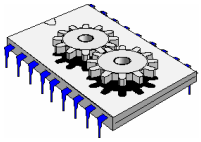


A la découverte des microcontrôleurs PIC
Première partie

Cette série de cours concerne les microcontrôleurs PIC et plus particulièrement la famille 16F84. Vous apprendrez tout au long de ces cours à programmer et à utiliser ce composant très performant.

- Introduction - *Du microprocesseur au microcontrôleur*

Déjà trente ans ...

La découverte du microprocesseur date aujourd'hui de près de trente ans, en effet la fabrication du premier circuit commence en 1970 , année ou la société INTEL® met au point le premier microprocesseur le 4004. On n'imagine pas à l'époque que cette révolution industrielle donnera naissance à l'ordinateur individuel. Depuis, leur puissance de calcul et l'intégration des transistors les constituant n'ont cessé d'évoluer. On retrouve désormais les microprocesseurs dans la plupart des applications, que ce soit, pour piloter une centrale, à l'intérieur d'un ordinateur ou bien encore pour remplacer le programmateur d'une machine à laver.

Les microprocesseurs ne sont jamais employés seuls, des circuits périphériques leur sont toujours associés pour pouvoir être intégrés au sein d'une application (**figure 1**).
Un des avantages d'un montage à base de microprocesseur par rapport à un montage en logique câblée, est sa souplesse d'emploi, en effet nous entrons dans le domaine de la logique programmable ou le fonctionnement du montage, dépend d'un programme logé dans une mémoire, celui-ci peut être modifié pour changer les équations régissant l'application, sans toutefois entraîner de changement au niveau du câblage des entrées sorties.

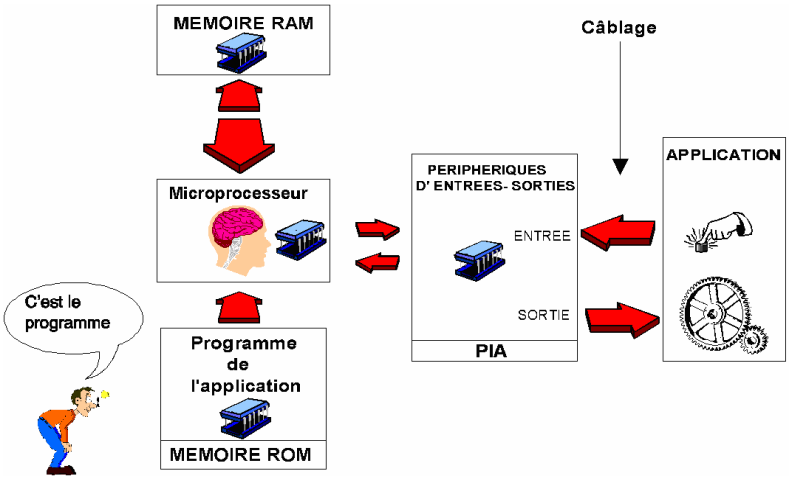


Figure 1

- **Les microcontrôleurs**

Les microprocesseurs nous venons de le voir possèdent un indéniable avantage sur la logique câblée, en effet pour modifier le fonctionnement d'une application il suffit de modifier le programme sans refaire de câblage.
Les microcontrôleurs possèdent quant à eux la puissance d'un microprocesseur mais ont un atout en plus, en effet ils possèdent dans le même boîtier, les périphériques intégrés (**figure 2**).
Cela veut dire que le programme de l'application est en interne et non plus dans un circuit mémoire externe et que les périphériques d'entrées - sorties sont également intégrés, ce qui fait l'économie de nombreux circuits périphériques.
Cette caractéristique fait que les montages deviennent encore plus simples et la programmation plus aisée (un système à base de microprocesseur, oblige le concepteur à réaliser un décodage d'adresse pour permettre au microprocesseur de ne dialoguer qu'avec un seul périphérique à la fois).
Un microcontrôleur seul peut donc gérer une application, sans faire appel forcement à d'autres circuits associés.

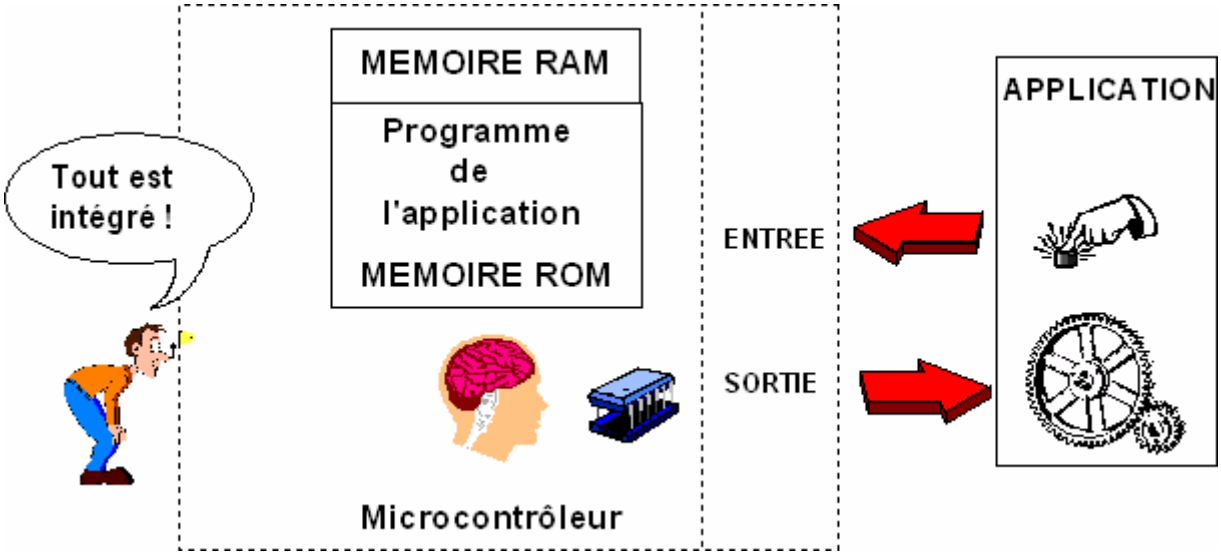


Figure 2

Sur la **figure 2** le microcontrôleur possède en interne la mémoire programme contenant le programme de l'application ainsi que le port d'entrées - sorties qui va permettre au microcontrôleur de s' interfacer avec l'application.
On le voit ici, par rapport au schéma à base de microprocesseur présenté **figure 1**, on a encore franchi un degré d'intégration en rassemblant tous les circuits nécessaires au fonctionnement d'une application dans le même boîtier.

- **Structure interne d'un microcontrôleur (figure 3**)

Un microcontrôleur le plus simple qu'il soit, possède au minimum les éléments suivants :

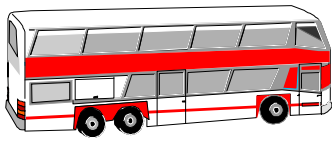
- Une unité centrale qui est le coeur du système, également appelé CPU pour Central Processing Unit, dans cette unité centrale nous retrouverons plusieurs éléments telle que l'unité arithmétique et logique (UAL) que nous détaillerons dans un prochain numéro.

- Une mémoire contenant le programme à exécuter par le microcontrôleur, généralement appelée mémoire morte ou ROM (Read Only Mémory) , mémoire à lecture seule. Cette mémoire a la particularité de sauvegarder en permanence les informations qu'elle contient, même en absence de tension (ce qui est primordiale, sinon il faudrait reprogrammer le microcontrôleur à chaque remise sous tension !).

- Une mémoire vive également appelée RAM (Random Access Mémory), cette mémoire permet de sauvegarder temporairement des informations. Il est à noter que le contenu d'une RAM n'est sauvegardé que pendant la phase d'alimentation du circuit. Le microcontrôleur pourra utiliser cette mémoire pour stocker des variables temporaires ou faire des calculs intermédiaires.

- Un port d'entrées - sorties permettant au microcontrôleur de dialoguer avec l'extérieur pour par exemple prendre l'état d'un capteur , d'un interrupteur ou bien pour allumer une led ou piloter un relais (via un transistor bien sûr).

Des bus internes permettent la communication entres les différents éléments intégrés au microcontrôleur. Il existe trois sortes de bus que l'on détaillera dans un prochain cours.



Cette architecture simplifiée vous est présentée figure 3.

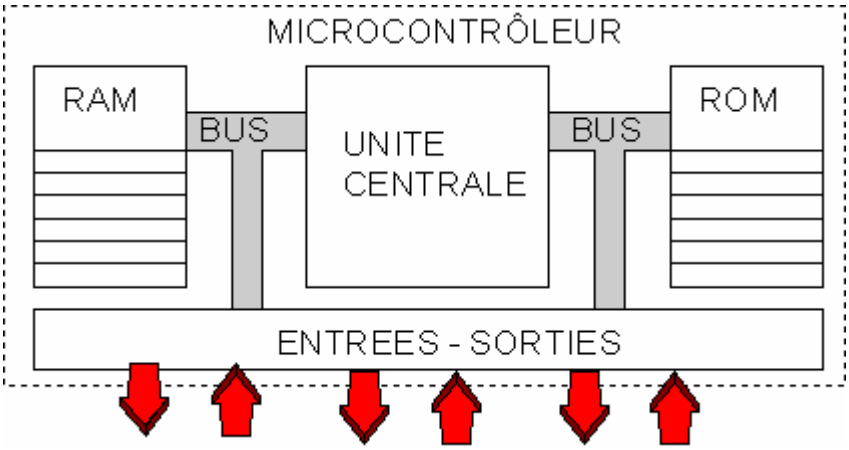


Figure 3

Le microcontrôleur exécutera une à une les instructions codées sous forme binaire dans la mémoire programme (figure 4). En fait le microcontrôleur exécutera les instructions que vous avez transférées dans sa mémoire programme.

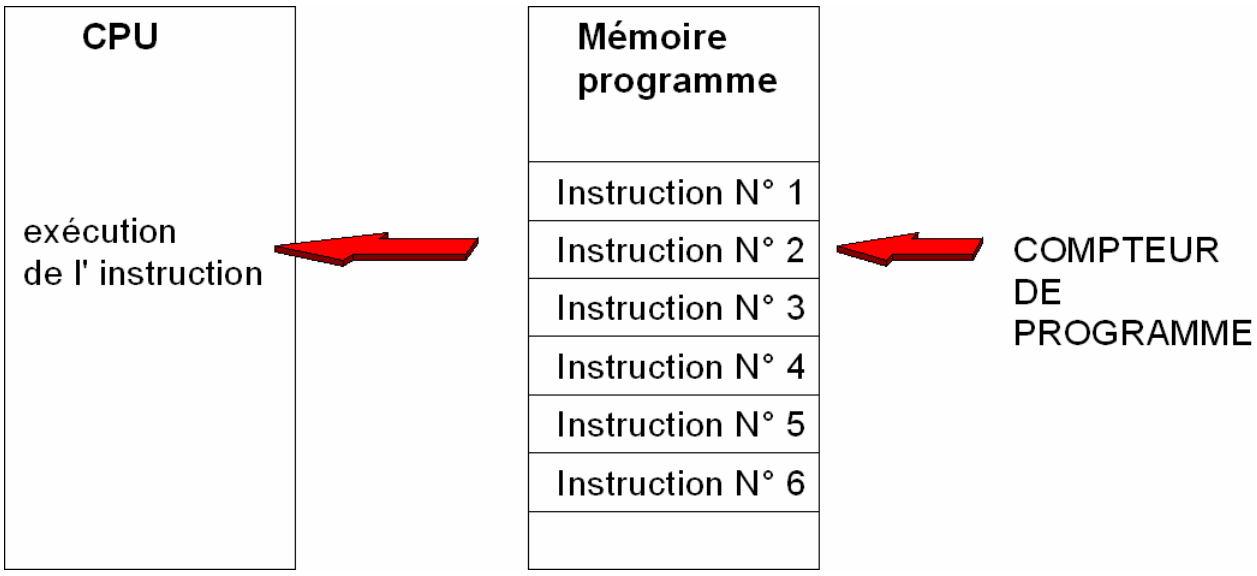
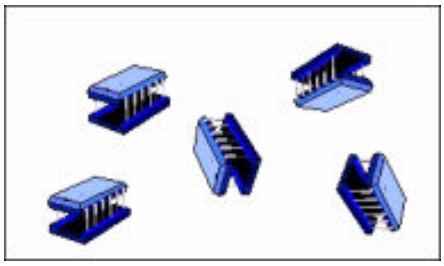


Figure 4

Sans encore entrer trop dans les détails, sachez qu' un registre spécifique du microcontrôleur (le CP - ou compteur de programme) est chargé de pointer l' instruction stockée en ROM (c'est votre programme...) que devra exécuter la CPU du microcontrôleur . En fait le contenu du registre compteur de programme va s' incrémenter au fur et à mesure pour "sélectionner" la case mémoire suivante, ainsi la CPU du microcontrôleur va exécuter toutes les instructions que vous avez transférez dans la mémoire du microcontrôleur.

