni par ce registre Program Counter Low), et le PCH (fourni par le registre PCLATH Program Counter LATch High).

Il s'agit d'un compteur dont la tâche est d'adresser la mémoire dans laquelle sont logées les instructions du programme.

L'organisation de ce type de compteur, dans les microcontrôleurs PIC, veut que l'adresse soit composée de deux parties : la partie basse (fournie par PCL, sur dix lignes d'adresse) et la partie haute (fournie par PCLATH).

PCLATH (Program Counter LATch High)

L’adresse du compteur de programme est obtenue en mettant ensemble la partie basse fournie par PCL (Program Counter Low) et la partie haute fournie par PCLATH.

Contrairement à ce qu’on pourrait penser, ce registre ne fournit pas un nombre complémentaire fixe de bits, mais un nombre de bits variable, en fonction des instructions qui sont traitées.

PORT A - PORT B

Alors que TRIS A et TRIS B se limitent à définir le sens de chaque ligne des ports (*entrée* ou *sortie*), PORT A et PORT B permettent concrètement au microcontrôleur de communiquer avec l'extérieur.

Voici des exemples :

1) Comment écrire un 0 sur une ligne de port (par exemple, sur RA0) :

MOVLW	11111110	(en binaire, pour que ce soit plus parlant). Octet de configuration de port : 0 = sortie 1 = entrée Dans ce cas : on veut programmer le bit 0 en *sortie*, et tous les autres en *entrée*.
MOVWF	TRISA	Charge l'octet de configuration dans le registre TRIS A, mais toutes les lignes sont encore maintenues en haute impédance.
BCF	PORTA,0	Met à zéro (clear) le bit 0 du port A. Toutes les autres lignes du port restent à haute impédance.

2) Comment écrire un 1 sur une ligne de port (par exemple, sur RA2) :

MOVLW	11111110	(en binaire, pour que ce soit plus parlant). Octet de configuration de port : 0 = sortie 1 = entrée Dans ce cas : on veut programmer le bit 2 en *sortie* et tous les autres en *entrée*.
MOVWF	TRISA	Charge l'octet de configuration dans le registre TRIS A.

BSF	PORTA,2	Met à 1 (set) le bit 2 du port A. Toutes les autres lignes du port restent à haute impédance.

3) Comment lire l'état logique d'une ligne de port.
C'est à dire : comment savoir si une ligne est à 0 ou à 1 :

MOVLW	11111111	(en binaire). Octet de configution de port (on veut que toutes les lignes du port soient des *entrées*).
MOVWF	TRISA	Charge l'octet de configuration dans le registre TRIS A.
BTFSC	PORTA,3	Teste le bit 3 des lignes du port A. S'il est à 1, l'instruction suivante est exécutée. Si par contre il est à 0, l'instruction suivante est ignorée et le programme exécute l'instruction se trouvant encore après. NB : Au lieu de BTFSC, on aurait pu utiliser l'instruction BTFSS pour, dans ce cas, exéter l'instruction suivante si le bit testé est à 0.

4) Comment lire l'octet entier d'un port configuré en *entrée* :

MOVF	PORTA,W	Charge le contenu du port A dans le registre W.

STATUS (Registre d'état)

Les cinq premiers bits de ce registre (bits 0 à 4) correspondent à des flags que le programmeur peut interroger pour obtenir des informations lui permettant d'écrire correctement la suite des instructions de son programme ; tandis que les bits 5, 6 et 7 (RP_0, RP_1, RP_2), d'après la façon dont on les programme, pourraient sélectionner 8 pages de registres internes (chacune de 128 octets).
Comme dans le 16F84 il n'y a que deux pages de registres (Page 0 et Page 1), seul le bit 5 (RP_0) sert (les bits 6 et 7 sont à ignorer purement et simplement).

7	6	5	4	3	2	1	0
		RP_0	TO	PD	Z	DC	C

Bit 0 : C (Carry)
Flag indiquant si une retenue a eu lieu dans un octet lors d'une addition ou d'une soustraction.
Si une retenue a été générée, ce bit passe à 1.

Bit 1 : DC (Digit Carry)
Flag fonctionnant comme le bit de Carry, sauf qu'ici la surveillance de la retenue s'exerce non pas sur l'octet entier, mais sur le premier demi-octet.
Ce flag se positionne à 1 si une retenue est générée du bit 3 (bit de poids fort du quartet inférieur) vers le bit 0 du quartet supérieur.
Il est utile pour corriger le résultat d'opérations effectuées en code BCD.

Bit 2 : Z (Zero)
Ce flag passe à 1 si le résultat d'une opération (arithmétique ou logique) est 0.

Bit 3 : PD (Power Down)
Mise en veilleuse de l'alimentation, effectuée par l'instruction SLEEP.
Passe à 1 lorsqu'on utilise l'instruction CLWDT, ou à la mise sous tension.

Bit 4 : TO (Time Out)
Dépassement de délai.

Passe à 0 si le timer du Watch-Dog (chien de garde) déborde.
Est mis à 1 par les instructions CLWDT et SLEEP, ainsi qu'à la mise sous tension.

Bit 5 : RP0 (Register Page zero)
Sert à sélectionner l'une des deux pages de registres (Page 0 ou Page 1).

0 = sélectionne la page mémoire 0 (adresses de 00 à 7F)
1 = sélectionne la page mémoire 1 (adresses de 80 à FF).

Exemples de programmation :

1) BCF STATUS, RP0 (Bit Clear File STATUS Register Page 0).
Met à zéro le bit RP0 du registre d'état. Autrement dit : sélectionne le banc mémoire 0 (adresses 00 à 7F).

2) BSF STATUS,RP0 (Bit Set File STATUS Register Page 0).
Met à 1 le bit RP0 du registre d'état. Autrement dit : sélectionne le banc mémoire 1 (adresses de 80 à FF).

TMR0 (TiMeR zero)

Est le registre de contrôle de l'horloge interne (timer) du microcontrôleur.
Ce timer peut soit fonctionner seul, soit être précédé par un pré diviseur programmable à 8 bits dont la programmation se fait par l'intermédiaire du registre OPTION.

Ce registre peut être lu et modifié à tout moment, soit pour connaître sa position courante, soit pour le déclencher à partir d'une valeur déterminée.
Une quelconque opération d'écriture dans ce registre met automatiquement à zéro le comptage en cours dans le pré diviseur.
Se rappeler que le timer compte sur 8 bits, et qu'une fois que le comptage est arrivé à FF, celui-ci revient à 00 (ce qui provoque le passage à 1 du bit 2 du registre INTCON appelé T0IF).

En programmation on peut écrire :

1) pour le lire :
MOVF TMR0 ,W
(porte la valeur de TMR0 dans W)

2) pour lui donner une valeur de départ :
MOVLW valeur
MOVWF TMR0

3) pour le mettre à zéro :
CLRF TMR0

TRIS A - TRIS B

Ce sont les registres qui *définissent le sens* de chacune des lignes des ports A et B.

Toute ligne mise à 1 est programmée comme entrée, tandis que toute ligne mise à zéro est programmée comme sortie.

Il n'y a aucune instruction permettant d'écrire directement dans ces registres : on y accède en transitant par le registre de travail W. En programmation, on commence donc par charger l'octet de configuration dans le registre W, puis on copie celui-ci dans TRIS A ou TRIS B.

Exemple :

```
MOVLW     00000001   (en binaire, sinon 01 en hexa)
MOVWF     TRISA
```

Le bit 0 du port A est défini comme *entrée*, tandis que les sept autres lignes sont définies comme *sorties*.

La PROGRAMMATION

Différentes façons de programmer

Il existe plus d'un chemin possible pour programmer les PIC.

Nous en examinerons deux :

1) la programmation en **langage ASSEMBLEUR**

2) la programmation en **langage BASIC**.

Avantages et inconvénients de la programmation en langage ASSEMBLEUR

Avantages :
La programmation en langage ASSEMBLEUR se fait à l'aide d'un outil de programmation entièrement gratuit et que l'on peut diffuser librement.
Cet outil (qui est un magnifique environnement de programmation complet) s'appelle **MPLAB**.
Il est disponible en téléchargement gratuit sur le site de Microchip.

Inconvénients :
Pour programmer en langage ASSEMBLEUR, il faut non seulement connaître le fonctionnement de chaque instruction, mais aussi l'architecture interne du microcontrôleur, la structure de sa mémoire, les adresses des registres spéciaux, le fonctionnement de chacune de ses ressources internes, etc..
La programmation en langage ASSEMBLEUR s'appuie sur des organigrammes plus travaillés, et requiert plus de rigueur et de minutie.
Le programmeur doit plus faire attention aux *impératifs machine* qu'à la finalité de son programme.
Distrait par le impératifs machine, le programmeur commet souvent des erreurs.
Ces erreurs sont souvent difficiles à déceler et à corriger.

Avantages et inconvénients de la programmation en langage BASIC

Avantages :
La programmation en BASIC se fait à l'aide d'un langage facile et direct qui (bien qu'étant de l'anglais) comprend des mots puissants, si bien qu'un programme écrit en BASIC comporte peu de mots.
Les erreurs de programmation sont plus rares, et se décèlent facilement.
L'écriture des programmes prend peu de temps.

Inconvénients :
La programmation en BASIC nécessite un **COMPILATEUR** expressément conçu pour la programmation des PIC.
Il s'agit d'un produit commercial, fruit d'un travail d'équipe, et donc payant.
Les programmes en langage BASIC, bien que très courts pour le programmeur qui les écrit, demandent plus de place EEPROM car, vus côté PIC, ils demandent plus d'instructions élémentaires. A tel point que parfois un microcontrôleur pouvant contenir à l'aise un programme écrit en langage assembleur, s'avère posséder une mémoire insuffisante s'il était programmé en langage BASIC, pour faire la même chose.

Les outils nécessaires pour programmer en langage ASSEMBLEUR

Pour programmer en langage ASSEMBLEUR il faut :

1) un PC et une imprimante, pour écrire les instructions permettant de confectionner le fichier à extension . asm

2) un **ASSEMBLEUR** fourni gratuitement par Microchip (à télécharger sur INTERNET) permettant de confectionner le fichier à extension .hex
Cet ASSEMBLEUR s'appelle **MPLAB**.
Il faut l'installer sur votre PC et apprendre à vous en servir.

3) un **PROGRAMMATEUR** de PIC.
Relativement simple à réaliser.
Il existe des modèles pour port série et des modèles pour port parallèle.
Je vous conseille un modèle pour port parallèle.
Cherchez un schéma sur un magazine d'Electronique, ou achetez un kit.

4) un **LOGICIEL** adapté à votre programmateur de PIC.
Si vous achetez un kit, il vous sera fourni avec le kit.
Si vous copiez le schéma dans un magazine, vous devez pouvoir télécharger le logiciel à l'adresse citée dans l'article.

5) Des câbles de liaison et une petite alimentation (un bloc secteur).

Les outils nécessaires pour programmer en langage BASIC

Pour programmer en langage BASIC il faut :

1) un PC et une imprimante, pour écrire les instructions permettant de confectionner le fichier à extension .bas

2) un **COMPILATEUR PicBASIC** proposé par Micro Engineering Labs (dont l'importateur exclusif pour la France est **SELECTRONIC** à Lille) permettant de confectionner le fichier à extension .hex

3) un **PROGRAMMATEUR** de PIC
(mêmes remarques qu'à propos des outils pour programmer en langage ASSEMBLEUR)

4) un **LOGICIEL** adapté à votre programmateur de PIC
(idem)

5) Des câbles de liaison et une petite alimentation (un bloc secteur).

STRUCTURE d'un PROGRAMME

L'écriture d'un programme implique l'élaboration d'une véritable structure.
C'est pourquoi je ne saurais trop vous conseiller d'agir avec méthode et précision.

Tout programme doit comporter un titre : une sorte de définition succincte de ce que fait le programme.

L'étape suivante consiste à mettre ce programme sur papier (*listing*).
Nous y ajouterons des commentaires, ligne par ligne, pour chaque opération effectuée.
Ceux-ci seront clairs et abondants. Ils vous aideront – plus tard – à comprendre les détails du programme.
Ils doivent être tels que si vous repreniez votre feuille plusieurs mois après, vous devriez facilement savoir vous relire et comprendre.
Mieux encore : si vous travaillez en équipe, n'importe qui de votre équipe devrait être en mesure de comprendre de quoi il s'agit.

Prenez l'habitude de signer et dater vos programmes.

Dans la mesure du possible, accompagnez-les d'un organigramme.

CANEVAS d'un programme

Tout programme se construit selon un modèle, une sorte de squelette (*template*, en anglais).

Voici le squelette d'un programme pour PIC 16F84 :

```
Processor 16F84
Déclarations obligatoires        Radix .....
                                 Include <<P16F84>>

Equivalences                     ..... EQU .....

Initialisation de la RAM
et réservation d'un              ORG 0C
certain nombre d'adresses        RES ...
mémoire

Début du programme               ORG 00
après Reset

Configuration des lignes
de port

Instructions

Sous programmes

Fin du programme                 END
```

PROGRAMMATION
en langage ASSEMBLEUR

La programmation en langage ASSEMBLEUR se fait en utilisant les 37 instructions formant son dictionnaire.