Avant d'en venir au microcontrôleur PIC, on peut dire en simplifiant qu'un microcontrôleur est un microprocesseur auquel on a intégré divers périphériques, dont en particulier la mémoire contenant le programme à exécuter, ainsi qu'un circuit spécialisé qui permet au microcontrôleur de "dialoguer " avec l'extérieur, que se soit pour " mesurer " ou bien pour " actionner "...
Il existe de nombreux types de microcontrôleurs qu'ils soient spécifiques pour une fonction donnée ou bien banalisés et configurables pour de nombreuses applications.



Quelle famille...

- Les microcontrôleurs PIC

De nombreux fabricants se sont implantés sur le marché des microcontrôleurs.
La société Américaine Microchip® technologie à mis au point un microcontrôleur CMOS (Complémentary Métal Oxyde Semi-conducteur): **Le PIC** .
Ce microcontrôleur encore très utilisé à l'heure actuelle est un compromis entre simplicité d'emploi et prix de revient.
Il fait partie de la famille des circuits RISC (Reduced Instruction Set Computer) , caractérisée par leur vitesse d' exécution et leur jeu d'instruction réduit (le PIC 16F84 possède seulement 35 instructions de base).
Il existe de nombreuses versions de PIC possédant chacune des caractéristiques différentes, des tableaux comparatifs permettent de choisir le PIC le plus adéquat par rapport à l'application envisagée. Un comparatif vous est proposé sur le **tableau 1**, nous reviendrons ultérieurement sur les caractéristiques des PIC présentés.

PIC	Mémoire programme	RAM	Commentaires	Timer	E-S
PIC 12C508	512 x 12	25 x 8		1 8bits	6
PIC 12C509	1024 x 12	41 x 8		1 8bits	6
PIC 12CE518	512 x 12	25 x 8	E²PROM : 16 x 8	1 8bits	6
PIC 12CE519	1024 x 12	41 x 8	E²PROM : 16 x 8	1 8bits	6
PIC 12C671	1024 x 14	128 x 8	4 ADC 8 bits	1 8bits	6
PIC 12C672	2048 x 14	128 x 8	4 ADC 8 bits	1 8bits	6
PIC 12C673	1024 x 14	128 x 8	E²PROM : 16 x 8 -4 ADC 8 bits	1 8bits	6
PIC 12C674	2048 x 14	128 x 8	E²PROM : 16 x 8 -4 ADC 8 bits	1 8bits	6
PIC 16C52	384 x 12	25 x 8		1 8bits	12
PIC 16C54	512 x 12	25 x 8		1 8bits	12
PIC 16C55	512 x 12	24 x 8		1 8bits	21
PIC 16C56	1k x 12	32 x 8		1 8bits	13
PIC 16C57	2k x 12	80 x 8		1 8bits	21
PIC 16C58	2k x 12	80 x 8		1 8bits	21
PIC 16C62A	2048 x 14 (flash)	128 x 8		1 16bits	22
PIC 16C63	4096 x 14 (flash)	192 x 8		1 16bits	22
PIC 16C64	2048 x 14	128 x 8		1 16bits	33
PIC 16C65A	4096x 14 (flash)	192 x 8		1 16bits	33
PIC 16C66	8192 x 14	128 x 8		1 16bits	22
PIC 16C67	8192 x 14	128 x 8		1 16bits	33
PIC 16C71	1024 x 14	36 x 8	4 canaux ADC	1 8bits	13
PIC 16C72	2048x 14	128 x 8	5 canaux ADC	1 16bits	22
PIC 16C73A	4096 x 14	192 x 8	5 canaux ADC	1 16bits	22
PIC 16C74A	4096 x 14	192 x 8	8 canaux ADC	1 16bits	33
PIC 16C76	8192 x 14 (flash)	368 x 8	5 canaux ADC	1 16bits	22
PIC 16C77	8192 x 14	368 x 8	8 canaux ADC	1 16bits	33
PIC 16F83	512 x 14 (flash)	36 x 8	E²PROM : 64 x 8	1 8bits	13
PIC 16C84	1024 x 14	68 x 8	E²PROM : 64 x 8	1 8bits	13
PIC 16F84	1024 x 14 (flash)	68 x 8	E²PROM : 64 x 8	1 8bits	13

Tableau 1

Un minimum de matériel pour développer une application.

Ce premier article vous a plu, vous avez décidé de réaliser une application à base de PIC, que vous faut-il pour commencer ?

1. Dans un premier temps une fois que votre projet est établi (nous y reviendrons ultérieurement avec une application concrète), il va falloir écrire un programme en assembleur (pour débuter). Un programme en assembleur est constitué par une suite d'ordres (Mnémoniques) que devra exécuter le microcontrôleur, cela suppose bien sûr que le programmeur (c'est à dire vous !) se devra de connaître les instructions disponibles pour le PIC choisi et également la façon d'utiliser ces instructions (pas d'affolement, nous verrons bien entendu toutes les instructions du 16F84).
   Pour écrire notre programme en assembleur nous pouvons utiliser n'importe quel éditeur de texte par exemple notepad.exe logiciel fourni avec Windows® , une autre possibilité plus conviviale consistera à utiliser l'éditeur fourni dans l' outil de développement MPLAB disponible gratuitement sur le site internet de Microchip (WWW.MICROCHIP.COM) . Un article sera consacré à cet outil très performant.

2. Nous venons d'écrire notre programme en assembleur (également appelé code source) il faut maintenant compiler ce fichier. La compilation consiste à remplacer les ordres mnémoniques du fichier assembleur par des codes binaires compréhensibles par le microcontrôleur, en effet les microprocesseurs tout comme les microcontrôleurs ne comprennent que les "0" et les "1". Pour effectuer cette compilation il faut disposer d'un compilateur, l'intégré MPLAB cité plus haut permet la compilation.

3. Dernière étape, la compilation à réussie (ouf il n'y a pas d'erreur de syntaxe dans le code source), un fichier binaire comportant l'extension .hex (en général) vient d'être créé par le compilateur. Il reste désormais à transférer ce fichier présent sur le disque dur de votre ordinateur vers la mémoire programme de votre microcontrôleur PIC. De nombreux kits de programmateurs de PIC proposés par les annonceurs de notre revue (pour environ 300f) permettent de transférer le fichier binaire vers la mémoire du PIC, des programmateurs ont également été proposés sur notre revue. Si toutefois vous possédez le programmateur proposé par Microchip, le logiciel MPLAB vous permettra de réaliser le transfert.

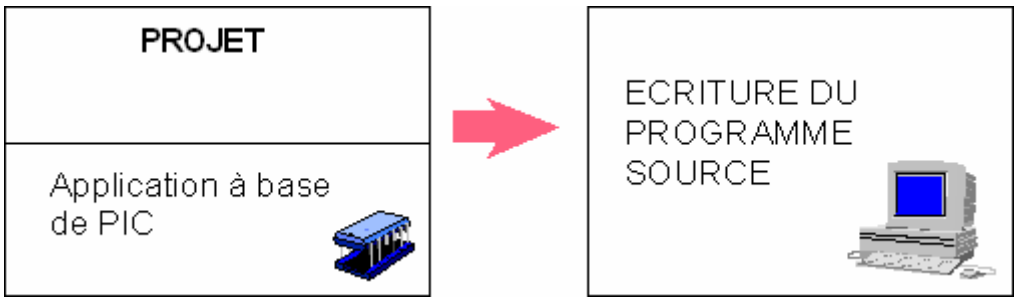


figure 5

Une fois le que le programme en assembleur est réalisé (c'est finalement le plus compliqué), vous pourrez avec MPLAB réaliser une simulation (si vous le souhaitez), si cette simulation correspond au fonctionnement désiré, vous pourrez alors compiler la programme source puis le transférer vers la mémoire du microcontrôleur. Ces quelques étapes devront être exécutées à chaque nouveau programme.

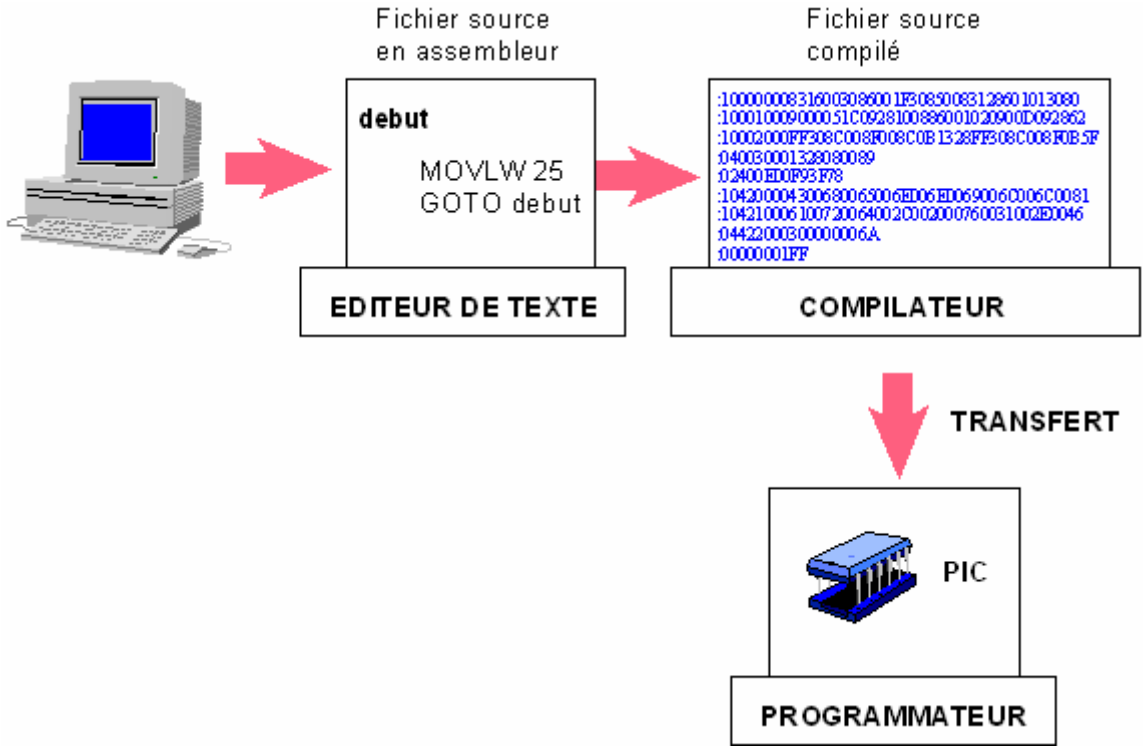
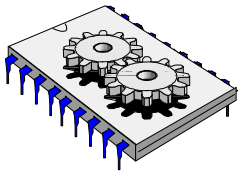


figure 6

Pour conclure

Pour terminer cette première approche autour d'un microcontrôleur PIC on peut déjà dire qu'avec un minimum de moyen (un programmateur de PIC), on va pouvoir réaliser une application simple ne comportant que très peu de composants et peu coûteuse. Il faudra bien évidemment se familiariser (petit à petit !) avec le langage assembleur qui paraît au premier abord assez compliqué et surtout avec les instructions (il n'y en a que 35) , ainsi qu' à la façon de les utiliser.