En langage ASSEMBLEUR le PIC 16F84 ne comprend que ces 37 mots (en fait : 35 instructions communes à tous les modèles de PIC, plus deux instructions spécifiques au 16F84 : OPTION et TRIS).

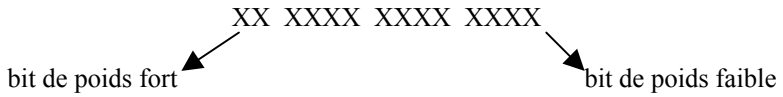
Ces 37 mots forment ce que l'on appelle le *set d'instructions* du 16F84.
Il convient de toutes les connaître.

Ecrire un programme en langage ASSEMBLEUR revient donc à détailler au PIC ce qu'il doit faire, en le disant exclusivement au moyen de ces 37 mots de son vocabulaire : les seuls mots qu'il est capable de comprendre.

Examinons-les une par une.

Les INSTRUCTIONS du 16F84

Le microcontrôleur 16F84 possède un set de seulement 37 instructions codées (en représentation binaire) sur 14 bits, selon le modèle :



Codées en hexadécimal, elles prennent la forme :

XXXX

A remarquer qu'étant donné que les deux bits de poids fort (bits 12 et 13) ne peuvent prendre que seulement quatre valeurs binaires (00 - 01 - 10 et 11), il en résulte que la première valeur de toute instruction codée en hexadécimal ne peut dépasser 3.
Autrement dit : eu égard à la première valeur de chaque instruction codée en hexadécimal, les seuls formats possibles sont :

0XXX...
1XXX...
2XXX...
3XXX...

avec une étendue comprise entre 0000 et 3FFF.

La plupart des instructions opèrent en utilisant le registre de travail W (*Working register*) comparable à l'accumulateur des anciens microprocesseurs, et soit un registre soit une valeur immédiate codée sur 8 bits appelée *literal*.
Le résultat des opérations peut être envoyé soit dans le registre W (accumulateur) soit dans le registre sollicité (soit dans les deux, avec certaines instructions).
Un petit nombre d'instructions opèrent en utilisant uniquement un registre (c'est le cas des instructions BCF, BSF, BTFSC, BTFSS, CLRW, CLRWT et SLEEP).

Les 37 instructions du 16F84 peuvent être classées comme on veut.

Je vous propose quatre types de classement:

a) classement par ordre alphabétique
b) classement par genre
c) classement par type
d) classement par ordre croissant d'encodage.

Toutes les instructions sont codées en un seul mot de 14 bits (0 à 13).

Elles sont toutes exécutées en un seul cycle d'horloge , sauf CALL, GOTO, RETFIE, RETLW et RETURN qui demandent 2 cycles, et BTFSC, BTFSS, DECFSZ, INCFSZ qui – selon le cas – peuvent demander soit un cycle, soit deux cycles.

NB : Parmi les 37 instructions constituant le set du 16F84, deux lui sont spécifiques (je l'ai déjà dit) et ont un caractère spécial : OPTION et TRIS.
Ces deux instructions ne figurent pas dans les autres modèles de PIC. Aussi Microchip recommande de ne pas les utiliser, dans le but de laisser compatibles les programmes (écrits pour ce μC) avec ceux écrits pour d'autres modèles de PIC.
Il suffit de le savoir.
Mais ceci n'est pas un obstacle pour nous, du fait que notre intérêt est exclusivement tourné vers le 16F84.

a) Classement par lettre alphabétique

ADDLW	ADD Literal to W
ADDWF	ADD W to File
ANDLW	AND Literal and W
ANDWF	AND W and File
BCF	Bit Clear File
BSF	Bit Set File
BTFSC	Bit Test File, Skip if Clear
BTFSS	Bit Test File, Skip if Set
CALL	CALL subroutine
CLRF	CLeaR File
CLRW	CLeaR W
CLRWDT	CLeaR Wach Dog Timer
COMF	COMplement File
DECF	DECrement File
DECFSZ	DECrement File, Skip if Zero
GOTO	
INCF	INCrement File
INCFSZ	INCrement File, Skip if Zero
IORLW	Inclusive OR Literal with W
IORWF	Inclusive OR W with File
MOVF	MOVe File
MOVLW	MOVe Literal to W
MOVWF	MOVe W to File
NOP	No OPeration
OPTION	load OPTION register
RETFIE	RETurn From IntErrupt
RETLW	RETurn from subroutine with Literal in W
RETURN	RETURN from subroutine
RLF	Rotate Left File

RRF	Rotate Right File
SLEEP	
SUBLW	SUBtract Literal with W
SUBWF	SUBtract W from File
SWAPF	SWAP File
TRIS	TRIState port
XORLW	eXclusive OR Literal and W
XORWF	eXclusive OR W and File

b) Classement par genre

- *instructions arithmétiques :*
  - ADDLW
  - ADDWF
  - SUBLW
  - SUBWF

- *instructions d'incrémentation :*
  - DECF
  - DECFSZ
  - INCF
  - INCFSZ

- *instructions d'effacement :*
  - CLRF
  - CLRW
  - CLRWDT

- *instructions de mouvement :*
  - MOVF
  - MOVLW
  - MOVWF

- *instructions de rotation :*
  - RLF
  - RRF

- *instructions logiques :*
  - ANDLW
  - ANDWF
  - COMF
  - IORLW
  - IORWF
  - XORLW
  - XORWF

- *instructions de saut et branchement :*
  - CALL
  - GOTO
  - RETFIE
  - RETLW
  - RETURN

- *instructions agissant sur les bits :*
  - BCF
  - BSF
  - BTFSC
  - BTFSS

- *instructions diverses :*
  - NOP
  - OPTION
  - TRIS
  - SLEEP
  - SWAPF

c) Classement par type

- *instructions travaillant seules :*

CLRW
CLRWDT
NOP
OPTION
RETFIE
RETURN
SLEEP

- *instructions travaillant avec une constante ou une étiquette* (**k**) *:*

ADDLW k
ANDLW k
CALL k
GOTO k
IORLW k
MOVLW k
RETLW k
SUBLW k
XORLW k

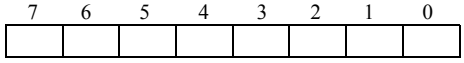
- *instructions travaillant avec un registre* (**f**) *:*

CLRF f
MOVWF f
TRIS f

- *instructions travaillant sur un bit* (**b**) *d'octet* (**f**) *:*

BCF f,b
BSF f,b
BTFSC f,b
BTFSS f,b

Il est évident que **b** ne peut prendre qu'une valeur comprise entre 0 et 7, désignant la position du bit dans l'octet :



- *instructions travaillant avec un registre* (**f**) *mais proposant un choix* (**d**) *dans la destination du résultat :*
- **d** = 0 *: le résultat est placé dans* W
- **d** = 1 *: le résultat est placé dans le registre* **f**

ADDWF f,d
ANDWF f,d
COMF f,d
DECF f,d
DECFSZ f,d
INCF f,d
INCFSZ f,d
IORWF f,d
MOVF f,d
RLF f,d
RRF f,d
SUBWF f,d
SWAPF f,d
XORWF f,d

d) Classement par ordre croissant d'encodage

0	1	2	3
0000 NOP	1xxx BCF	2xxx CALL	30xx MOVLW
0008 RETURN	1xxx BSF	2xxx GOTO	34xx RETLW
0009 RETFIE	1xxx BTFSC		38xx IORLW
0062 OPTION	1xxx BTFSS		39xx ANDLW
0063 SLEEP			3Axx XORLW
0064 CLWDT			3Cxx SUBLW
006x TRIS			3Exx ADDLW
00xx MOVWF			
0100 CLRW			
01xx CLRF			
02xx SUBWF			
03xx DECF			
04xx IORWF			
05xx ANDWF			
06xx XORWF			
07xx ADDWF			
08xx MOVF			
09xx COMF			
0Axx INCF			
0Bxx DECFSZ			
0Cxx RRF			
0Dxx RLF			
0Exx SWAPF			
0Fxx INCFSZ			

ADDLW

ADD Literal to W

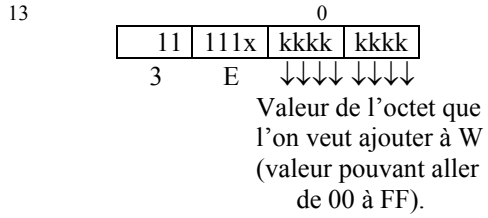
- Additionne de manière immédiate le *literal* au contenu du registre W, et place le résultat dans W.

- Le *literal* est un mot de 8 bits (de 00 à FF).

- Cette instruction affecte 3 bits du registre d'état :
  le flag C : Carry
  le flag DC : Digit Carry
  le flag Z : Zero

- 1 cycle d'horloge

- Encodage de l'instruction :



- Exemple de programmation :

```
ADDLW    06
```

En supposant que W contienne 04 avant l'instruction, après l'instruction il contient 0A (en hexa : 06 + 04 = 0A).

ADDWF

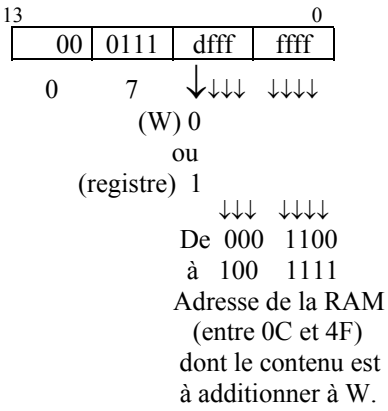
ADD W to File

- Additionne le contenu du registre W à l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour faire l'addition (la nouvelle valeur prend la place de l'ancienne).

- Cette instruction affecte 3 bits du registre d'état :
le flag C : Carry
le flag DC : Digit Carry
le flag Z : Zero

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

1) MOVF VentesDuMois,W
ADDWF,0

```
2)          MOVF        VentesDuMois,W
            ADDWF,1
```

L'exemple 1 met dans W ce qui se trouve à l'adresse VentesDu Mois, puis additionne VentesDuMois au contenu de W, *et range le résultat dans W*.

L'exemple 2 met dans W ce qui se trouve à l'adresse VentesDu Mois, puis additionne VentesDuMois au contenu de W, *et range le résultat à l'adresse VentesDuMois*, en renplaçant l'ancienne valeur par le total qu'on vient de trouver.

ANDLW

AND Literal and W

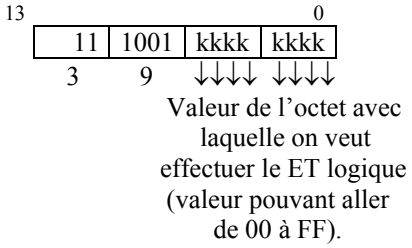
- Effectue une opération logique ET (AND) entre la valeur immédiate du *literal* et l'octet se trouvant dans le registre W.

- Le *literal* est un mot de 8 bits (de 00 à FF)

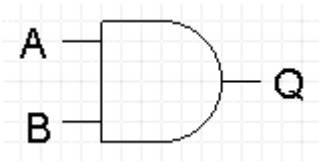
- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction :



- Table de vérité d'une porte ET :



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

- Exemple de programmation :

ANDLW A7

En supposant que W contienne 5C (01011100) avant l'instruction, après l'instruction W contient 04.

A7 = 10100111
5C = 01011100
AND = 00000100 (04 en hexa)

Pourquoi cette instruction dans le set du 16F84 ? A quoi sert-elle ?

Elle sert lorsque – dans un octet – on a besoin de récupérer un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à éliminer) et de 1 (aux emplacements où se trouvent les bits à récupérer).

Exemple : on souhaite récupérer uniquement le bit 5 de l'octet 01111010.

On prépare alors le *masque* 00100000 et on fait un ET logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	00100000	(*masque*)
Ce qui donne :	00100000	(résultat)

Le résultat de l'opération permet donc bien de récupérer uniquement le bit 5 de l'octet : ici c'est un 1 (00**1**00000).

Une fois récupéré, on peut utiliser ce bit comme on veut.

ANDWF

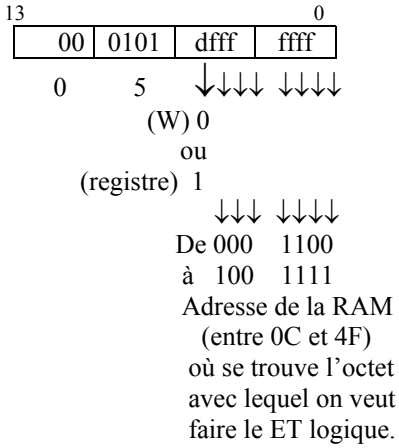
AND W and File

- Effectue une opération logique ET (AND) entre l'octet se trouvant dans le registre W et l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour effectuer le ET logique (la nouvelle valeur prend la place de l'ancienne).

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

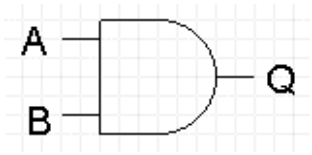
1) ANDWF,0 Adresse

2) ANDWF,1 Adresse

L'exemple 1 effectue un ET logique entre l'octet se trouvant dans W et l'octet se trouvant à Adresse, *et range le résultat dans W*.

L'exemple 2 effectue un ET logique entre l'octet se trouvant dans W et l'octet se trouvant à Adresse, *et range le résultat dans Adresse*.

- Table de vérité d'une porte ET :



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Pourquoi cette instruction dans le set du 16F84 ? A quoi sert-elle ?

Elle sert lorsque – dans un octet – on a besoin de récupérer un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à éliminer) et de 1 (aux emplacements où se trouvent les bits à récupérer).

Exemple : on souhaite récupérer uniquement le bit 5 de l'octet 01111010.

On prépare alors le *masque* 00100000 et on fait un ET logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	00100000	(*masque*)
Ce qui donne :	00100000	(résultat)

Le résultat de l'opération permet donc bien de récupérer uniquement le bit 5 de l'octet : ici c'est un 1 (00**1**00000).

Une fois récupéré, on peut utiliser ce bit comme on veut.

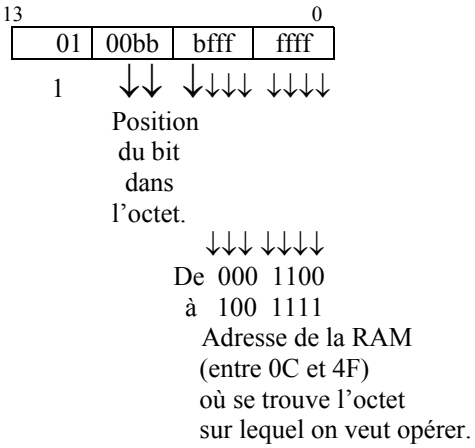
BCF

Bit Clear File

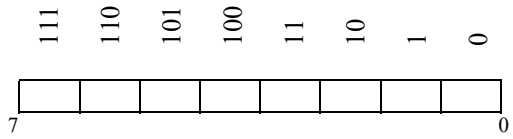
- Met à zéro le bit désigné de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F).

- 1 cycle d'horloge

- Encodage de l'instruction:



- Position du bit dans l'octet :



- Exemples de programmation :

En supposant qu'on veuille mettre à zéro (Clear) un certain bit de l'octet situé à l'adresse 27 de la mémoire RAM, la programmation serait :

```
        BCF   27,0    (pour mettre à zéro le bit 0)
ou      BCF   27,1    (pour mettre à zéro le bit 1)
        ou    BCF     27,2    (pour mettre à zéro le bit 2)
        etc...
```

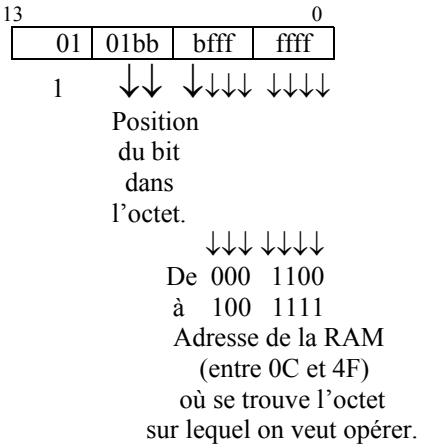
BSF

Bit Set File

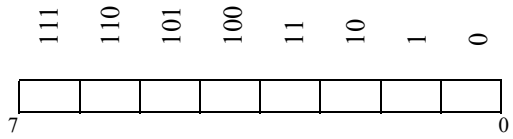
- Met à 1 (Set) le bit désigné de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F).

- 1 cycle d'horloge

- Encodage de l'instruction:



- Position du bit dans l'octet :



- Exemples de programmation :

En supposant qu'on veuille mettre à 1 (Set) un certain bit de l'octet situé à l'adresse 1C de la mémoire RAM, la programmation serait :

```
        BSF     1C,0    (pour mettre à 1 le bit 0)
ou      BSF     1C,1    (pour mettre à 1 le bit 1)
ou      BSF     1C,2    (pour mettre à 1 le bit 2)
        etc...
```

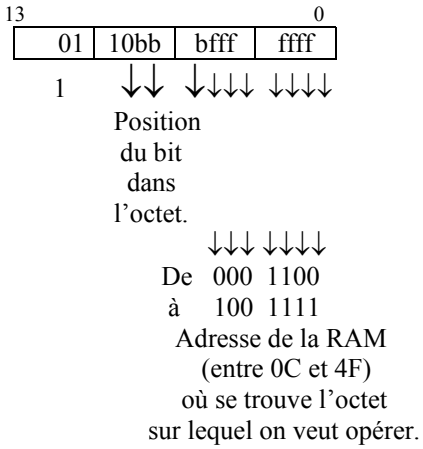
BTFSC

Bit Test File, Skip if Clear

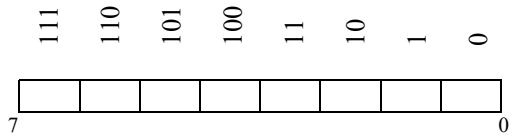
- Vérifie l'état logique du bit désigné de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F).
Est-il à zéro ?
S'il est à zéro, ignore l'instruction suivante.

- Selon que la réponse soit *OUI* ou *NON*, l'instruction prend 1 ou 2 cycles d'horloge.

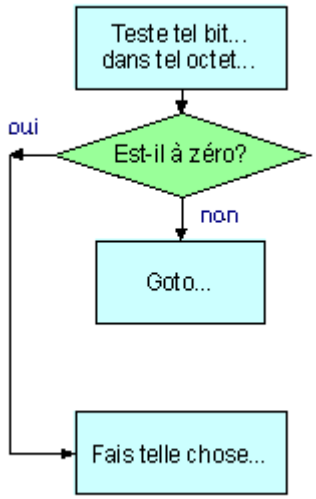
- Encodage de l'instruction:



- Position du bit dans l'octet :



- Organigramme du traitement :



- Exemple de programmation :

En supposant que l'octet dont on veut tester un bit soit situé à l'adresse 1A, la programmation serait la suivante :

BTFSC 1A,*0* (pour tester le bit 0)

ou BTFSC 1A,*1* (pour tester le bit 1)

ou BTFSC 1A,*2* (pour tester le bit 2)

etc...

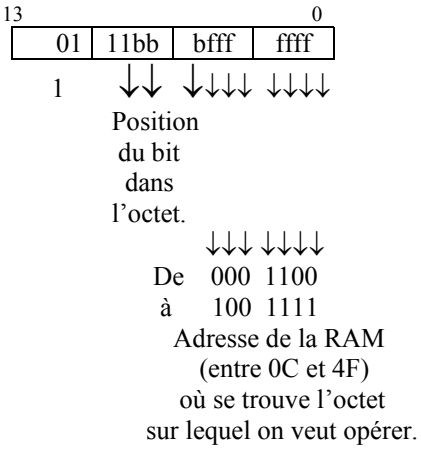
BTFSS

Bit Test File, Skip if Set

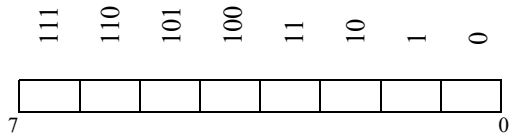
- Vérifie l'état logique du bit désigné de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F).
Est-il à 1 ?
S'il est à 1, ignore l'instruction suivante.

- Selon que la réponse soit *OUI* ou *NON*, l'instruction prend 1 ou 2 cycles d'horloge.

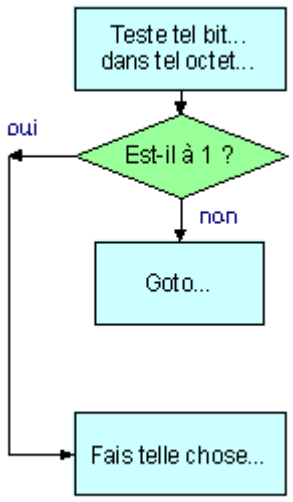
- Encodage de l'instruction:



- Position du bit dans l'octet :



- Organigramme du traitement :



- Exemple de programmation :

En supposant que l'octet dont on veut tester un bit soit situé à l'adresse 1C, la programmation serait la suivante :

```
        BTFSS   1C,0    (pour tester le bit 0)
ou      BTFSS   1C,1    (pour tester le bit 1)
ou      BTFSS   1C,2    (pour tester le bit 2)
        etc...
```

CALL

CALL subroutine

- Appel à sous-programme.

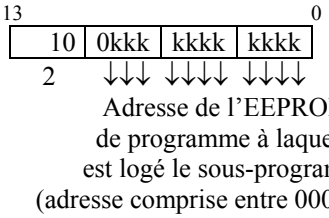
- Le µC sauvegarde l'adresse de retour dans la pile (stack), puis charge dans le PC (Program Counter) l'adresse à laquelle il est invité à se rendre. Il peut s'agir aussi bien d'une adresse que d'une label ; et c'est là que démarre le sous-programme.

- Tout sous-programme appelé par l'instruction CALL doit obligatoirement se terminer soit par l'instruction RETURN, soit par l'instruction RETLW qui renvoient au programme principal.
Ne pas confondre l'instruction CALL avec l'instruction GOTO.
L'instruction CALL fait toujours revenir le programme principal à l'endroit où il avait été abandonné ; tandis que l'instruction GOTO provoque l'abandon total de la séquence et peut conduire soit à une toute autre action, soit à l'arrêt total du programme.

- La pile (stack) est une zone de mémoire ne pouvant contenir que 8 mots de 13 bits.
Ceci limite à 8 niveaux les possibilités d'imbrication. S'il y en avait un neuvième, la première adresse de retour serait perdue...

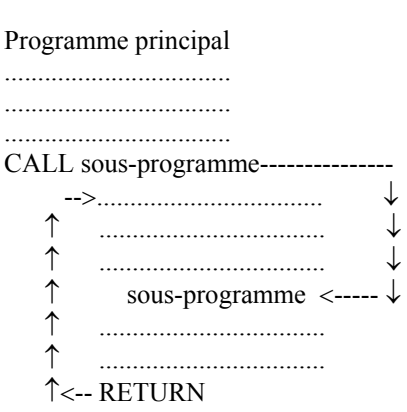
- Cette instruction prend 2 cycles d'horloge.

- Encodage de l'instruction :



Adresse de l'EEPROM
de programme à laquelle
est logé le sous-programme
(adresse comprise entre 000 et 3FF).

- Organigramme du traitement :



- Exemple de programmation :

CALL Tempo (saute *à l'adresse* correspondant à Tempo).

CLRF

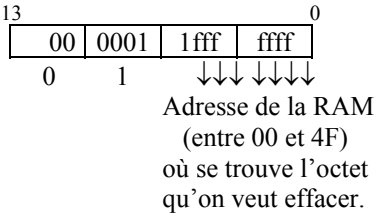
CLeaR File

- Efface (Clear) ce qui se trouve (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 00 et 4F).

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction :



- Exemples de programmation :

1) CLRF INTCON (pour désactiver les interruptions)

2) CLRF PORTB (pour mettre à zéro tous les bits du port B)

3) CLRF 1E (pour effacer ce qui se trouve à l'adresse 1E)

CLRW

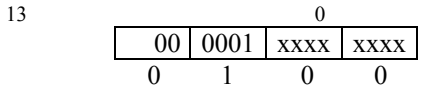
CLeaR W

- Efface (Clear) le registre W.

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction :



- Exemple de programmation :

CLRW (efface le registre W).

En supposant que W contienne F8 avant l'instruction, après l'instruction il contient 00.

CLRWDT

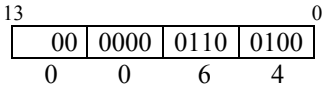
CLeaR Wach Dog Timer

- Met à zéro le compteur du chien de garde (ainsi que celui du pré diviseur, si celui-ci est activé).

- Cette instruction affecte deux bits du registre d'état :
- le flag TO (Time Out) passe à 1
- le flag PD (Power Down) passe à 1

- 1 cycle d'horloge

- Encodage de l'instruction :



- Exemple de programmation :

CLWDT (efface le compteur du chien de garde).

Peu importe où en était le compteur du chien de garde, cette instruction le fait revenir à zéro.

COMF

COMplement File

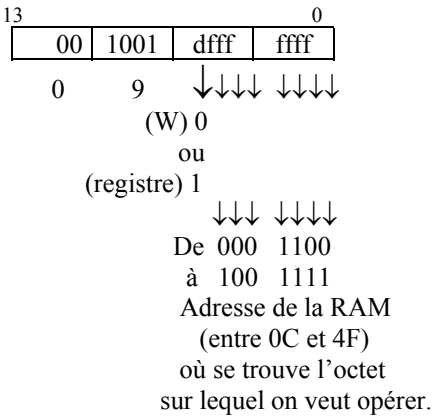
- Effectue un complément bit à bit sur l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour faire le complément (la nouvelle valeur prend la place de l'ancienne).

Faire le complément bit à bit d'un octet équivaut à changer ses zéros en 1, et inversement.
Exemple : le complément de 00111100 est 11000011.