Nous allons dans le prochain cours "voyager" au coeur d'un PIC en détaillant les différents blocs qui constituent son architecture interne.



A la découverte des microcontrôleurs PIC
Deuxième partie

Nous voici comme promis dans notre dernier cours au coeur d'un microcontrôleur PIC, l'architecture présentée **figure 6** est celle d'un microcontrôleur PIC 16 F 84 sur lequel notre étude portera.

Avant d'étudier l'architecture interne d'un PIC, nous allons faire un bref rappel sur les bascules, pour nous amener jusqu' au "circuit registre" constituant essentiel dans l'architecture d'un PIC.

Il existe en logique combinatoire différentes bascules ayant chacune des caractéristiques et un mode de fonctionnement propre. Nous allons étudier ici le fonctionnement de l'une d'elles : la bascule D.

Une bascule D peut être construite à l'aide de 4 portes NAND et d'un inverseur tel que représenté sur la **figure 1.**

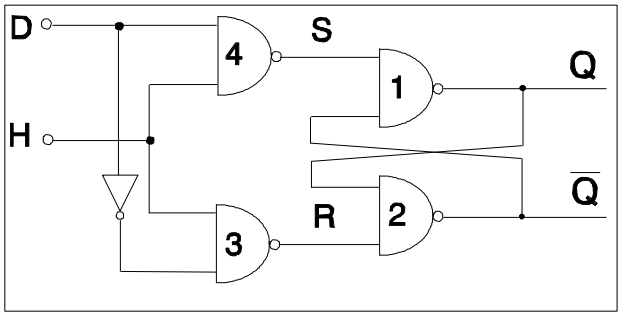


Figure 1

Représentation schématique d'une bascule D

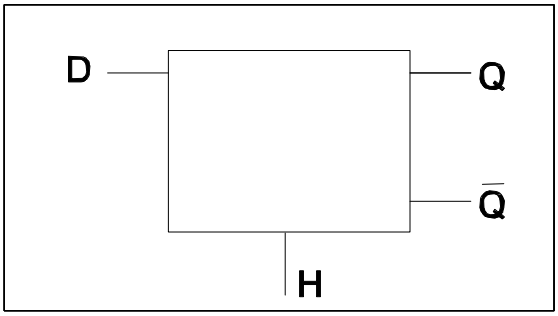


Figure 2

Mode de fonctionnement

- Tant que l'entrée de validation H est au niveau logique bas (0v) la sortie des portes NAND (3 et 4) est au niveau logique haut , en effet un 0v appliqué sur une des entrées d'une porte NAND provoque la mise au niveau logique "1" de la sortie de cette porte, on dit que le "0" est l 'élément absorbant sur une NAND
  (voir la table de vérité figure 3). La sortie des portes 3 et 4 restera "bloquée" au niveau logique "1" tant que l'entrée H est à 0 , ce qui signifie que le signal présent sur l' entrée D (data) ne modifiera pas les sorties des portes 3 et 4 (sortie R et sortie S). La sortie Q de la bascule D restera donc dans l'état précédent, on peut dire que la bascule D à mémorisé l'état antérieur (voir la table de vérité) .

A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

figure 3

- Passons maintenant l 'entrée H au niveau logique "1" , pour une porte NAND le niveau logique "1" sur l'une de ses entrées représente "l ' élément neutre" , c'est à dire que la sortie de la porte ne dépendra alors que de l 'état de la deuxième entrée logique (figure 3). Si l'entrée D est au niveau logique 1 la sortie de la porte 4 (S) passe au niveau logique "0" ce qui provoque une mise au niveau logique "1" de la sortie de la porte 1 appelée "Q".
  Un inverseur est inséré entre le signal d'entrée D et la porte 3 de ce fait, sachant que nous avons mis précédemment D à 1 cela signifie que la sortie de la porte 3 est au niveau logique "1". La deuxième entrée de la porte 2 est connectée sur la sortie de la porte 1 qui est au niveau logique "1", de ce fait la sortie Q barre de la porte 2 est donc au niveau logique "0".

  Conclusion : On peut dire que la sortie Q d'une bascule D recopie l'état de l'entrée D tant que le signal de validation H est au niveau logique "1". Si le signal H est au niveau logique "0 ", alors la bascule peut être assimilée à une cellule mémoire de 1 bit, car elle mémorise l'état antérieur de la sortie Q.

D	***H***	***Q***	***Q/***	***Commentaires***
1	0	0	1	État initial
1	1	1	0	La sortie Q prend l'état de D ("1")
1	0	1	0	Mémorisation de l'état antérieur
0	0	1	0	Mémorisation de l'état antérieur
0	1	0	1	La sortie Q prend l'état de D ("0")
0	0	0	1	Mémorisation de l'état antérieur
1	0	0	1	Mémorisation de l'état antérieur
0	1	0	1	Recopie de l'entrée D
1	1	1	0	Recopie de l'entrée D

Figure 4 Table de vérité de la bascule D

- Comment utiliser des bascules D pour former une mémoire statique ?

de la bascule à la mémoire ...

Nous venons de voir le fonctionnement d'une bascule D unique, celle-ci peut être assimilée à une cellule mémoire de 1 bit (en effet on ne mémorise qu'une seule information binaire), pour réaliser par exemple une mémoire 8 bits (1 octet), on pourra juxtaposer 8 bascules D comme représenté sur le schéma de la **figure 5.**
La donnée à mémoriser est présentée via les interrupteurs sur les entrées Data des bascules D (D0 à D7) puis lorsque l'on veut mémoriser l'état de ces 8 entrées, il suffit d'appliquer une impulsion sur la broche de validation H (remarquez que toutes les entrées de validation H sont reliées ensembles). Les 8 leds connectées sur les sorties Q donnent l'état du contenu de chaque bascule. Le montage ci-après peut se réaliser à l'aide d'un circuit TLL comportant 8 bascules D tel que le 74374.

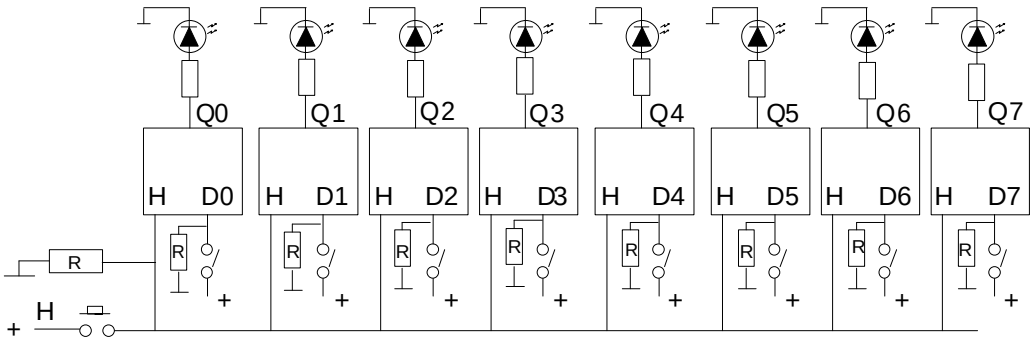


Figure 5

Cette mémoire 8 bits peut également s'appeler registre 8 bits. Dans l'architecture interne d'un microcontrôleur PIC nous retrouverons de nombreux registres semblables à celui-ci. Le PIC 16F84 possède 15 registres ayant chacun une fonction bien définie.

- **Architecture interne simplifiée du PIC 16 F 84**

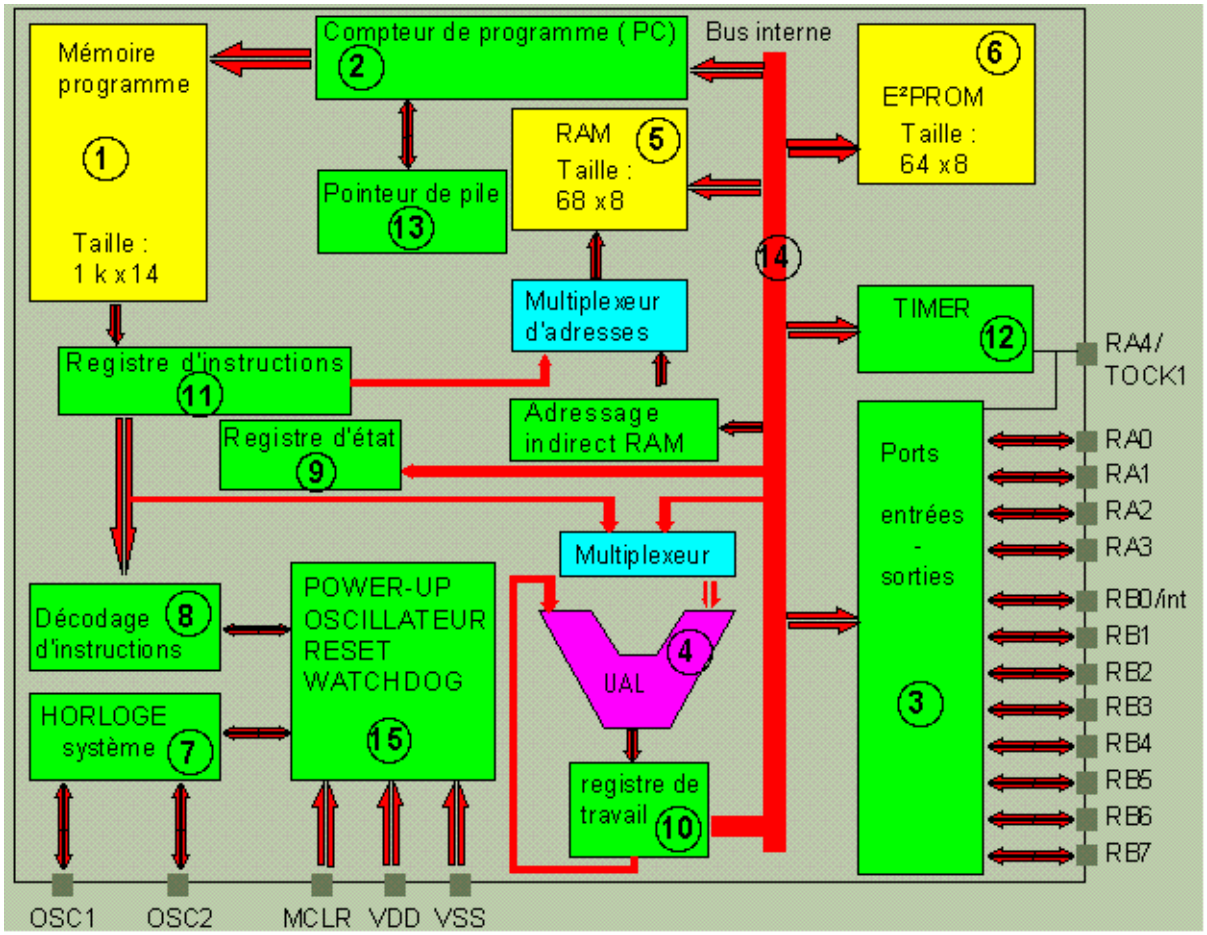


figure 6

- 1 - Mémoire programme
- 2 - Registre compteur de programme
- 3 - Port A et Port B d'entrées - sorties
- 4 - Unité Arithmétique et logique
- 5 - RAM
- 6 - E²PROM
- 7 - Horloge système
- 8 - Registre de décodage des instructions
- 9 - Registre d'état
- 10 - Registre de travail
- 11 - Registre d'instruction
- 12 - Timer
- 13 - Pointeur de pile
- 14 - Bus internes
- 15 - Reset ; Watch dog ; Alimentation

Rôle et description des principaux blocs constituants l'architecture d'un PIC

- La mémoire de programme :

Sur le PIC présenté (16F84) on retrouve une mémoire de type flash EPROM ayant une capacité de 1024 instructions (rep: 1) . Le constructeur donne environ 1000 cycles d'effacement et d'écriture pour cette mémoire, ce qui nous laisse une marge assez confortable pour mettre au point un programme .Chaque instruction est codée sur 14 bits cela signifie que la mémoire programme du PIC 16F84 à une capacité de 1024x14 bits.
C'est dans cette mémoire que sera stocké votre programme (compilé) qui correspondra aux instructions que devra effectuer le microcontrôleur .Les 1024 (1k) instructions possibles semblent un peu dérisoires face aux "méga-octets" d'aujourd'hui , mais vous verrez par la suite que cela suffit largement pour une application "grand public", il ne faut pas oublier également que dans chaque case mémoire on peut stocker 14 bits , ce qui permet d'utiliser des instructions plus "puissantes" que dans une mémoire traditionnelle 8 bits.