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

1) COMF 2B,*0* (effectue le complément bit à bit de l'octet situé à l'adresse 2B et range le résultat dans W)

En supposant que 2B contienne 11100000 avant l'instruction, après l'instruction cette valeur est transformée en 00011111.

2) COMF	2B,*1* (même chose, mais le résultat est rangé à la place de l'octet utilisé pour faire le complément).

DECF

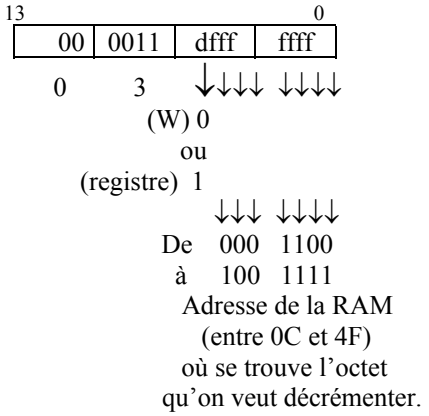
DECrement File

- Décrémente la valeur de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet qu'on a décrémenté (la nouvelle valeur prend la place de l'ancienne).

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

1) DECF COMPTEUR,*0* (décrémente l'octet se trouvant à l'adresse COMPTEUR, et range le résultat dans W)

2) DECF COMPTEUR,*1* (même chose, mais cette fois le résultat est rangé à l'adresse COMPTEUR).

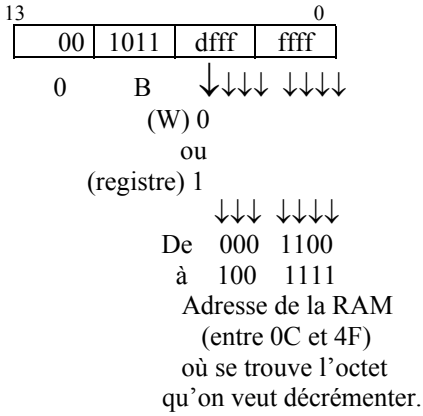
DECFSZ

DECrement File, Skip if Zero

- Décrémente la valeur de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F), et effectue un test : l'octet a-t-il atteint zéro ?
Si OUI, ignore l'instruction suivante.
Avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet décrémenté (la nouvelle valeur prend la place de l'ancienne).

- Selon qu'à la suite de la décrémentation l'octet ait atteint ou pas la valeur zéro, l'instruction prend 1 ou 2 cycles d'horloge.

- Encodage de l'instruction:



- Cette instruction est généralement suivie par l'instruction CALL.

- Exemples de programmation :

1) DECFSZ 2F,*0* (décrémente l'octet se trouvant à l'adresse 2F, et range le résultat dans W. Si le résultat est zéro, ignore l'instruction suivante).

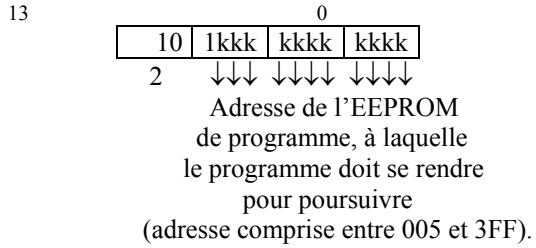
2) DECFSZ 2F,*1* (même chose, mais cette fois le résultat est rangé à l'adresse 2F. La nouvelle valeur prend la place de l'ancienne).

GOTO

- Branchement inconditionnel.
Va de façon inconditionnelle à l'adresse indiquée de suite (adresse de démarrage du sous-programme). Il peut s'agir aussi bien d'une adresse que d'une label.
Cette instruction interrompt l'exécution séquentielle des instructions et oblige à poursuivre le programme d'une adresse complètement ailleurs.
A la différence de l'instruction CALL (qui fait toujours revenir le programme principal à l'endroit où il avait été abandonné) , l'instruction GOTO provoque l'abandon complet de la séquence et peut conduire soit à une toute autre action, soit à l'arrêt total du programme.

- Cette instruction prend 2 cycles d'horloge.

- Encodage de l'instruction :



- Exemple de programmation

```
GOTO   ALLUMAGE
```

Après cette instruction, le PC (Program Counter) est chargé avec la *valeur de l'adresse* à laquelle commence le programme ALLUMAGE.

INCF

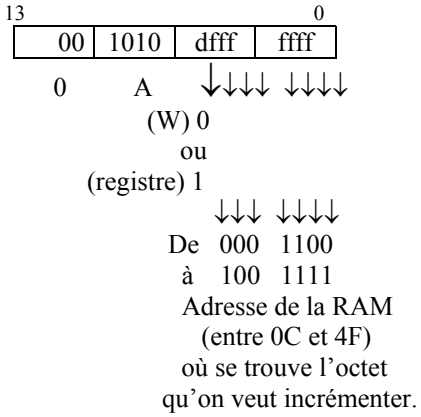
INCrement File

- Incrémente la valeur de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet qu'on a incrémenté (la nouvelle valeur prend la place de l'ancienne).

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

1) INCF NOMBRE,*0* (incrémente l'octet se trouvant à l'adresse NOMBRE, et range le résultat dans W)

2) INCF NOMBRE,*1* (même chose, mais cette fois le résultat est rangé à l'adresse NOMBRE. La nouvelle valeur prend la place de l'ancienne).

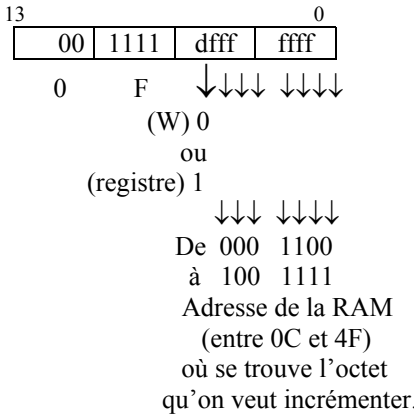
INCFSZ

INCcrement File, Skip if Zero

- Incrémente la valeur de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F), et effectue un test : l'octet a-t-il atteint zéro ?
Si OUI, ignore l'instruction suivante.
Avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet incrémenté (la nouvelle valeur prend la place de l'ancienne).

- Selon qu'à la suite de l'incrémentation l'octet ait atteint ou pas la valeur zéro, l'instruction prend 1 cycle ou deux d'horloge.

- Encodage de l'instruction:



- Cette instruction est généralement suivie par CALL.

- Exemples de programmation :

1) INCFSZ DATE,*0* (incrémente l'octet se trouvant à l'adresse DATE, et range le résultat dans W. Si le résultat est zéro, ignore l'instruction suivante).

2) INCFSZ DATE,*1* (même chose, mais cette fois le résultat est rangé à l'adresse DATE. La nouvelle valeur prend la place de l'ancienne).

IORLW

Inclusive OR Literal with W

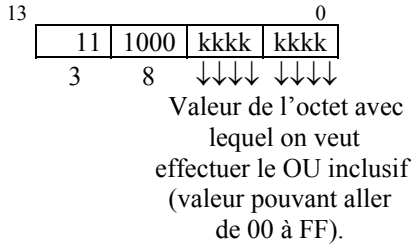
- Effectue une opération logique OU inclusif (Inclusive OR) entre la valeur immédiate du literal et l'octet se trouvant dans le registre W.
Le résultat de l'opération reste dans le registre W.

- Le literal est un mot de 8 bits (de 00 à FF)

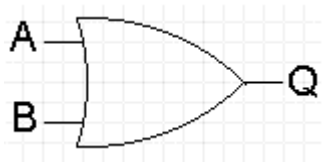
- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction :



- Table de vérité d'une porte OU inclusif :



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

- Exemple de programmation :

IORLW B5 (10110101)

En supposant que W contienne 49 (01001001) avant l'instruction, après l'instruction il contient FD.

B5 = 10110101
49 = 01001001
OR = 11111101 (FD en hexa)

Pourquoi cette instruction dans le set du 16F84 ? A quoi sert-elle ?

Elle sert lorsque – dans un octet – on a besoin de *forcer à 1* un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à ignorer) et de 1 (aux emplacements où se trouvent les bits qu'on veut *forcer à 1*).

Exemple : on souhaite *forcer à 1* les bits 7 et 6 de l'octet 01111010.

On prépare alors le *masque* 11000000 et on fait un OU logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	11000000	(*masque*)
Ce qui donne :	11111010	(résultat)

Le résultat de l'opération permet donc bien de forcer à 1 les bits 7 et 6 de l'octet. Il se trouve qu'ici le bit 6 était déjà à 1. Mais le programme ne le savait pas. L'instruction IORLW permet de préciser les choses.

Une fois forcés à 1, on peut utiliser ces bits (ou l'octet) comme on veut.

IORWF

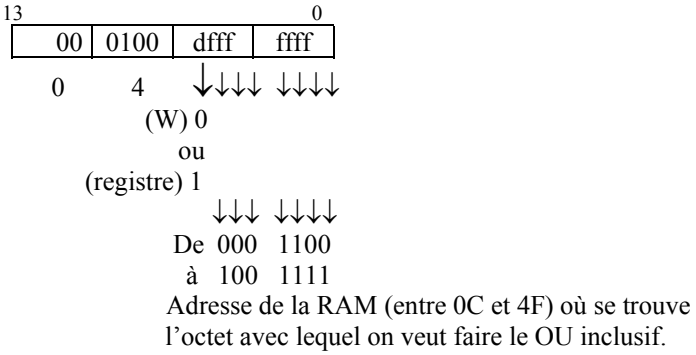
Inclusive OR With File

- Effectue une opération logique OU inclusif (Inclusive OR) entre l'octet se trouvant dans le registre W et l'octet situé (en mémoire RAM) à l'adresse située de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être placé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour faire le OU inclusif (la nouvelle valeur prend la place de l'ancienne).

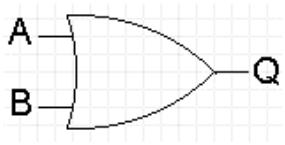
- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Table de vérité d'une porte OU inclusif :



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

- Exemples de programmation :

1) IORWF 29,*0* (effectue le OU inclusif entre l'octet se trouvant dans W et celui situé à l'adresse 29, et range le résultat dans W).

En supposant que le contenu de l'adresse 29 soit C7 (11000111) et que W contienne 69 (01101001), après l'instruction on obtient EF (11101111).

2) IORWF 29,*1* (même chose, mais le résultat est rangé à l'adresse 29, à la place de l'octet utilisé pour faire le OU inclusif).

Pourquoi cette instruction dans le set du 16F84 ? A quoi sert-elle ?

Elle sert lorsque – dans un octet – on a besoin de *forcer à 1* un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à ignorer) et de 1 (aux emplacements où se trouvent les bits qu'on veut *forcer à 1*).

Exemple : on souhaite *forcer à 1* les bits 7 et 6 de l'octet 01111010.

On prépare alors le *masque* 11000000 et on fait un OU logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	11000000	(*masque*)
Ce qui donne :	11111010	(résultat)

Le résultat de l'opération permet donc bien de forcer à 1 les bits 7 et 6 de l'octet. Il se trouve qu'ici le bit 6 était déjà à 1. Mais le programme ne le savait pas. L'instruction IORLW permet de préciser les choses.

Une fois forcés à 1, on peut utiliser ces bits (ou l'octet) comme on veut.

MOVF

MOVe File

- Cette instruction peut faire deux choses :

1) soit porter dans W le contenu situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F), avec l'option de programmation *,0*

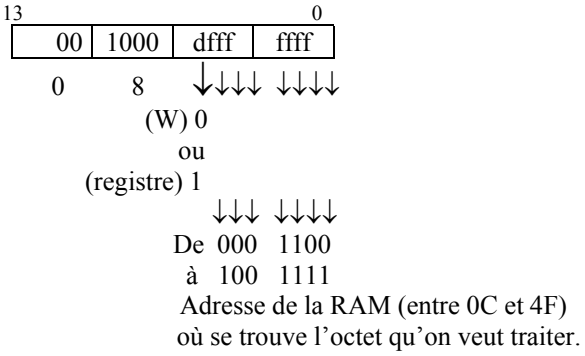
2) soit copier l'octet sur lui-même au même emplacement de la RAM, avec l'option de programmation *,1*

Bien que ça paraisse comique de copier le contenu d'un registre sur lui-même, en fait - étant donné que cette opération modifie le bit Z du registre d'état - elle est utile quand on a besoin de faire un test à zéro sur l'octet, en toute sécurité.

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



- Exemples de programmation :

1)	MOVF	PORTB,*0*	(porte dans W l'état es lignes du port B).
1)	MOVF	18,*1*	(copie le contenu de l'adresse 18 sur lui-même).

MOVLW

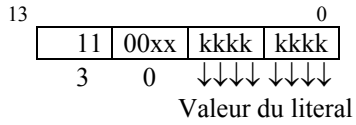
MOVe Literal to W

- Charge de manière immédiate le literal dans le registre W

- Le literal est un mot de 8 bits (de 00 à FF)

- 1 cycle d'horloge

- Encodage de l'instruction :



- Exemple de programmation :

MOVLW F5 (met F5 , c'est à-dire 11110101, dans W).

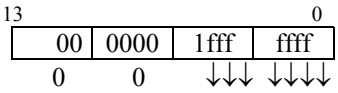
MOVWF

MOVe W to File

- Prend le contenu du registre W et le met (dans la mémoire RAM) à l'adresse indiquée de suite (adresse de 0C à 4F)

- 1 cycle d'horloge

- Encodage de l'instruction:



Adresse de la RAM (entre 0C et 4F) à laquelle on veut mettre la valeur du registre W.

- Exemple de programmation :

MOVWF	0D	(écrit le contenu de W à l'adresse RAM 0D).

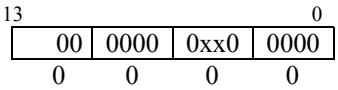
NOP

No OPeration

- L’instruction la plus paresseuse !
Ne fait rien.
Elle sert juste à occuper le processeur pour laisser passer un peu de temps (1 cycle d’horloge).
S’utilise essentiellement pour créer des temporisations.

- 1 cycle d’horloge

- Encodage de l’instruction:



- Exemple de programme:

```
NOP
```

OPTION

load OPTION register

- NB : *cette instruction est spécifique au 16F84*.

Microchip recommande de ne pas l'utiliser, dans le but de laisser les programmes (écrits pour ce type de microcontrôleur) compatibles avec ceux écrits pour d'autres modèles de PIC.
Il suffit de le savoir.
Mais ceci n'est pas un obstacle pour nous, du fait que notre intérêt est exclusivement tourné vers le 16F84.

- Charge le registre OPTION, c'est à-dire le registre qui sert à configurer le TMR0 (l'horloge interne du microcontrôleur) ainsi que le prédiviseur.

- S'agissant d'un registre à lecture/écriture simultanée, on ne peut pas y écrire directement, mais il faut obligatoirement transiter par le registre W.

En programmation, on commence par écrire l'octet de configuration dans W. Puis l'instruction OPTION - en même temps qu'elle adresse ce registre- y copie automatiquement l'octet de configuration.

RETFIE

RETurn From IntErrupt

- Retour au programme principal après exécution d'un sous-programme d'interruption.
Charge le PC (Program Counter : compteur d'instructions) avec la valeur qui se trouve au sommet de la pile (stack) ; ce qui provoque le retour au programme principal.

- Lorsqu'une interruption est demandée, le microcontrôleur, avant de sauter à l'adresse 004 de l'EEPROM mémoire de programme, sauve la valeur du Program Counter dans la pile.
Cette valeur - comme dans une pile d'assiettes - se place tout en haut de la pile (dans laquelle il y a seulement 8 places).
A la fin du sous-programme de réponse à l'interruption, le μC rencontre l'instruction RETFIE par laquelle il récupère la valeur se trouvant au sommet de la pile (correspondant à la dernière valeur entrée) et la positionne dans le Program Counter, faisant ainsi revenir le programme à son flux normal (pile de type LIFO : Last In, First Out).

- Après cette instruction, le pointeur de pile (stack pointer) se positionne tout en haut de la pile, et le bit du GIE (General Interrupt Enable) du registre INTCON (bit7) bascule à 1.

- Cette instruction prend 2 cycles d'horloge

- Encodage de l'instruction:

13			0
00	0000	0000	1001
0	0	0	9

- Exemple de programme :

```
RETFIE
```

RETLW

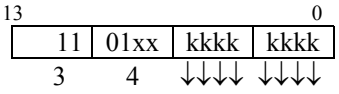
RETurn from subroutine with Literal in W

- Instruction jusqu'à un certain point similaire à RETURN, en ce sens qu'elle ferme un sous-programme et provoque le retour au programme principal à l'endroit où il avait été abandonné ; mais avec une particularité en plus : charge dans le registre W la valeur du literal.

- Le literal est un mot de 8 bits (de 00 à FF)

- Cette instruction prend 2 cycles d'horloge

- Encodage de l'instruction:



Valeur de l'octet avec laquelle on veut charger W en rentrant du sous-programme (valeur pouvant aller de 00 à FF).

RETURN

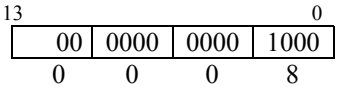
RETURN from subroutine

- Retour d'un sous-programme.
Le PC (Program Counter) est chargé avec l'adresse se trouvant au sommet de la pile.

- C'est l'instruction qui ferme un sous-programme et provoque le retour au programme principal à l'endroit auquel il avait été abandonné.

- Cette instruction prend 2 cycles d'horloge

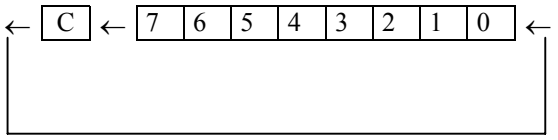
- Encodage de l'instruction:



RLF

Rotate Left File

- Rotation à gauche.



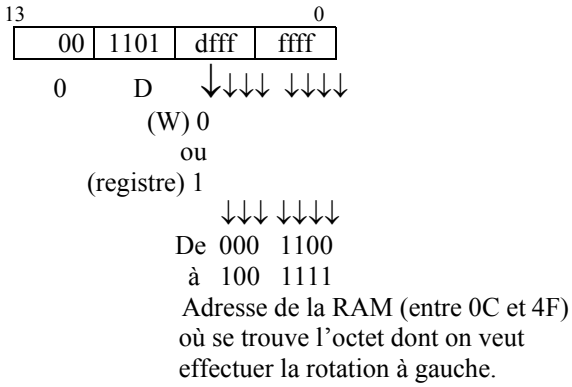
- Effectue le déplacement d'une position vers la gauche des bits de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) en utilisant le bit de CARRY du registre d'état : le contenu du bit de CARRY devient le nouveau bit 0 de l'octet ayant effectué la rotation à gauche, tandis que l'ancien bit 7 entre dans CARRY ; avec deux variantes : le résultat de la rotation peut être rangé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour effectuer la rotation (la nouvelle valeur prend la place de l'ancienne).

- NB : Avant d'utiliser cette instruction il convient de préalablement effacer le bit de CARRY.
Car, à supposer que dans CARRY il ait un 1, après une rotation à gauche de 0000000*1* on aurait 000000*11* alors qu'on s'attendait à 000000*10*.
L'instruction qui efface le bit de CARRY est : BCF STATUS,0
(efface le bit zéro du registre STATUS, c'est à-dire le bit de CARRY).

- Cette instruction affecte le bit C du registre d'état

- 1 cycle d'horloge

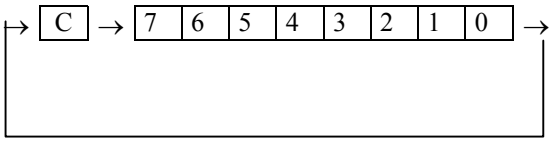
- Encodage de l'instruction:



RRF

Rotate Right File

- Rotation à droite.



- Effectue le déplacement d'une position vers la droite des bits de l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) en utilisant le bit de CARRY du registre d'état : le contenu du bit de CARRY devient le nouveau bit 7 de l'octet ayant effectué la rotation à droite, tandis que l'ancien bit 0 entre dans CARRY ; avec deux variantes : le résultat de la rotation peut être rangé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour effectuer la rotation (la nouvelle valeur prend la place de l'ancienne).

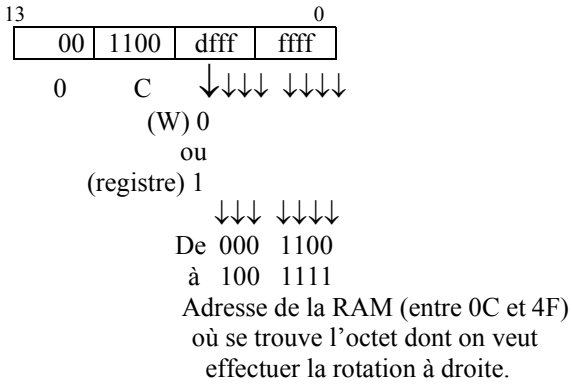
- NB : Avant d'utiliser cette instruction il convient de préalablement effacer le bit de CARRY .

L'instruction qui efface le bit de CARRY est : BCF STATUS,0
(efface le bit zéro du registre STATUS, c'est à-dire le bit de CARRY).

- Cette instruction affecte le bit C du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



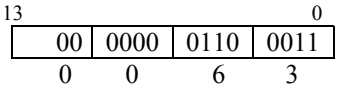
SLEEP

- Mise en veilleuse.
Cette instruction s'utilise non pas pour mettre le µC hors tension, mais pour arrêter le séquencement des instructions (ralentir le signal d'horloge jusqu'à l'extrême limite : la fréquence zéro).
Pendant cette mise en veilleuse, l'horloge externe (faisant partie de ce que nous avons appelé le cortège des invariants) est coupée. Le flux du programme est bloqué.
Seul le chronomètre du Watch Dog (chien de garde) reste actif.
La consommation du boîtier (qui normalement est de 2 mA) tombe à 30 µA.
Parmi les causes pouvant réveiller le µC retenons : une demande d'interruption, ou un signal provenant du chronomètre (timer) du chien de garde.

- Cette instruction affecte deux bits du registre d'état :
TO (Time Out) passe à 1
PD (Power Down) passe à 0
En plus, elle met à zéro le chronomètre du chien de garde, ainsi que le pré diviseur.

- 1 cycle d'horloge

- Encodage de l'instruction:



SUBLW

SUBtract Literal with W

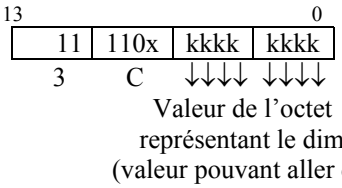
- Soustrait le literal (valeur immédiate représentée par un octet pouvant aller de 00 à FF) du contenu du registre W, et place le résultat dans W.

Literal	diminuende
- W	- diminuteur
= résultat	= différence

- Cette instruction affecte 3 bits du registre d'état :
- le flag C (Carry)
- le flag DC (Digit Carry)
- le flag Z (Zero)

- 1 cycle d'horloge

- Encodage de l'instruction:



SUBWF

SUBtract W from File

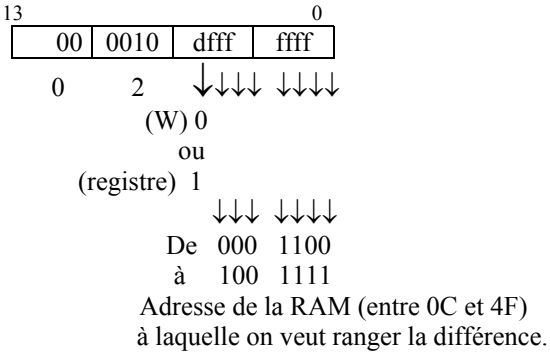
- Soustrait la valeur contenue dans le registre W de la valeur se trouvant (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat (différence) peut être rangé soit dans le registre W, soit dans la mémoire RAM à la place du diminuende.

File (valeur se trouvant à l'emplacement RAM, le diminuende)
- W (valeur contenue dans le registre W, le diminuteur)
Résultat différence

- Cette instruction affecte 3 bits du registre d'état :
- le flag C (Carry)
- le flag DC (Digit Carry)
- le flag Z (Zero)

- 1 cycle d'horloge

- Encodage de l'instruction:



SWAPF

SWAP File

- Echange les quatre bits de poids fort d'un octet se trouvant (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 00 et 4F), avec ses propres quatre bits de poids faible :

1	0	0	0	1	0	1	1

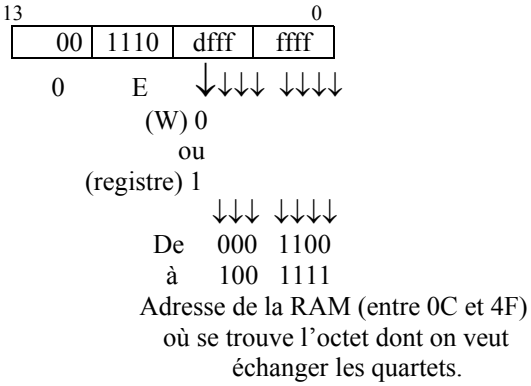
devient

1	0	1	1	1	0	0	0

Avec deux variantes : le résultat de l'échange peut être placé soit dans le registre W, soit dans la mémoire RAM en lieu et place de l'octet utilisé pour effectuer le *swap*.

- 1 cycle d'horloge

- Encodage de l'instruction:



TRIS

TRIState port

- NB : *cette instruction est spécifique au 16F84*.

Microchip recommande de ne pas l'utiliser, dans le but de laisser les programmes (écrits pour ce type de microcontrôleur) compatibles avec ceux écrits pour d'autres modèles de PIC.
Il suffit de le savoir.
Mais ceci n'est pas un obstacle pour nous, du fait que notre intérêt est exclusivement tourné vers le 16F84.

- Charge le registre TRIS (A ou B), et met les lignes de port à haute impédance.
Ce registre *configure*, c'est à dire *définit le sens* de fonctionnement de chacune des lignes des ports A et B ; assigne à chaque ligne soit le rôle d'*entrée*, soit le rôle de *sortie*, sans pour autant provoquer aucune entrée ni aucune sortie.