Largeur d'une case mémoire

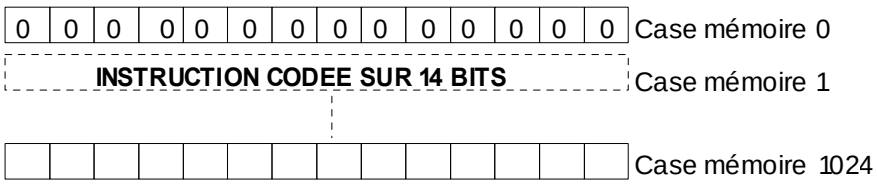


Figure 7

Organisation de la mémoire de programme

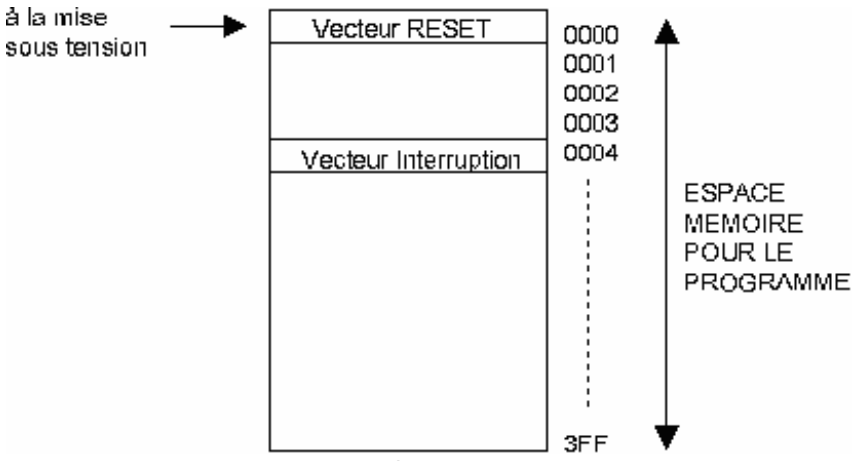


Figure 8

- A la mise sous tension c'est la case située à l'adresse 0 qui sera lue (c'est ce que l'on appel le vecteur reset). Nous verrons par la suite lors d'une application que le PIC 16F84 peut travailler également en mode interruption (selon 4 sources différentes), dans ce cas la case mémoire qui sera pointée se trouve à l'adresse 4 (c'est le vecteur interruption).

- Le compteur de programme (CP ou PC en anglais pour Programm Counter)

  Nous venons de voir que la mémoire programme contient les codes binaires du programme que nous avons définit. Nous avions également vu dans le premier cours que le microcontrôleur exécute une à une les instructions stockées dans la mémoire.
  Un registre interne nommé "compteur de programme" (**rep : 2**) va être chargé de pointer (sélectionner) chaque case mémoire une à une, afin que le microcontrôleur puisse exécuter l 'instruction correspondante au code binaire stocké dans la case mémoire concernée .Le contenu du registre du pointeur de programme augmentera au grès de l ' exécution des instructions , le PC pointe toujours la prochaine instruction à exécuter.
  Dés que le microcontrôleur est alimenté le contenu du registre compteur de programme est remis à zéro ce qui fait que c'est case mémoire située à l'adresse 0 qui va être pointée la première (vecteur RESET).

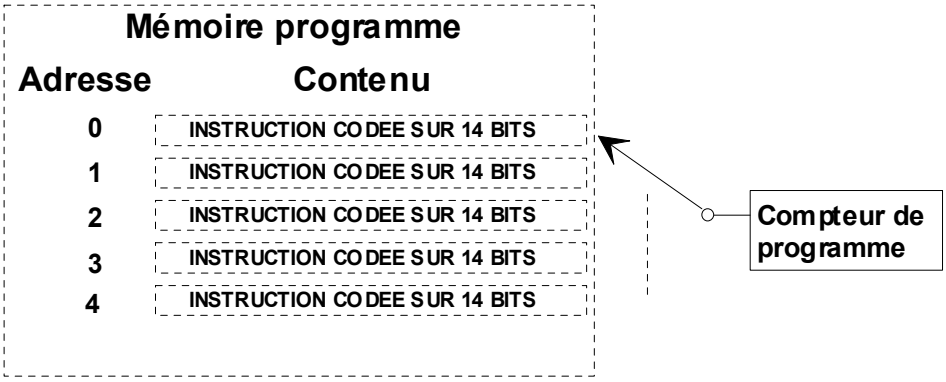


Figure 9

Le compteur de programme possède une largeur 13 bits, il peut donc adresser une mémoire de 8 k ($2^{13} = 2^{10} \times 2^{3}$ soit 1 k x 8 = 8 k) .

Le compteur de programme s'est incrémenté alors que l'exécution de l'instruction précédente est en cours.

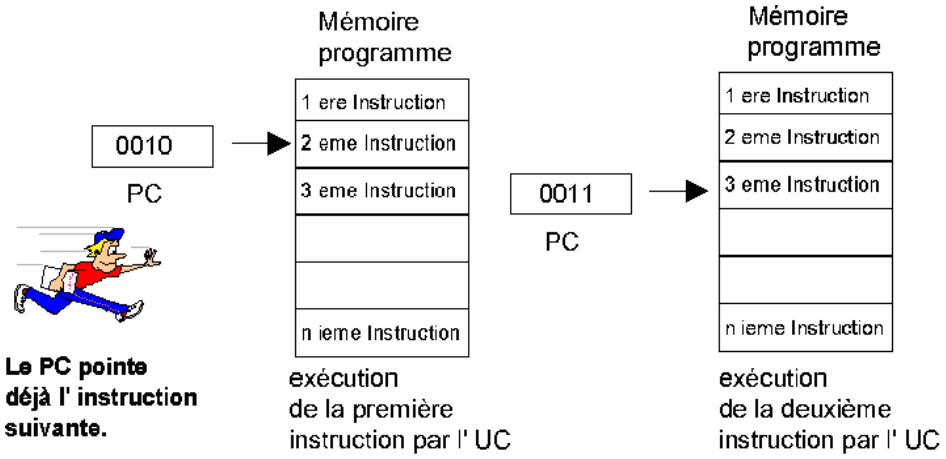


figure 10

Cheminement d'une instruction

- Le registre de contrôle et décodage des instructions

  Le programme binaire correspondant à votre source est maintenant dans la mémoire programme du microcontrôleur PIC, le compteur de programme pointe l'instruction a exécuter, cette instruction est analysée par un registre de contrôle et de décodage des instructions véritable analyseur logique, qui est chargé de définir ce que le microcontrôleur devra effectuer comme opération. Le contrôleur et décodeur d'instruction définissent la stratégie des actions à accomplir en interne pour effectuer l'instruction demandée.

- L ' unité arithmétique et logique

  L ' unité arithmétique et logique est considérée souvent comme étant le coeur de l 'unité centrale (UC) , en effet c'est elle qui va être chargée d 'effectuer toutes les opérations de type arithmétique (addition , soustraction etc...) ou bien de type logique (rotation , décalage , complément etc...) .
  Selon l'opération à effectuer le contrôleur et décodeur d'instruction enverra les signaux nécessaires à l ' unité arithmétique et logique pour pouvoir accomplir l 'opération demandée.

  Exemple : Nous voulons soustraire deux nombres, comment l'unité centrale va elle procéder ?
  - Dans un premier temps la première valeur à soustraire va être stockée dans le registre de travail W, remarquez que le registre de travail est relié sur une des entrées de l Unité Arithmétique et Logique.
  - Puis la deuxième valeur à soustraire est dirigée vers une autre entrée de l ' Unité Arithmétique et Logique. Ensuite un code indiquant qu'une soustraction doit être effectuée est envoyé vers l ' Unité Arithmétique et Logique qui exécute cette instruction.
  - Le résultat de la soustraction est stockée dans le registre de travail qui lui même est relié au bus de donnée interne, cela veut dire que le résultat peut être transféré en interne vers n'importe quel registre.

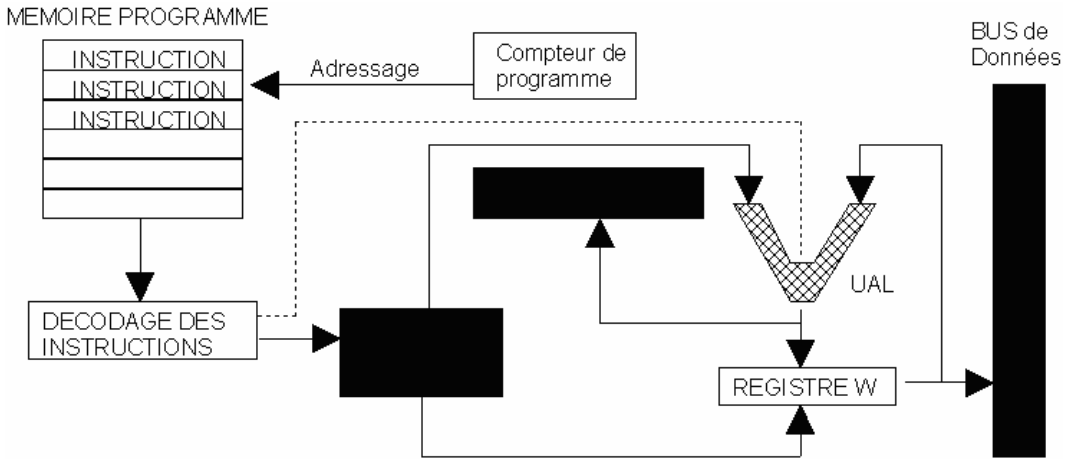


Figure 11

- Le registre de travail (registre W rep : 10)

  L Unité Arithmétique et Logique est en étroite relation avec un registre nommé W, (work register) c'est un registre de travail qui correspond aux anciens "accumulateurs" sur les microprocesseurs et par lequel vont transiter un bon nombre d' informations que ce soit une donnée à traitée (pour réaliser par exemple une addition , une soustraction etc ...) ou bien pour stocker le résultat d 'une opération ou d' un traitement.
  Nous verrons par la suite lorsque nous réaliserons des programmes que ce registre est très important, en effet l'accès à certain registre du PIC ne peut se faire directement, nous sommes obligés de "passer" la valeur à lire ou à écrire par ce fameux registre de travail. Le fait que le registre de travail soit relié au bus de données interne permet à celui-ci d'être en relation avec le reste de l'architecture du PIC (RAM , E²PROM , TIMER , PORTS A et B , etc...).

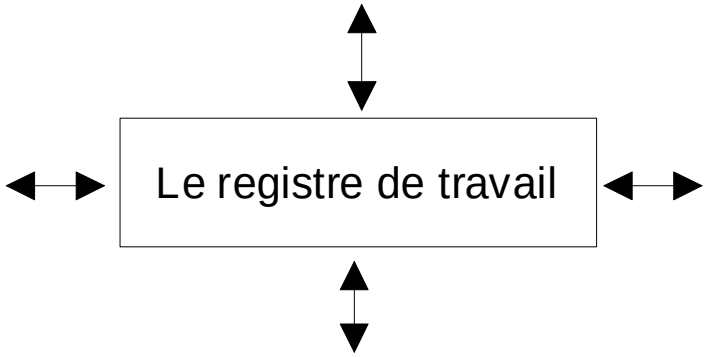


Figure 12

- Le registre d état (ou registre status)

A chaque fois que l'on devra faire un test au cours d'un programme nous allons utiliser sans le savoir un registre interne appelé registre d'état qui est en relation avec le résultat de la dernière opération demandée au microcontrôleur PIC. C'est un registre qui contient 8 bits ayant chacun un rôle bien particulier.

Exemple d 'un test au cours d 'un programme :

Nous avons réalisé une temporisation et nous devons tester si celle-ci est terminée pour passer à la suite du programme, comment le microcontrôleur va t'il gérer ce programme ?

- Pour réaliser une temporisation nous allons "charger une valeur dans un registre du PIC, puis nous allons décrémenter cette valeur jusqu ' à atteindre la valeur 0 ce qui définira la fin de notre temporisation. Pour pouvoir dire que le registre que nous avons utilisé est bien à 0, nous allons utiliser une instruction de test qui va nous avertir quand le contenu du registre sera égal à 0. Cette instruction de test contrôle l ' état d 'un bit du registre d' état , le bit Z (comme zéro) qui passera à 1 lorsque le résultat de la dernière opération effectuée vaudra 0.
Cela peut paraître assez difficile au premier abord mais avec l'habitude. On se fait à tout. (Figure 13)

Chronogramme de la temporisation

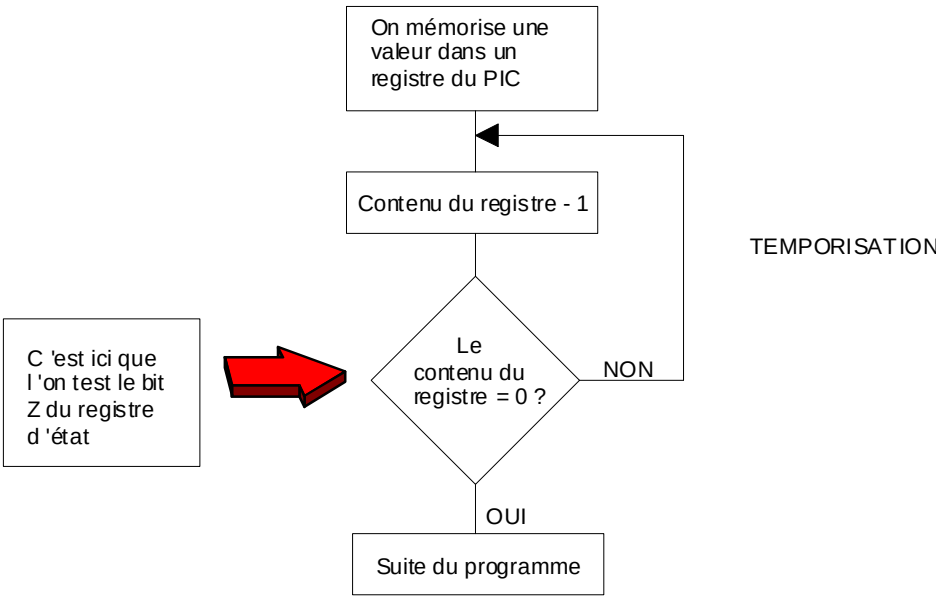


figure 13

Les bits du registre d 'état

IRP	RP1	RP0	TO/	PD/	Z	DC	C

- Bit C (carry) : Ce bit du registre d' état va passer à "1" lorsque le résultat de la dernière opération arithmétique à provoqué une retenue.

- Bit DC (digit carry) : Ce bit du registre d' état va passer à "1" lorsque le résultat de la dernière opération arithmétique à provoqué une retenue sur les quatre premiers bits, ce bit (ou bien flag ... pour drapeau) sera utilisé lorsque l 'on travaillera en BCD (binaire codé décimal) .

- Bit Z (zéro) : Ce bit du registre d' état va passer à "1" lorsque le résultat de la dernière opération est égal à zéro. Dans l'exemple de la temporisation précédente on utilise ce bit zéro.

- Bit PD/ (power down) : Ce bit du registre d'état va passer à "0" lorsque le microcontrôleur rencontre l'instruction particulière "Sleep" (mise en sommeil) qui détermine le mode de mise en veille PIC en bloquant les impulsions d'horloge nécessaires au cadencement de tous les échanges, le PIC attend alors un événement pour "repartir".

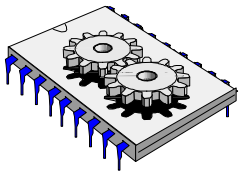
- Bit TO/ (time out) : Ce bit du registre d' état va passer à "0" lorsque le chien de garde interne (nous reviendrons ultérieurement sur son fonctionnement) à atteint la fin de comptage que le programmeur lui à définit. Le chien de garde (ou watch dog) peut être désactivé, il ne servira que lorsque nous voudrons savoir si le programme se déroule correctement.

- RP0 et RP1 sont deux bits qui permettent d'accéder à deux zones mémoire RAM différentes (bank 0 et bank 1), le prochain cours détaille le fonctionnement de ces deux bits

- IRP : Bit de réserve pour application future, en ce qui concerne le PIC que nous étudions c'est à dire le PIC 16F84 ce bit ne sert pas.

Pour terminer cette deuxième partie ...

Nous avons passer en revue certains registres constituant l'architecture d'un PIC, dans le prochain cours nous allons continuer notre "voyage" dans l'architecture interne afin de pouvoir débuter notre cours sur la programmation et donc notre premier programme.



A la découverte des microcontrôleurs PIC
Troisième partie

Architecture interne suite...

- **Le pointeur de pile**

Le pointeur de pile (rep :13) ou SP en anglais (stack pointeur) est un registre pouvant mémoriser huit adresses différentes, on dit que le pointeur de pile est à huit niveaux .
Le rôle du pointeur de pile consiste à mémoriser l'adresse courante lorsque le programme principal est dérouté vers un sous-programme. En effet lors d'un saut vers un sous-programme le compteur de programme se charge avec l'adresse de celui-ci, lorsque le sous-programme se termine le compteur de programme doit repointer alors la case mémoire suivant l'instruction de saut (voir **figure 1**) pour pouvoir reprendre le programme principal ou celui-ci a été dérouté, le pointeur de pile va alors rechercher automatiquement l'adresse qu'il a mémorisé et il "recharge " le compteur de programme avec cette adresse. Le pointeur de pile sera sollicité dès que dans un programme il y aura un sous-programme. Comme nous le verrons par la suite lors d'un programme fonctionnant avec une interruption le pointeur de pile aura le même rôle de sauvegarde de l'adresse courante.
Les huit niveaux du pointeur de pile veulent signifier que l'on peut imbriquer huit sous-programmes.
Il est à noter que le pointeur de pile est autonome c'est à dire qu'il gère tout seul la mémorisation et la restitution d'une adresse (ouf ...) .

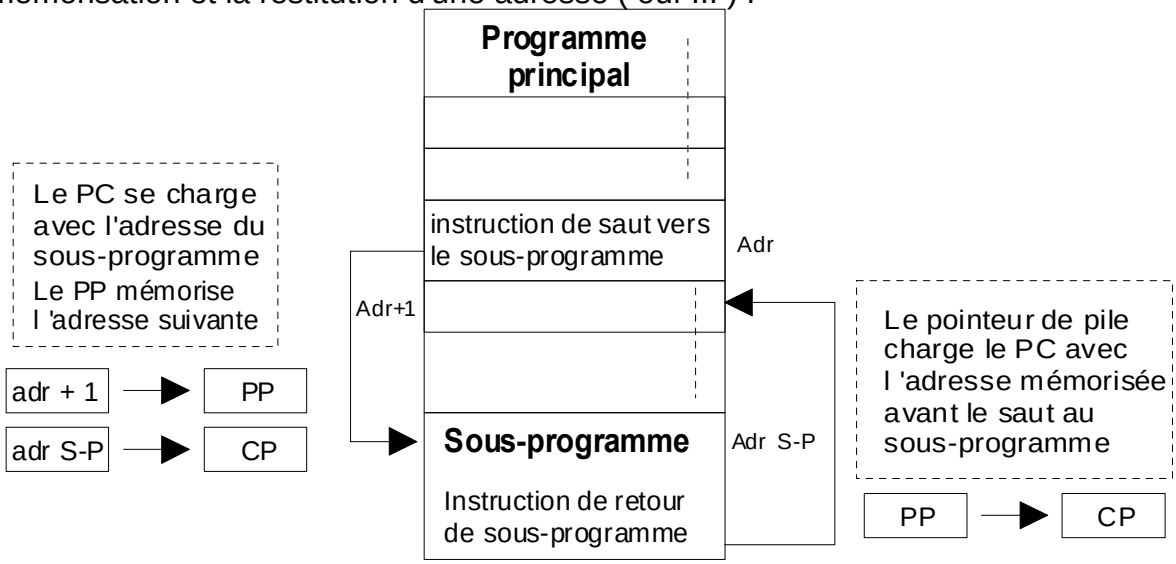
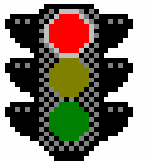


Figure 1

Qu'est-ce qu'un sous-programme ?
Un sous-programme est une suite d'instructions correspondant à une fonction bien définie à laquelle votre programme principal fera appel plusieurs fois. En simplifiant le fait d'écrire un ou plusieurs sous-programmes vous évitera d'écrire plusieurs fois la même chose, d' ou une économie de place en mémoire programme et bien sur une économie de temps.

Prenons un exemple :
Nous devons réaliser un feu tricolore en ayant des temps d'allumage et d'extinction identiques pour chaque lampe, par exemple 3 secondes, la première façon de procéder est la suivante :



- allumage lampe rouge et extinction des autres lampes
- temporisation 3 s
- allumage lampe verte et extinction des autres lampes
- temporisation 3 s
- allumage lampe orange et extinction des autres lampes
- temporisation 3 s
- retour à la première instruction

On voit bien dans cette première façon de réaliser le programme on a écrit trois fois les instructions définissant la temporisation de 3 secondes, ce qui représente en langage assembleur 3 fois 10 instructions soit 30 instructions

La deuxième façon de traiter le problème est de définir un sous-programme de temporisation à qui l'on fera appel autant de fois que nécessaire, ce qui revient à dire que l'on va écrire une seul fois la temporisation de 3 s.

- allumage lampe rouge et extinction des autres lampes
- appel du sous-programme de temporisation 3 s (**1 instruction**)
- allumage lampe verte et extinction des autres lampes
- appel du sous-programme de temporisation 3 s (**1 instruction**)
- allumage lampe orange et extinction des autres lampes
- appel du sous-programme de temporisation 3 s (**1 instruction**)
- retour à la première instruction

Sous programme de temporisation

- temporisation 3 s
- retour de sous-programme

Dans ce deuxième exemple on voit bien que cette fois ci nous avons écrit qu'une seule fois la temporisation en créant un sous-programme.

- **La mémoire RAM**

Lorsque vous allez créer un programme vous allez pouvoir faire appel à des données qui seront stockées temporairement, la mémoire RAM interne du PIC 16F84 vous met à disposition 68 octets banalisés vous permettant à votre grès de sauvegarder ou de rapatrier une information.

Exemple : nous devons réaliser une temporisation, comment allons nous procéder ?