- Il s'agit d'un registre de 8 bits, pouvant tous se programmer individuellement par 0 ou par 1 :

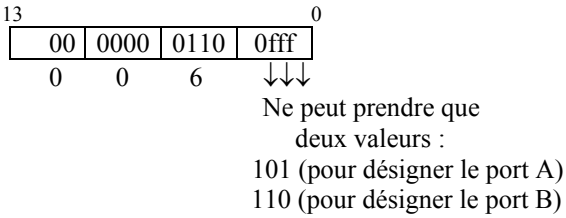
0 = la ligne de port (qui lui correspond) est configurée comme *sortie*

1 = la ligne de port (qui lui correspond) est configurée comme *entrée*

- S'agissant d'un registre à lecture/écriture simultanée, on ne peut pas y écrire directement, mais il faut obligatoirement transiter par le registre W.

En programmation, on commence par écrire l'octet de configuration dans W. Puis l'instruction TRIS (A ou B), en même temps qu'elle adresse ce registre, copie automatiquement l'octet de configuration dans le port A ou dans le port B.

- Encodage de l'instruction :



- Exemple de programmation :

```
MOVLW     00000100  (en binaire)
MOVWF     TRISB
```

On charge dans le registre W l'octet de configuration de port (ligne 2 en entrée, toutes les autres lignes en sortie), que l'instruction TRIS valide.

A partir de ce moment le port est configuré, *mais aucune donnée n'y entre, aucune donnée n'y sort.*
Les lignes du port sont mises en haute impédance (tristate).

XORLW

EXclusive OR Literal and W

- Effectue un OU exclusif (Exclusive OR) entre la valeur immédiate du literal et l'octet se trouvant dans le registre W.
Le résultat est rangé dans W.

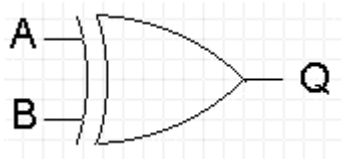
- Le literal est un mot de 8 bits (de 00 à FF).

- Un OR exclusif permet de comparer deux octets bit à bit.
Si les bits de même poids sont au même niveau, le résultat est zéro.
Si par contre ils sont à des niveaux différents, le résultat est 1.
Exemple de XOR entre deux octets :

```
          00110011
          01110010
résultat = 01000001
```

- Un OR exclusif permet d'inverser un état logique.
Exemple :
Soit au départ l'octet 11111111.
Si le XOR se fait avec 00000000 , rien ne change ;
le résultat est : 11111111.
Si par contre le XOR se fait avec 00000001,le résultat est :11111110 (Alors que les zéros ne font rien changer, les 1 provoquent un basculement d'état).

- Table de vérité d'une porte OU exclusif :

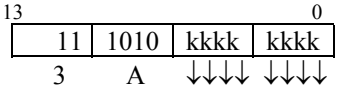


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



Valeur de l'octet avec lequel
on veut effectuer le OU exclusif
(valeur pouvant aller de 00 à FF).

Pourquoi cette instruction dans le set du 16F84 ? A quoi sert-elle pratiquement ?

Elle sert lorsque – dans un octet – on a besoin d'*inverser* un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à ignorer) et de 1 (aux emplacements où se trouvent les bits qu'on veut *inverser*).

Exemple : on souhaite *inverser* les bits 7, 6, 5 et 4 de l'octet 01111010.

On prépare alors le *masque* 11110000 et on fait un OU logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	11110000	(*masque*)
Ce qui donne :	10001010	(résultat)

L'instruction XORLW a donc bien inversé l'état logique des bits 7, 6, 5 et 4 de l'octet se trouvant dans W. Les bits qui étaient à 0 sont passés à 1, et inversement.

XORWF

Exclusive OR W and File

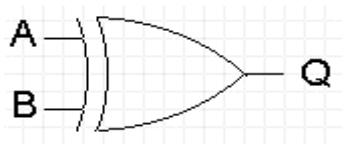
- Effectue un OU exclusif (Exclusive OR) entre l'octet se trouvant dans le registre W et l'octet situé (en mémoire RAM) à l'adresse indiquée de suite (adresse comprise entre 0C et 4F) ; avec deux variantes : le résultat peut être rangé soit dans le registre W, soit dans la mémoire RAM à la place de l'octet utilisé pour effectuer le OU exclusif (la nouvelle valeur prend la place de l'ancienne).

- Un OR exclusif permet de comparer deux octets bit à bit.
Si les bits de même poids sont au même niveau, le résultat est zéro.
Si par contre ils sont à des niveaux différents, le résultat est 1.
Exemple de XOR entre deux octets :

00110011
01110010
résultat = 01000001

- Un OR exclusif permet d'inverser un état logique.
Exemple :
Soit au départ l'octet 11111111.
Si le XOR se fait avec 00000000 , rien ne change ;
le résultat est : 11111111.
Si par contre le XOR se fait avec 00000001, le résultat est :11111110. (Alors que les zéros ne font rien changer, les 1 provoquent un basculement d'état).

- Table de vérité d'une porte OU exclusif :

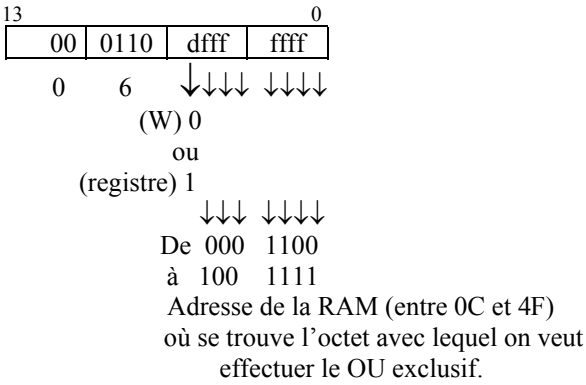


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

- Cette instruction affecte le bit Z du registre d'état

- 1 cycle d'horloge

- Encodage de l'instruction:



Pourquoi cette instruction dans le set du 16F84? A quoi sert-elle pratiquement?

Elle sert essentiellement lorsque – dans un octet – on a besoin d*'inverser* un bit en particulier (ou certains bits en particulier).

Pour cela il suffit de préparer un *masque*, c'est à dire on octet composé de 0 (aux emplacements où se trouvent les bits à ignorer) et de 1 (aux emplacements où se trouvent les bits qu'on veut *inverser*).

Exemple : on souhaite *inverser* les bits 7, 6, 5 et 4 de l'octet 01111010.

On prépare alors le *masque* 11110000 et on fait un OU logique entre l'octet et le *masque*. Comme ceci :

	01111010	(octet)
	11110000	(*masque*)
Ce qui donne :	10001010	(résultat)

L'instruction XORWF a donc bien inversé l'état logique des bits 7, 6, 5 et 4 de l'octet en mémoire. Les bits qui étaient à 0 sont passés à 1, et inversement.

ENCODAGE des INSTRUCTIONS

Signification des lettres utilisées dans l'encodage des instructions :

b = position du bit dans l'octet sur lequel on opère (il peut aller de 0 à 7)

d = registre de destination, avec deux variantes :
d = 0 : le résultat est placé dans W
d = 1 : le résultat est placé dans le registre

f = généralement adresse de la mémoire RAM où se trouve l'octet concerné (dans l'espace mémoire compris entre 00 et 4F).
Dans l'instruction TRIS : octet de configuration de port

k = valeur du *literal* (ou adresse, dans les instructions CALL et GOTO)

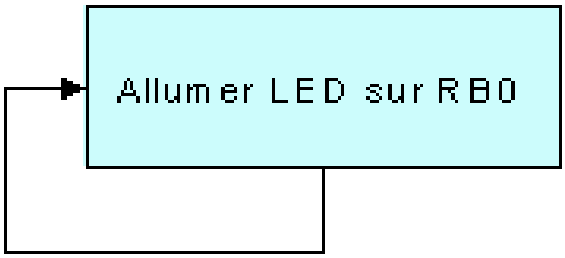
w = registre W (accumulateur)

x = valeur indifférente : peut être soit 0 soit 1.
L'assembleur lui donne automatiquement la valeur 0
(forme recommandée par Microchip)

Exemples de programmes écrits en langage ASSEMBLEUR

Programme 1

a) Organigramme



b) Fichier à extension .asm

```
;Titre du programme : PROG1
;Ce programme allume la LED branchée sur la
;sortie RB0 (bit 0 du Port B) et la laisse
;indéfiniment allumée.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                PROCESSOR       16F84
                RADIX           HEX
                INCLUDE         « P16F84.INC »
                __CONFIG        3FF1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                       VECTEUR de RESET
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ORG     00      ;Vecteur de Reset.
        GOTO    START   ;Renvoi à l'adresse EEPROM 05 (hexa)

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                       INITIALISATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START   ORG     05      ;Saut introduit pour passer au-dessus
                        ;des 5 premières adresses de la mémoire
                        ;EEPROM (00 – 01 – 02 – 03 et 04).

        CLRF    PORTB   ;Initialise le Port B.

        BSF     STATUS,RP0      ;Met à 1 (set) le bit 5 (RP0) du
                                ;registre d'état (STATUS).
                                ;Autrement dit : sélectionne la
                                ;page 1 du Register File (adresses
                                ;de 80 à 8B) dans laquelle se trouve
                                ;le Registre STATUS (à l'adresse 83).

        MOVLW   b'00000000'     ;Met la valeur binaire 00000000 dans
                                ;le registre W, matérialisant ainsi notre
                                ;intention d'utiliser les 8 lignes du
                                ;Port B comme SORTIES.

        MOVWF   TRISB           ;Port B configuré, mais encore en
                                ;haute impédance (Trhee-state).

        BCF     STATUS,RP0      ;Retour à la page 0 du Register File.
```

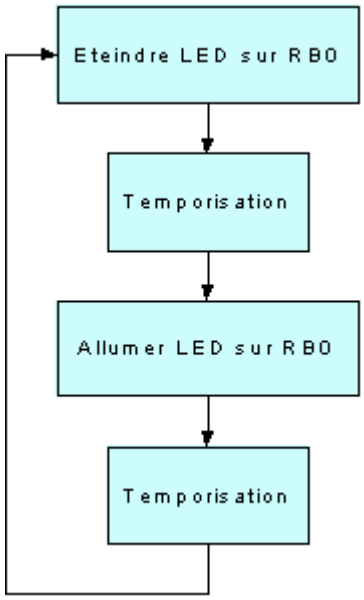
```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                                PROGRAMME
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LOOP    BSF     PORTB,0         ;Allume la LED, car l'instruction
                                ; « BSF » met à 1 (set).
                                ;Dans le cas présent, elle met à 1 le
                                ;bit zéro du Port B (PORTB,0).

        GOTO    LOOP            ;Le programme se reboucle.
                                ;La LED reste indéfiniment allumée.

        END                     ;Fin du programme.
```

c) Fichier à extension .hex

```
020000000528D1
06000A00860183160030A0
0800100086008312061 40A2881
02400E00F13F80
00000001FF
```

Programme 2

a) Organigramme



b) Fichier à extension .asm

```
;Titre du programme : PROG2
;Ce programme fait clignoter indéfiniment la LED branchée sur la
;sortie RB0 (bit 0 du Port B).
;Le programme comporte une temporisation (DELAI) pour rendre
;perceptibles les allumages et les extinctions de la LED, sinon les
;transitions auraient lieu à très grande vitesse et notre œil ne verrait
;pas les clignotements.

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    PROCESSOR       16F84
                    RADIX           HEX
                    INCLUDE         « P16F84.INC »
                    __CONFIG        3FF1

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DECLARATIONS DES VARIABLES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
COMPT1  EQU     0C      ;On met la variable COMPT1 à
                        ;l'adresse RAM 0C.
COMPT2  EQU     0D      ;On met la variable COMPT2 à
                        ;l'adresse RAM 0D.

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          VECTEUR DE RESET
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ORG     00      ;Vecteur de Reset.
        GOTO    START   ;Renvoi à l'adresse EEPROM 05 (hexa)

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          INITIALISATIONS
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START   ORG     05      ;Saut introduit intentionnellement pour faire
                        ;démarrer le programme à l'adresse EEPROM 05.

        CLRF    PORTB   ;Efface les 8 bits du Port B.

        BSF     STATUS,RP0      ;Met à 1 (set) le bit 5 (RP0) du
                                ;registre d'état (STATUS).
                                ;Autrement dit : sélectionne la
                                ;page 1 du Register File (adresses
                                ;de 80 à 8B) dans laquelle se trouve
                                ;le Registre STATUS (à l'adresse 83).

        MOVLW   b'00000000'     ;Met la valeur binaire 00000000 dans
                                ;le registre W, matérialisant ainsi notre
                                ;intention d'utiliser les 8 lignes du
```