Dans un premier temps nous allons dans un emplacement de la mémoire RAM stocker une valeur qui dépendra de la durée de la temporisation souhaitée. Puis nous allons créer un sous-programme qui va se reboucler (**figure 2**) tant que la valeur que nous avons stockée en mémoire ne vaut pas 0. Dans la boucle du sous-programme bien sur nous allons décrémenter la variable mémorisée. Une instruction de test va permettre de vérifier le contenu de la variable à analyser.

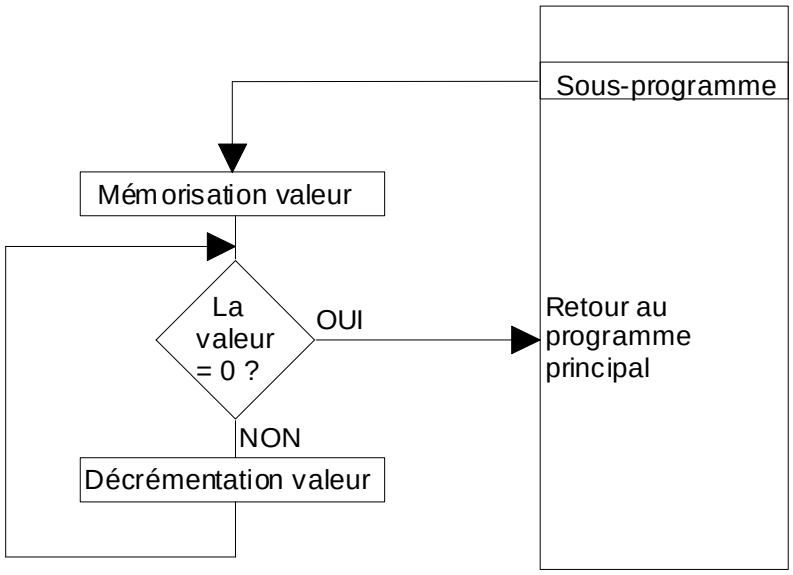


Figure 2

Organisation de la mémoire RAM

La mémoire RAM est organisée en 128 octets ayant chacun une fonction bien définie (figure ...) . Nous avons vu précédemment que pour l'utilisateur 68 octets sont réservés.
Les 11 premiers octets de l' espace mémoire sont réservés pour la configuration et l 'accès à certains registres spécifiques comme par exemple les ports d 'entrées - sorties A et B, le registre STATUS etc...
De l ' adresse 11 à l 'adresse 79 nous retrouvons les 68 octets réservés à L'utilisateur.

Un peu plus compliqué ...

En fait la mémoire RAM est scindée en 2 parties appelées bank 0 et bank 1 ces deux parties sont accessibles par la même adresse, la sélection de telle ou telle bank se fait selon la position (0 ou 1) de deux bits de contrôle , contenus dans le registre STATUS qui est situé à l 'adresse 3 en mémoire RAM . Ces deux bits de contrôle se nomment RP0 et RP1.
Si RP1 = 0 et RP0 = 0 alors on accède à la mémoire située en bank 0
Si RP1 = 0 et RP0 = 1 alors on accède à la mémoire située en bank 1
Lorsque nous allons commencer les cours de programmation nous devrons accéder à différents registres situés dans chacune des deux zones mémoire.
La zone située en bank 1 entre l'adresse 140 et 208 est une image de la bank 0 , de plus certain registres tels que le registre STATUS sont accessibles depuis n'importe quelle bank (bank 0 ou bank 1) .

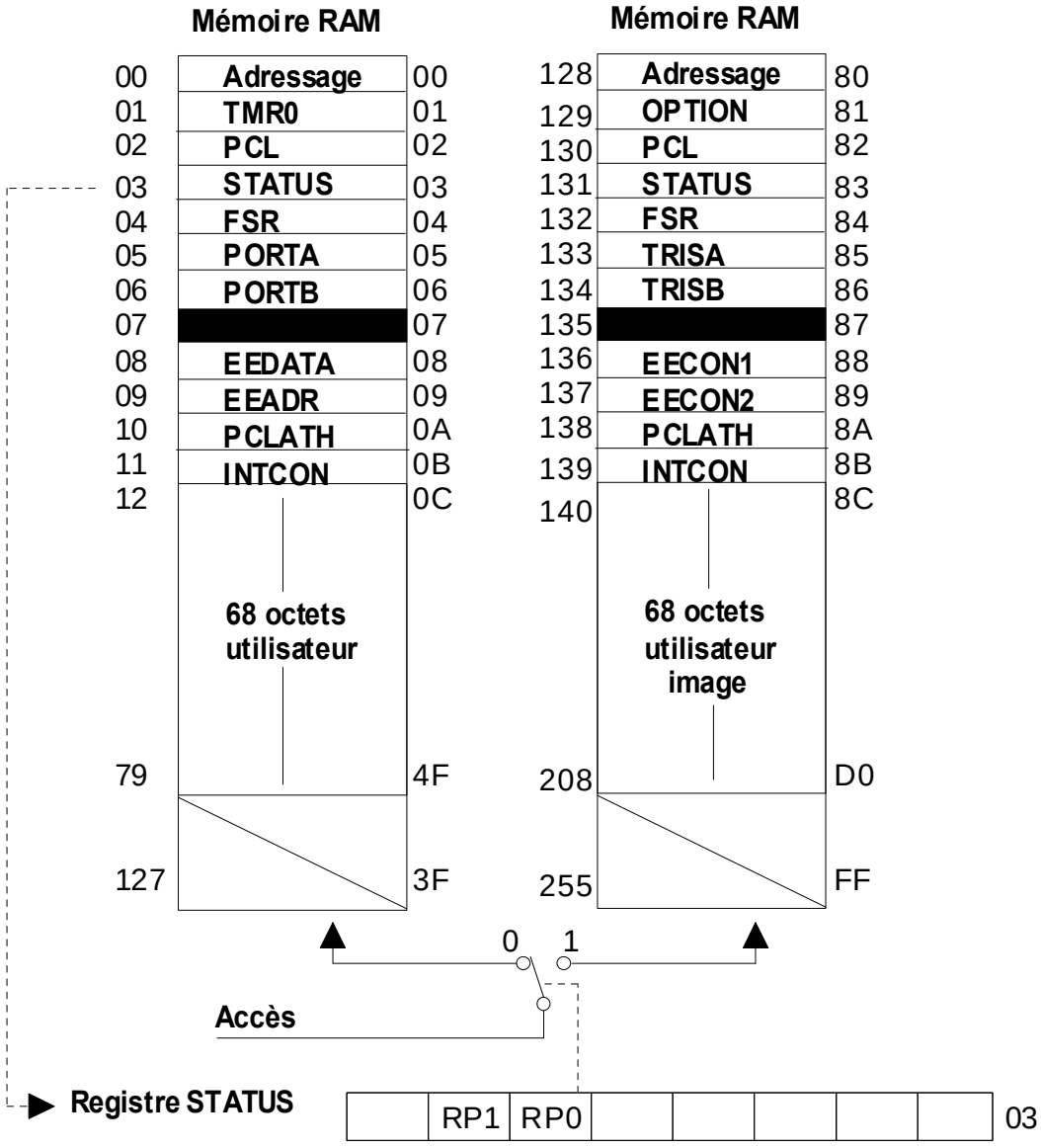
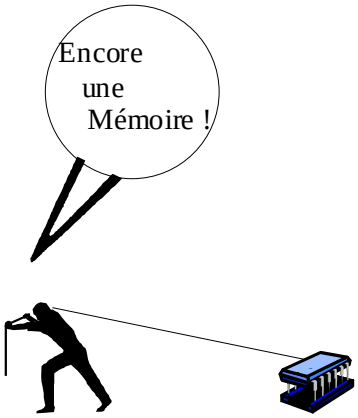


Figure 3

- **La mémoire EEPROM** (Electrically Erasable Programmable Read Only Memory)

La mémoire E²PROM est une particularité du PIC 16F84, c'est une zone comportant 64 octets mise à la disposition de l'utilisateur et dont la particularité est de pouvoir être sauvegardée même en absence d'alimentation.
Cette mémoire peut être très utile pour par exemple : mémoriser le nombre de passage vers un programme, mémoriser un code défaut, mémoriser une valeur invariable (ex : π = 3.14) , mémoriser un code d'accès , mémoriser la version d'un programme , etc ...
Il est à noter toutefois que les temps pour accéder à cette mémoire sont relativement longs par rapport à un accès en RAM par exemple.
La première case de la mémoire E²PROM débute à l'adresse hexadécimale 2100 h.
Un ordre spécifique permet lors du transfert du fichier compilé d ' écrire dans la mémoire E²PROM.
Quatre registres spécifiques (EEDATA ; EEADR ; EECON1 et EECON2) permettent les accès à l ' E²PROM que ce soit en lecture ou en écriture. Nous verrons lors d'une application la façon de lire et d'écrire dans cette mémoire un peu particulière.



- Le timer : Un timer est un registre 8 bits dans lequel on charge une valeur . Son fonctionnement est cadencé soit avec l'horloge interne, soit avec une horloge externe (ou bien des fronts sur une broche spécifique du PIC).
  Dès que le timer est validé, il réalise un comptage depuis la valeur que vous avez prédéterminée jusqu ' à 255, il vous prévient (on le verra par la suite en provoquant une interruption) alors que son contenu passe de 255 à 0, puis recommence son comptage.
  Nous reviendrons largement sur son fonctionnement exact et nous réaliserons une application, cette première approche nous permet de se familiariser avec le TIMER.
  Parmi les applications nombreuses pouvant être réalisées avec le TIMER citons la base de temps, la temporisation, le comptage, etc...
  La configuration du TIMER se réalisera dans un registre spécifique du PIC : le registre OPTION.

- Les Ports d'entrées - sorties

Pour dialoguer avec l'extérieur (application) le PIC 16F84 vous met à disposition 13 Entrées-Sorties programmables individuellement soit en entrée soit en sortie (**rep :3**).
Ces 13 entrées - sorties sont issues de 2 ports nommés PORT A pour les cinq entrées - sorties RA0 à RA3 et PORT B pour les huit entrées - sorties RB0 à RB7.

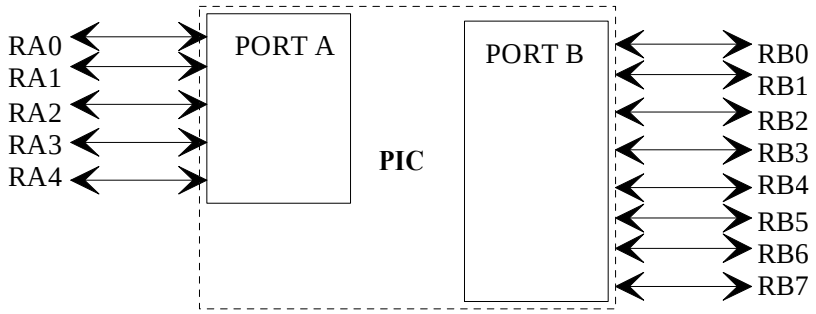


Figure 4- Les ports d'entrées - sorties du PIC 16F84

Exemple de configuration du port A et du port B :

Imaginons que pour réaliser une serrure codée , nous avons besoin de 4 lignes de sorties pour piloter les 4 colonnes d'un clavier matricé , 4 lignes d'entrées pour recevoir les 4 lignes du clavier et puis 1 entrée pour lancer le programme et 1 sortie pour piloter un relais.

Le schéma équivalent peut être celui-ci :

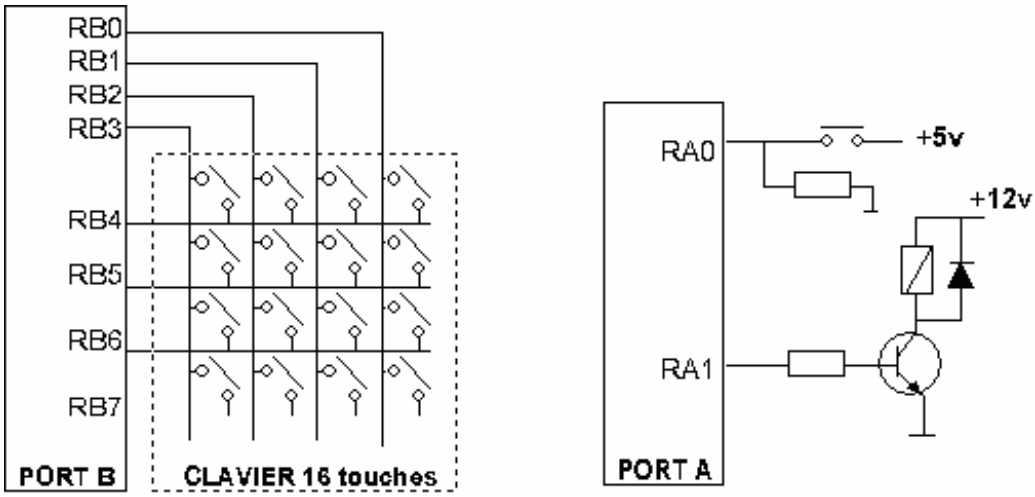


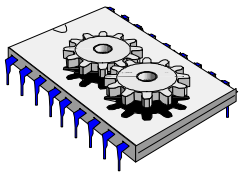
Figure 5

Nous utilisons RB0 à RB3 configurées en sortie pour piloter les 4 colonnes du clavier et RB4 à RB7 pour recevoir les lignes du clavier à 16 touches. Le principe de décodage est le suivant on passe à "1" séquentiellement chaque colonne et l'on vérifie l ' état de chaque ligne, selon la colonne qui est alimentée et la ligne activée, on détermine la touche appuyée.

La ligne RA0 du port A est configurée en entrée pour recevoir l 'état du poussoir de mise en service et la ligne RA1 est configurée en sortie pour pouvoir actionner un relais.
Dans le prochain cours nous verrons comment programmer les registres de contrôle du port A (TRISA) et du port B (TRISB) afin de configurer leurs broches soit en entrée soit en sortie.

Pour conclure...

Avec cette troisième partie nous avons vu simplement la plupart des constituants internes d'un PIC. Le fonctionnement exact de certain bloc tel que le TIMER, la mémoire E²PROM, le port d'entrées - sorties, le watch dog sera détaillé lorsque nous les utiliserons dans un programme. Le temps est donc venu de s'intéresser un peu au " hard " du PIC 16F84, nous verrons dans la prochaine partie le brochage ainsi que les conditions de RESET et l'horloge système.



A la découverte des microcontrôleurs PIC
Troisième partie

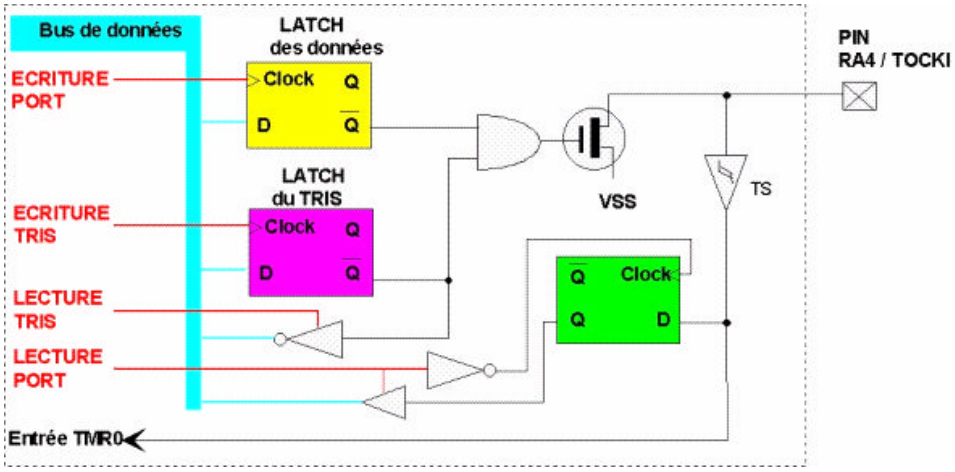
Dans notre dernière leçon nous avons vu les registres internes du PIC 16f84, nous allons revenir aujourd'hui plus en détail sur les ports d'Entrées-Sorties ainsi que sur leur mode de configuration.
Les deux ports du PIC 16F84 sont respectivement composés de 5 broches d'Entrées-Sorties pour le port A (RA0 à RA4) et de 8 broches Entrées-Sorties pour le port B (RB0 à RB7).
Toutes les broches de ces deux ports peuvent être configurées soit en entrée soit en sortie et certaines d'entres-elles peuvent avoir plusieurs rôles (voir tableau 1 et tableau 2)

- **Le Port A**

Nom de la broche	Fonction(s)	Type d'entrée sortie
RA0	Entrée / sortie	Type TTL
RA1	Entrée / sortie	Type TTL
RA2	Entrée / sortie	Type TTL
RA3	Entrée / sortie	Type TTL
RA4	Entrée / sortie et entrée clock du timer	Type trigger de schmitt

Tableau 1

La particularité pour le port A (tableau 1) concerne la ligne RA4 (figure 1) qui peut être utilisée soit en entrée-sortie soit en entrée d'horloge pour le timer, nous reviendrons ultérieurement lorsque nous aborderons le timer sur le rôle exact du fonctionnement de RA4 dans cette configuration. Le synoptique des autres lignes (RA0 à RA3) est illustré figure 2



Synoptique du Port A pour RA0 – RA3

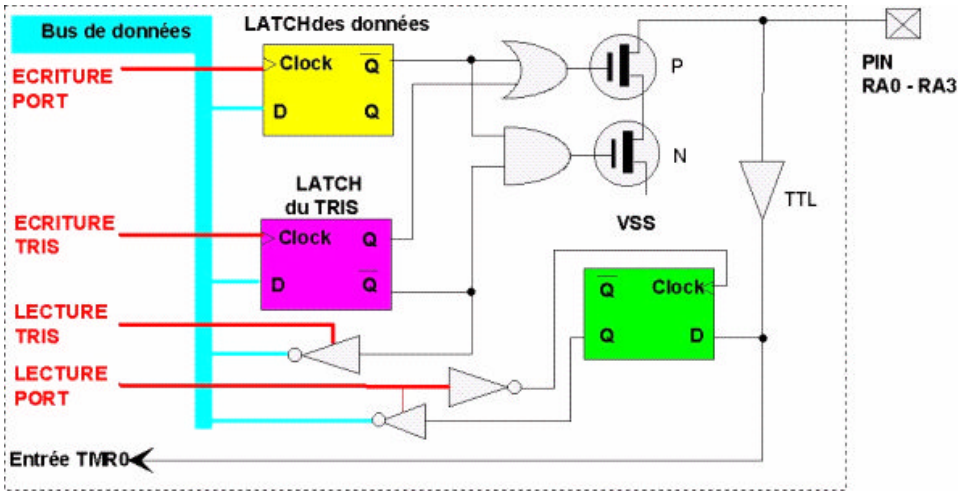


Figure 2

Le port B

Sur le port B toutes les lignes sont configurable bien sûr soit en entrée soit en sortie avec la possibilité d'utiliser un pull-up interne forçant la broche au niveau haut (voir figure 1), ce pull-up est configuré par logiciel via un bit nommé RBPU/ celui-ci fait partie du registre "OPTION" qui sera détaillé lors d'une prochaine leçon.
Il est à noter que le bit RBPU/ est actif à l'état bas ce qui signifie que pour avoir une résistance de rappel au + 5v en interne (pull-up) sur toutes les lignes (RB0 à RB7) il faut positionner ce bit à 0.

La broche RB0/INT peut être utilisée (nous le verrons dans un programme d'application lorsque nous aborderons les interruptions) comme entrée d'interruption (figure 3), dans ce cas il est possible de configurer le déclenchement d'une interruption par rapport à l'état de cette ligne soit sur un front montant ou bien sur un front descendant , tout cela est configurable par des bits à positionner dans différents registres interne lors de la phase de programmation.

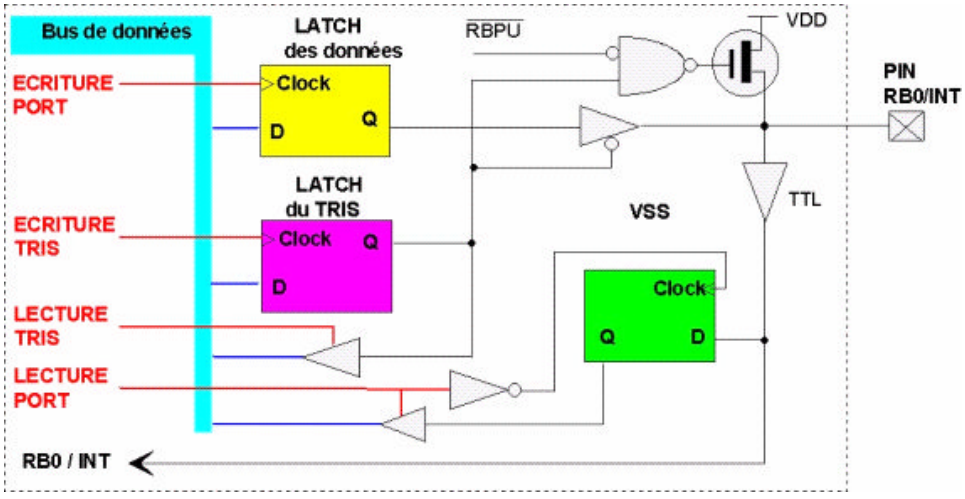


Figure 3

Les lignes RB4 à RB7 peuvent également êtres utilisées comme entrées d'interruption (figure 4) , dans ce cas un changement d'état sur une de ces entrées provoque un branchement du programme principal vers un sous-programme d'interruption que nous détaillerons bien sûr (restons simple dans un premier temps).
Enfin il existe une autre particularité des broches RB6 et RB7, en effet celles-ci sont utilisées lors de la phase de transfert du programme que vous avez réalisé et compilé vers la mémoire programme du PIC.

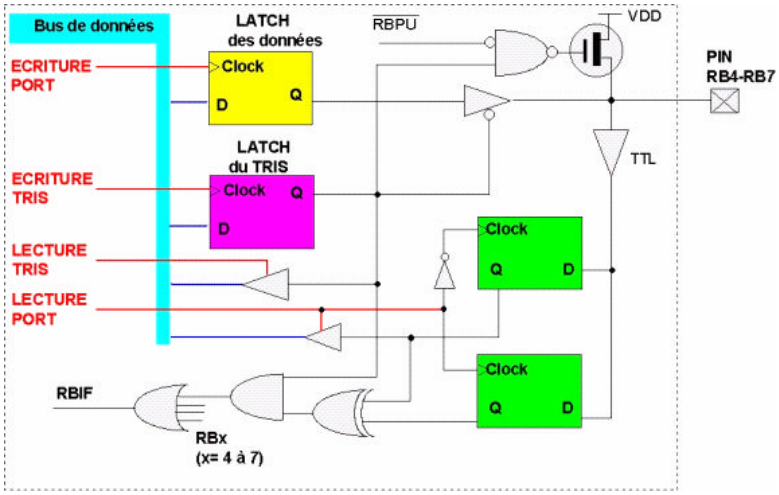


Figure 4

Tableau des lignes RB0 – RB7

Nom de la broche	Fonction(s)	Type d'entrée sortie
RB0 / INT	Entrée / sortie et entrée d'interruption sur fronts	Type TTL (avec pull-up configurable)
RB1	Entrée / sortie	Type TTL (avec pull-up configurable)
RB2	Entrée / sortie	Type TTL (avec pull-up configurable)
RB3	Entrée / sortie	Type TTL (avec pull-up configurable)
RB4	Entrée / sortie et entrée d'interruption sur changement d'état	Type TTL (avec pull-up configurable)
RB5	Entrée / sortie et entrée d'interruption sur changement d'état	Type TTL (avec pull-up configurable)
RB6	Entrée / sortie et entrée d'interruption sur changement d'état	Type TTL / Trigger de schmitt (avec pull-up configurable)
RB7	Entrée / sortie et entrée d'interruption sur changement d'état	Type TTL / Trigger de schmitt (avec pull-up configurable)

Tableau 2

Configuration du registre TRIS

Lorsque vous allez utiliser un PIC et créer un programme il sera nécessaire de configurer les ports A et B selon le projet que vous allez réaliser.
Deux registres internes spécifiques de huit bits nommés TRISA pour la configuration du port A et TRISB pour la configuration du port B vont vous permettre de définir le sens de chacune des treize lignes d'Entrées-Sorties.
Lorsque vous voulez configurer une ligne en entrée le bit correspondant du registre TRIS doit être à la valeur binaire "1" et la valeur "0" dans le cas d'une sortie.