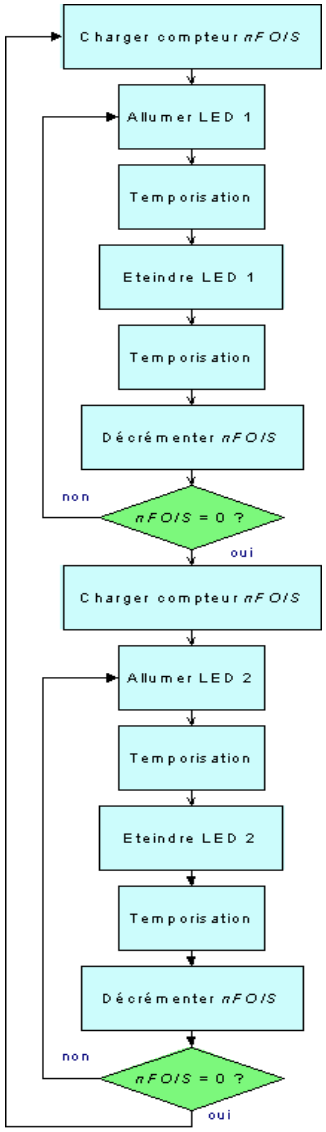
```
                                    ;Port B comme SORTIES.
                                    ;La notation b'00000000' indique que
                                    ;la valeur 00000000 est à interpréter
                                    ;en tant que chiffre binaire.

          MOVWF  TRISB              ;Port B configuré, mais encore en
                                    ;haute impédance (Trhee-state).

          BCF    STATUS,RP0         ;Retour à la page 0 du Register File.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                         PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MAIN      BCF    PORTB,0            ;LED éteinte car l'instruction
                                    ; « BCF » met à 0 (clear).
                                    ;Ici, elle met à 0 le bit 0 du
                                    ;Port B (PORTB,0).

          CALL   DELAI              ;Appelle le sous-programme de
                                    ;retard (DELAI).

          BSF    PORTB,0            ;LED allumée, car l'instruction
                                    ; « BSF » met à 1 (set).
                                    ;Ici elle met à 1 le bit 0 du
                                    ;Port B (PORTB,0).

          CALL   DELAI              ;On appelle à nouveau le
                                    ;sous-programme de retard.
          GOTO   MAIN               ;Retour au programme principal.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                   SOUS-PROGRAMME de TEMPORISATION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DELAI     DECFSZ COMPT1,1           ;Décrémente COMPT1et - s'il n'est pas
          GOTO   DELAI              ;à zéro - va à DELAI

          MOVLW  .255               ;Charge la variable COMPT1 (adresse
          MOVWF  COMPT1             ;RAM 0C) avec 255 (en décimal).

          DECFSZ COMPT2,1           ;Décrémente COMPT2, et s'il n'est pas
          GOTO   DELAI              ;à zéro, va à DELAI

          MOVLW  .255               ;Recharge COMPT1 avec 255
          MOVWF  COMPT1

          MOVLW  .255               ;Recharge COMPT2 avec 255
          MOVWF  COMPT2

          RETURN                    ;Fin du sous-programme DELAI

          END                       ;Fin du programme.
```

c) Fichier à extension .hex

```
020000000528D1
06000A00860183160030A0
1000100860083120610 0F2006140F200A288C0B6E
100020000F28FF308C008D0B0F28FF308C00FF3025
040030008D00080037
02400E00F13F80
00000001FF
```

PROGRAMME 3

a) Organigramme



b) Fichier à extension .asm

```
;Titre du programme : PROG3
;Ce programme fait clignoter un certain nombre de fois la LED
;branchée sur la sortie RB0 (bit 0 du PORT B), puis fait clignoter
;un certain autre nombre de fois la LED branchée sur la sortie
;RB1 (bit 1 du Port B), et recommence le cycle indéfiniment.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    PROCESSOR       16F84
                    RADIX           HEX
                    INCLUDE         « P16F84.INC »
                    __CONFIG        3FF1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DECLARATIONS DES VARIABLES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
COMPT1  EQU     0C      ;On met la variable COMPT1 à
                        ; l'adresse RAM 0C.
COMPT2  EQU     0D      ; On met la variable COMPT2 à
                        ; l'adresse RAM 0D.
nFOIS   EQU     0E      ; On met la variable nFOIS à
                        ; l'adresse RAM 0E.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          VECTEUR DE RESET
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ORG     00      ;Vecteur de Reset.
        GOTO    START   ;Renvoi à l'adresse EEPROM 05 (hexa)

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          INITIALISATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START   ORG     05      ;Saut introduit intentionnellement pour faire
                        ;démarrer le programme à l'adresse EEPROM 05.

        CLRF    PORTB   ;Efface les 8 bits du Port B.

        BSF     STATUS,RP0      ;Met à 1 (set) le bit 5 (RP0) du
                                ;registre d'état (STATUS).
                                ;Autrement dit : sélectionne la
                                ;page 1 du Register File (adresses
                                ;de 80 à 8B) dans laquelle se trouve
                                ;le Registre STATUS (à l'adresse 83).

        MOVLW   b'00000000'     ;Met la valeur binaire 00000000 dans
                                ;le registre W, matérialisant ainsi notre
                                ;intention d'utiliser les 8 lignes du
```

```
                                        ;Port B comme SORTIES.
                                        ;La notation b'00000000' indique que
                                        ;la valeur 00000000 est à interpréter
                                        ;en tant que chiffre binaire.

          MOVWF  TRISB                  ;Port B configuré, mais encore en
                                        ;haute impédance (Trhee-state).

          BCF    STATUS,RP0             ;Retour à la page 0 du Register File.

          GOTO   MAIN1                  ;Renvoi à l'adresse correspondant à
                                        ;la label MAIN1.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                  SOUS-PROGRAMME de TEMPORISATION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
TEMPO     MOVLW  .255                 ;Charge COMPT2  (« grande boucle »)
          MOVWF  COMPT2               ;avec la valeur décimale 255.

DELAI2    MOVLW  .255                 ;Charge COMPT1  (« petite boucle »)
          MOVWF  COMPT1               ;avec la valeur décimale 255.

DELAI1    DECFSZ COMPT1,1             ;Décrémente COMPT1et - s'il n'est pas
          GOTO   DELAI1               ;à zéro - va à DELAI1

          DECFSZ COMPT2,1             ;Décrémente COMPT2, et s'il n'est pas
          GOTO   DELAI2               ;à zéro, va à DELAI2.

          RETURN                      ;Fin du sous-programme TEMPO.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                        PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MAIN1     MOVLW  .2                   ;On définit le nombre de cycles de la
          MOVWF  nFOIS                ;première phase (ici : 2 clignotements)

LED1      BCF    PORTB,0              ;LED éteinte car l'instruction
                                      ; « BCF » met à 0 (clear).
                                      ;Ici, elle met à 0 le bit 0 du
                                      ;Port B (PORTB,0).

          CALL   TEMPO                ;Appelle le sous-programme de
                                      ;retard (TEMPO).

          BSF    PORTB,0              ;LED allumée, car l'instruction
                                      ; « BSF » met à 1 (set).
                                      ;Ici elle met à 1 le bit 0 du
                                      ;Port B (PORTB,0).

          CALL   TEMPO                ;On appelle à nouveau le
                                      ;sous-programme de retard.
```

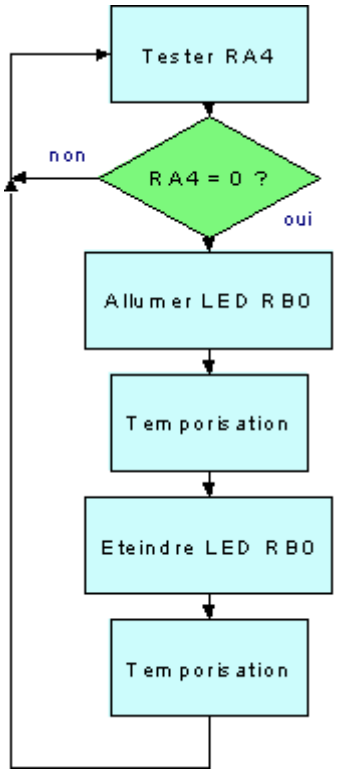
```
        DECFSZ  nFOIS,1     ;Décrémente le nombre de cycles
        GOTO    LED1        ;affectés à LED1, et si le compteur
        GOTO    MAIN2       ;n'est pas arrivé à 0, effectue un
                            ;nouveau cycle. Si par contre nFOIS
                            ;est à 0, l'instruction « GOTO LED1 »
                            ;est ignorée et le programme saute à
                            ; « GOTO MAIN2 ».

MAIN2   MOVLW   .5          ;On définit le nombre de cycles de la
        MOVWF   nFOIS       ;deuxième phase (ici :5 clignotements)

LED2    BCF     PORTB,1     ;LED éteinte car l'instruction
                            ; « BCF » met à 0 (clear).
                            ;Ici, elle met à 0 le bit 1 du
                            ;Port B (PORTB,1).

        CALL    TEMPO       ;Appelle le sous-programme de
                            ;retard (TEMPO).

        BSF     PORTB,1     ;LED allumée, car l'instruction
                            ; « BSF » met à 1 (set).
                            ;Ici elle met à 1 le bit 1 du
                            ;Port B (PORTB,1).

        CALL    TEMPO       ;On appelle à nouveau le
                            ;sous-programme de retard.

        DECFSZ  nFOIS,1     ;Décrémente le nombre de cycles
        GOTO    LED2        ;affectés à LED2, et si le compteur
        GOTO    MAIN1       ;n'est pas arrivé à 0, effectue un
                            ;nouveau cycle. Si par contre nFOIS
                            ;est à 0, l'instruction « GOTO LED2 »
                            ;est ignorée et le programme saute à
                            ; « GOTO MAIN1 ».

        END                 ;Fin du programme.
```

c) Fichier à extension .hex

```
020000000528D1
06000A00860183160030A0
1000100860083121428FF308D00FF308C008C0B7B
100020000F288D0B0D28080001308E0006140B20C0
1000300006100B208E0B16281D2802308E00861409
0C0040000B2086100B208E0B1F281428AC
00000001FF
```

PROGRAMME 4

a) Organigramme



b) Fichier à extension .asm

```
;Titre du programme : PROG4
;Ce programme fait clignoter une LED
;branchée sur la sortie RB0 (bit 0 du PORT B), si l'entrée RA4
;(bit 4 du Port A) est activée.
;En désactivant RA4, la LED s'éteint.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    PROCESSOR       16F84
                    RADIX           HEX
                    INCLUDE         « P16F84.INC »
                    __CONFIG        3FF1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DECLARATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
STATUS  EQU     03      ;
RP0     EQU     05      ;       Aurions pu ne pas les
PORTA   EQU     05      ;       déclarer, car ces
PORTB   EQU     06      ;       équivalences sont
TRISA   EQU     85      ;       définies dans le fichier
TRISB   EQU     86      ;       INCLUDE « P16F84.INC ».
START   EQU     05      ;       C'est redondant. Mais ce n'est
                                pas interdit.
COMPT1  EQU     0C      ;On met la variable COMPT1 à
                        ; l'adresse RAM 0C.
COMPT2  EQU     0D       .On met la variable COMPT2 à
                        ; l'adresse RAM 0D.
nFOIS   EQU     0E      ;On met la variable nFOIS
                        ;à l'adresse RAM 0E.

        ORG     00      ;Vecteur de Reset.

        GOTO    START   ;Renvoi à l'adresse EEPROM 05 (hexa)

        ORG     05      ;Voulons intentionnellement faire
                        ;démarrer le programme à l'adresse EEPROM 05.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          INITIALISATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        CLRF    PORTB           ;Efface les 8 bits du Port B.

        BSF     STATUS,RP0      ;Met à 1 (set) le bit 5 (RP0) du
                                ;registre d'état (STATUS).
                                ;Autrement dit : sélectionne la
                                ;page 1 du Register File pour atteindre
                                ;le Registre TRISB (à l'adresse 86).
```

```
            MOVLW  00               ;Met des zéros dans le registre W,
                                    ;pour les porter ensuite dans le
                                    ;Registre TRISB
            MOVWF  TRISB            ;matérialisant ainsi notre intention
                                    ;d'utiliser les 8 lignes du Port B
                                    ;comme sorties.
            MOVLW  0xFF             ;Met 11111111 dans le Registre W,
                                    ;pour les porter ensuite dans le
                                    ;Registre TRISA
            MOVWF  TRISA            ;matérialisant ainsi notre intention
                                    ;d'utiliser les 8 lignes du Port A
                                    ;comme entrées.

            BCF    STATUS,RP0       ;Retour à la page 0 du Register File.

            GOTO   MAIN             ;Renvoi au programme principal.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          SOUS-PROGRAMMES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TEMPO                               ;Début du sous-programme  TEMPO

            MOVLW  .255             ;Charge COMPT2 (« grande boucle »)
            MOVWF  COMPT2           ;avec la valeur décimale 255

DELAI2      MOVLW  .255             ;Charge COMPT1 («petite boucle »)
            MOVWF  COMPT1           ;avec la valeur décimale 255

DELAI1      DECFSZ COMPT1,1         ;Décrémente COMPT1, et s'il n'est pas
            GOTO   DELAI1           ;arrivé à zéro, va à DELAI1. Ces deux
                                    ;instructions permettent de vider la
                                    ;variable COMPT1 (qui est une zone
                                    ;RAM et qui pourrait, au démarrage du
                                    ;système, contenir une valeur
                                    ;aléatoire), pour ensuite lui donner
                                    ;une valeur précise.

            DECFSZ COMPT2,1         ;Décrémente COMPT2, et s'il n'est
                                    ;pas à zéro, va à DELAI2, « grande
                                    ;boucle » engendrant un retard long
            GOTO   DELAI2           ;COMPT2 fois la valeur de COMPT1.

            RETURN                  ;Fin du sous-programme TEMPO
                                    ;et retour au programme principal
                                    ;à la ligne située juste après
                                    ;l'instruction « CALL TEMPO ».

LED                                 ;Début du sous-programme « LED »

            BSF    PORTB,0          ;Allume la LED.
```

```
            CALL    TEMPO           ;Appel du sous-programme de
                                    ;temporisation.

            BCF     PORTB,0         ;Eteint la LED.

            CALL    TEMPO           ;Nouvel appel du sous-
                                    ;programme de temporisation.

            RETURN                  ;Fin du sous-programme « LED ».

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                           PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MAIN        BTFSS   PORTA,4         ;Est-ce que l'entrée RA4 est activée ?

            GOTO    MAIN            ;NON. Alors on continue à tester.

            CALL    LED             ;OUI. Dans ce cas on appelle le
                                    ;sous-programme LED.

            GOTO    MAIN            ;Retour au programme principal.

            END
```

c) Fichier à extension .hex

```
020000000528D1
06000A00860183160030A0
100010008600FF30850083121B28FF308D00FF30E3
100020008C008C0B11288D0B0F28080006140D2056
0E00300006100D200800051E1B2816201B2898
00000001FF
```

PROGRAMME 5

Fichier à extension .asm

```
;Titre du programme : PROG5
;Ce programme fait clignoter une LED
;en utilisant les interruptions générées
;par le timer interne (TMR0).

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                PROCESSOR       16F84
                INCLUDE         « P16F84.INC »
                __CONFIG        3FF1

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DECLARATIONS
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

SAVE_W          EQU     0C      ;Déclaration de
SAVE_STAT       EQU     0D      ;deux variables.

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          VECTEURS
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ORG     00              ;Vecteur de Reset.
        GOTO    START

        ORG     04              ;Vecteur d'interruption.
        GOTO    INT_VECT

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          START
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START   ORG     05

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          INITIALISATIONS
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        BSF     STATUS,RP0      ;On passe en Page 1.

        MOVLW   b'00000000'     ;Port B en sortie.
        MOVWF   TRISB
```

```
        MOVLW  b'00000111'      ;On configure OPTION.
        MOVWF  OPTION_REG       .Le pré diviseur divise par 255.

        BCF    STATUS,RP0       ;On revient en Page 0.

        CLRF   TMR0             ;Timer à zéro.

        CLRF   PORTB            ;Toutes LED éteintes.

        MOVLW  b'10100000'      ;On configure INTCON.
        MOVWF  INTCON           ;       - GIE   (bit 7) à 1
                                ;       - T0IE (bit 5) à 1
                                ;       - tous les autres bits à zéro.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                       PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LOOP    GOTO    LOOP            ;Boucle introduite juste pour occuper
                                ;le processeur, car le but du programme
                                ;est d'attendre l'apparition du signal
                                ;d'interruption.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                       PROGRAMME d'INTERRUPTION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INT_VECT        MOVWF   SAVE_W          ;Phase de PUSH (store).

                MOVF    STATUS,W        ;On sauve le Registre W
                MOVWF   SAVE_STAT       ;ainsi que le Registre
                                        ;STATUS.
                BCF     INTCON,T0IF     ;On met à zéro le flag
                                        ;T0IF qui passe à 1 à chaque
                                        ;débordement du TMR0
                                        ;(bit 2).
                BTFSC   PORTB,0
                GOTO    LED_OFF
                BSF     PORTB,0         ;On allume la LED RB0.
                GOTO    FIN

LED_OFF         BCF     PORTB,0         ;On éteint la LED RB0.

FIN             MOVF    SAVE_STAT,W     ;Phase de POP (restore).
                MOVWF   STATUS          ;On remet en place le
                MOVF    SAVE_W,W        :Registre STATUS, ainsi
                                        ;que le Registre W.

                RETFIE                  ;Lorsqu'une interruption est générée,
                                        ;le PIC met automatiquement à zéro le
                                        ;bit GIE du Registre INTCON pendant
                                        ;toute la durée d'exécution du
                                        ;sous-programme d'interruption (pour
```

```
                                        ;empêcher la prise en compte d'une
                                        ;nouvelle interruption pouvant surgir,
                                        ;alors qu'il est justement en train d'en
                                        ;traiter une. (On dit que pendant ce
                                        ;temps-là les interruptions sont
                                        ;masquées).
                                        :L'instruction RETFIE, en même temps
                                        ;qu'elle provoque le retour au
                                        ;programme principal, remet à 1 le bit
                                        ;GIE (Global Interrupt Enable).

                END
```

PROGRAMME 6

Fichier à extension .asm

```
;Titre du programme : PROG6
;Ce programme fait clignoter une LED
;en utilisant les interruptions générées
;par le timer interne (TMR0), et - en plus - allume et laisse
;toujours allumée une deuxième LED branchée sur RB1.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    PROCESSOR       16F84
                    INCLUDE         « P16F84.INC »
                    __CONFIG        3FF1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DECLARATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

SAVE_W          EQU     0C
SAVE_STAT       EQU     0D      ;Variables
CHOIX           EQU     0E
COMPT           EQU     0F

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          VECTEURS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ORG     00              ;Vecteur de Reset.
        GOTO    START

        ORG     04              ;Vecteur d'interruption.
        GOTO    INT_VECT

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          START
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START   ORG     05

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          INITIALISATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        BSF     STATUS,RP0      ;On passe en Page 1.
```

```
            MOVLW  b'00000000'      ;Port B en sortie.
            MOVWF  TRISB

            MOVLW  b'00000111'      ;On configure OPTION.
            MOVWF  OPTION_REG       .Le pré diviseur divise par 255.

            BCF    STATUS,RP0       ;On revient en Page 0.

            CLRF   TMR0             ;Timer à zéro.

            CLRF   PORTB            ;Toutes LED éteintes.

            MOVLW  b'10100000'      ;On configure INTCON.
            MOVWF  INTCON           ;          - GIE   (bit 7) à 1
                                    ;          - T0IE (bit 5) à 1
                                    ;          - tous les autres bits à zéro.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MAIN        BSF    PORTB,1          ;On allume la LED (BR1) et on la
            GOTO   MAIN             ;laisse toujours allumée, en attendant
                                    ;l'apparition d'une interruption.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          PROGRAMME d'INTERRUPTION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INT_VECT                MOVWF  SAVE_W           ;Phase de PUSH (store).
                        MOVF   STATUS,W         ;On sauve W
                        MOVWF  SAVE_STAT        ;et STATUS dans la RAM.

                        BCF    INTCON,T0IF      ;On met à zéro le flag
                                                ;T0IF qui passe à 1 à chaque
                                                ;débordement du TMR0
                                                ;(bit 2).
                        BTFSC  PORTB,0
                        GOTO   SUITE
                        BSF    PORTB,0          ;On allume la LED RB0.
                        GOTO   FIN

SUITE                   BCF    PORTB,0          ;On éteint la LED RB0.

FIN                     MOVF   SAVE_STAT,W      ;Phase de POP (restore).
                        MOVWF  STATUS           ;On remet en place
                        MOVF   SAVE_W,W         :STATUS et W.

                        RETFIE          ;Lorsqu'une interruption est générée,
                                        ;le PIC met automatiquement à zéro le
                                        ;bit GIE du Registre INTCON pendant
                                        ;toute la durée d'exécution du
                                        ;sous-programme d'interruption (pour
```

```
                                        ;empêcher la prise en compte d'une
                                        ;nouvelle interruption pouvant surgir,
                                        ;alors qu'il est justement en train d'en
                                        ;traiter une. (On dit que pendant ce
                                        ;temps-là les interruptions sont
                                        ;masquées).
                                        :L'instruction RETFIE, en même temps
                                        ;qu'elle provoque le retour au
                                        ;programme principal, remet à 1 le bit
                                        ;GIE (Global Interrupt Enable).

                END
```

PROGRAMME 7

Fichier à extension .asm

```
;Titre du programme : PROG7
;Ce programme active un chenillard sur 4 LED
;utilisant les bits 0, 1, 2, et 3 du Port B.
;Chaque LED s'allume pendant un temps calibré de juste 1 minute
;(60 secondes) fourni par le TMR0 en association avec un compteur.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          DIRECTIVES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    PROCESSOR       16F84
                    INCLUDE         « P16F84.INC »
                    __CONFIG        3FF1

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                    DECLARATIONS des VARIABLES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SHIFT               EQU     0C
COMPT1              EQU     0D

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                          VECTEURS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          ORG       00              ;Vecteur de Reset.
          GOTO      START           ;Renvoi à l'adresse EEPROM 05

          ORG       04              ;Vecteur d'interruption.
          GOTO      INT_VECT        ;Renvoi au programme d'interruption.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                        INITIALISATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
START     ORG       05              ;Saut introduit intentionnellement
                                    ;pour passer au-dessus des adresses
                                    ;EEPROM 01 – 02 – 03 – et  04,
                                    ;et obliger le programme à démarrer
                                    ;à l'adresse 05.

          BSF       STATUS,RP0      ;On passe en Page 1 pour atteindre
                                    ;TRISB (adresse 86) et OPTION
                                    ;(adresse 81).

          MOVLW     b'00000000'     ;Port B configuré en sortie (0=sortie).
          MOVWF     TRISB
          MOVLW     b'00000100'     ;Pré diviseur affecté au TMR0, et
```

```
                MOVWF   OPTION_REG      ;configurer pour diviser par 32.

                BCF     STATUS,RP0      ;On revient en Page 0.

                MOVLW   .125
                MOVWF   COMPT1

                MOVLW   b'10100000'     ;On configure le Registre INTCON :
                MOVWF   INTCON          ;       - GIE   (bit 7) à 1
                                        ;       - T0IE (bit 5) à 1
                                        ;       - tous les autres bits à zéro

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                               PROGRAMME  PRINCIPAL
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LOOP1   GOTO    LOOP1           ;Boucle introduite pour occuper le
                                ;processeur, car le but principal du
                                ;programme est d'attendre l'apparition
                                ;du signal d'interruption pour
                                ;déclencher l'animation du chenillard.

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                               PROGRAMME d'INTERRUPTION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INT_VECT        MOVLW   b'00000001'     ;On allume la LED
                MOVWF   SHIFT           ;branchée sur RB0,
                MOVF    SHIFT,0         ;en transitant par
                MOVWF   PORTB           ;la variable SHIFT.

                BCF     STATUS,0        ;Pour créer un effet de
                RLF     SHIFT,1         ;chenillard, on utilise
                                        ;l'instruction RLF (Rotate
                                        ;Left File) laquelle :
                                ; 1) provoque un glissement (rotation)
                                ;    à gauche (Left), et
                                ; 2) insère le contenu du bit de CARRY
                                ;    dans le bit 0 du Registre STATUS.
                                ;    Comme on ignore la valeur (0 ou 1)
                                ;    de la CARRY, il est impératif de
                                ;    l'effacer au préalable.
                BTFSC   SHIFT,4 ;Lorsque – de rotation en
rotation –
                                ;le 1 baladeur atteint le bit 4, on
                                ;provoque un SWAP (croisement) entre
                                ;le demi octet de poids faible et le demi
                                ;octet de poids fort (on inverse les
                                ;demi octets)
                SWAPF   SHIFT,1 ;Après chaque rotation, l'octet est
                                ;gardé dans le même Registre SHIFT,
                                ;pour y évoluer.

                CALL    DELAI   ;On appelle le sous programme DELAI
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                    SOUS-PROGRAMME de TEMPORISATION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

DELAI   MOVLW  .6              ;On initialise le TMR0 avec la valeur
        MOVWF  TMR0            ;6 (décimal), afin qu'il déborde
                               ;après 250 impulsions.

TEST    MOVF   TMR0,0          ;On porte la valeur du TMR0 dans
        BTFSS  STATUS,Z        ;W pour pouvoir la tester :
                               ;a-t-elle vu arriver (256 – 6)
                               ;250 impulsions ?
                               ;C'est à dire : est-ce que le TMR0
                               ;est arrivé à FF ?
        GOTO   TEST            ;Si NON : on continue à tester.

        MOVLW  .6              ;Si OUI : on recharge TMR0
        MOVWF  TMR0            ;avec 6 (décimal) et ensuite
        DECFSZ COMPT1,1        ;on décrémente la variable COMPT1
        GOTO   TEST            ;Est-ce que COMPT1 = 0 ?
                               ;NON : on continue à décrémenter.
                               ;OUI :  1 seconde s'est écoulée.
        MOVLW  .6              ;On recharge
        MOVWF  TMR0            ;le TMR0.

        MOVLW  .125            ;On recharge
        MOVWF  COMPT1          ;la variable COMPT1.

        RETURN                 ;Fin du sous-programme DELAI.

        END                    ;Fin du programme.
```