Prenons de suite un exemple

Nous devons réaliser un projet nécessitant treize entrées-sorties à l'aide d'un PIC 16F84, nous avons à notre disposition les données suivantes :

PORT A :
Broche RA0 : interrupteur de mise en service
Broche RA1 : sortie sur un buzzer
Broche RA2 : sortie sur une led de signalisation
Broche RA3 : sortie sur un buzzer
Broche RA4 : sortie sur une led de signalisation

PORT B :
Broche RB0 : sortie sur une led de signalisation
Broche RB1 : sortie sur une led de signalisation
Broche RB2 : sortie sur une led de signalisation
Broche RB3 : capteur de fin de course
Broche RB4 : entrée de validation
Broche RB5 : sortie sur une led de signalisation
Broche RB6 : interrupteur d'arrêt d'urgence
Broche RB7 : sortie sur une led de signalisation

Après une rapide analyse on s'aperçoit que sur le port A il faudra configurer RA0 en entrée et RA1 RA2 RA3 et RA4 en sortie. Cela signifie que le registre TRISA devra être configurer comme suit :

Registre TRISA

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	X	X	**0**	**0**	**0**	**0**	**1**

Nous allons voir prochainement les instructions du PIC 16F84 et nous aborderons à cet occasion le langage assembleur. A titre d'exemple voici ce que donnerait le chargement du registre TRISA avec la valeur 0000 0001 correspondant à l'exemple précèdent.
On considérera dans l'exemple que les registres utilisés (TRISA et TRISB) sont préalablement déclarés.

Syntaxe :

ici on charge le registre de travail W avec 0000 0001

MOVLW b'00000001'

Puis on transfert le contenu du registre de travail W vers le registre TRISA

MOVWF TRISA

Voilà cela est tout, il faut deux instructions pour configurer le registre TRISA, une fois que celui-ci est chargé avec la valeur adéquate les broches sont configurées soit en entrée soit en sortie.

Pour le port B et selon l'énoncé de l'exemple on retrouve :
Registre TRISB

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	**1**	**0**	**1**	**1**	**0**	**0**	**0**

Syntaxe :

ici on charge le registre de travail W avec 0101 1000

MOVLW b'01011000'

Puis on transfert le contenu du registre de travail W vers le registre TRISB

MOVWF TRISB

Brochage du microcontrôleur PIC

Après ce rapide aperçu des PORT A et B du microcontrôleur PIC (figure 5) nous allons nous intéresser au brochage (figure 6) ainsi qu'à l'horloge système et au Reset.

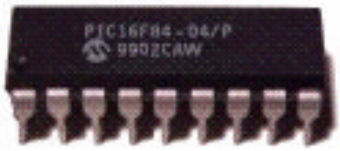


Figure 5

Le brochage du PIC 16F84

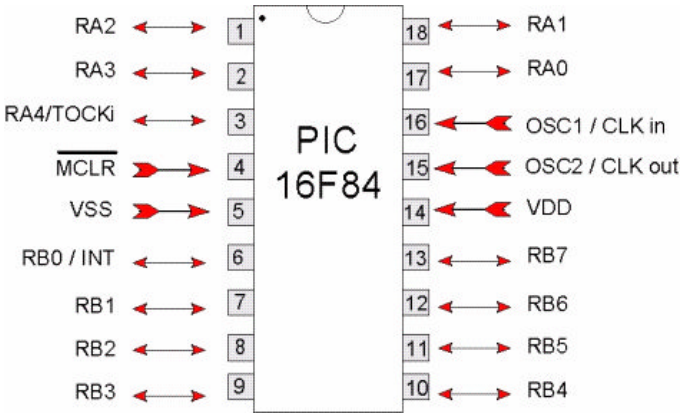


FIGURE 6

Fonction des broches :

PIN 1 : RA2 est la ligne d'entrée sortie N°2 du port A
PIN 2 : RA3 est la ligne d'entrée sortie N°3 du port A
PIN 3 : RA4/TOCKi est la ligne d'entrée sortie N°4 du port A et également l'entrée d'horloge du TIMER.
PIN 4 : MCLR/ est une broche active à 0 elle permet le reset du PIC lorsqu'elle est positionnée à l'état bas
PIN 5 : VSS est le zéro volt de l'alimentation
PIN 6 : RB0 est la ligne d'entrée sortie N°0 du port B, celle ci peut être également utilisée en tant qu'entrée d'interruption.
PIN 7 : RB1 est la ligne d'entrée sortie N°1 du port B
PIN 8 : RB2 est la ligne d'entrée sortie N°2 du port B
PIN 9 : RB3 est la ligne d'entrée sortie N°3 du port B

PIN 10 : RB4 est la ligne d'entrée sortie N°4 du port B
PIN 11 : RB5 est la ligne d'entrée sortie N°5 du port B
PIN 12 : RB6 est la ligne d'entrée sortie N°6 du port B
PIN 13 : RB7 est la ligne d'entrée sortie N°7 du port B
PIN 14 : VDD est relié au +5 V de l'alimentation
PIN 15 : OSC2 / Clk out est l'une des broche avec la PIN 16 sur laquelle sera relié le quartz permettent le cadencement du PIC, dans le cas d'une utilisation d'une horloge externe on retrouvera sur cette broche le signal d'horloge divisé par 4.
PIN 16 : OSC1 / Clk in est l'une des broche avec la PIN 15 sur laquelle sera relié le quartz permettent le cadencement du PIC, dans le cas d'une utilisation d'une horloge externe c'est sur cette broche que l'on appliquera le signal d'horloge.
PIN 17 : RA0 est la ligne d'entrée sortie N°0 du port A
PIN 18 : RA1 est la ligne d'entrée sortie N°1 du port A

Principe de cadencement d'horloge

Pour qu'un microcontrôleur fonctionne correctement il est nécessaire d'utiliser un signal d'horloge dont le rôle est de cadencé tous les échanges soit en interne (de

registre à registre par exemple), soit vers l'extérieur (registre vers un port d'Entrées-Sorties).

Sur le PIC 16F84 nous pouvons réaliser les tops d'horloge soit avec un quartz (figure 7), ce sera le cas dans la majorité des montages, ou bien nous pourrons réaliser l'horloge avec un condensateur et une résistance, ou bien encore avec une horloge extérieure provenant d'un autre circuit par exemple (figure 9).

Horloge à quartz

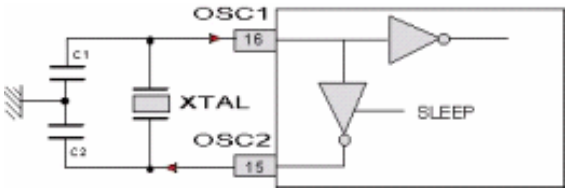


Figure 7

Mode	Fréquence quartz	C1 – C2
LP	32 kHz	68-100 pF
LP	200 kHz	15 -30 pF
XT	100 kHz	68-150 pF
XT	2 MHz	15 -33 pF
XT	4 MHz	15 -33 pF
HS	4 MHz	15 -33 pF
HS	10 MHz	15 -47 pF

Tableau 3

Fonctionnement avec un réseau RC (figure 8) : Il est possible de réaliser l'horloge avec un condensateur et une résistance, dans ce cas la tolérance du cadencement n'est plus aussi précise qu'avec un quartz, ce type d'horloge n'est pas indiquée si par exemple vous devez réaliser des temporisations de précision.
Le constructeur préconise une résistance comprise entre 3.3 k et 100 k et un condensateur compris entre 20 pF et 300 pF (tableau 4).

Horloge à Circuit RC

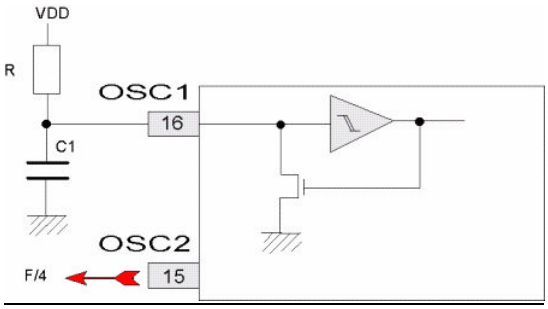


Figure 8

Tableau de correspondance pour horloge à circuit RC

R	C1	Freq osc	R	C1	Freq osc
3.3 k	20 pF	4.68 MHz	3.3 k	100 pF	1.49 MHz
5.1 k	20 pF	3.94 MHz	5.1 k	100 pF	1.12 MHz
10 k	20 pF	2.34 MHz	10 k	100 pF	620 kHz
10 k	20 pF	250.16 KHz	10 k	100 pF	90.25 KHz
3.3 k	300 pF	524.24 kHz			
5.1 k	300 pF	415.52 kHz			
10 k	300 pF	270.33 kHz			
10 k	300 pF	25.37 kHz			

Tableau 4

Horloge extérieure

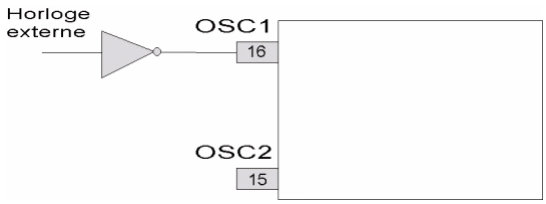


FIGURE 9

Il y a trois games de fréquence d'horloge reconnues par le PIC.

Le premier type est nommé "LP" pour low Power et qui est utilisable avec des fréquences de quartz allant de 32 kHz à 200 kHz., dans cette gamme de fréquence la consommation varie autour de quelques dizaines de microampères.

Le deuxième type est nommé "XT" pour XTal et qui est le plus utilisé, celui-ci est réalisé avec un quartz. Les fréquences d'oscillation sont comprises entre 100 kHz et 4 MHz. La consommation est d'environ 5mA.

Le troisième type est nommé "HS" pour High Speed et qui concerne les montages équipés d' un quartz ayant une fréquence comprise entre 4 MHz et 10 MHz. La consommation est d'environ 10mA

Pour simplifier, si vous réalisez votre horloge avec un circuit RC il faudra positionner lors de la programmation du PIC le choix de la configuration, cela se fera dans une variable nommée "_CONFIG", dans celle-ci nous inscrirons le type d'oscillateur choisit (ici se sera "RC"), dans le deuxième cas si vous utilisez un quartz allant jusqu'à 200 kHz il faudra mettre dans la variable _CONFIG le type "LP" dans le troisième cas si vous utilisez un quartz allant jusqu'à 4 MHz (le plus courant) il faudra mettre dans la variable _CONFIG le type "XT", dans le quatrième cas si vous utilisez un quartz 10 MHz par exemple (l'indication sur le pic permet de savoir la

fréquence max de fonctionnement ; sur un PIC 16F84 /04 ce sera 4 MHz) , il faudra mettre dans la variable _CONFIG le type "HS"

Exemple de déclaration avec un quartz de 4MHz

*_CONFIG XT_*OSC

FIGURE 10

Tout cela peut paraître compliqué, en fait sur de nombreux logiciels tel que ICPROG, il suffit de sélectionner dans une liste déroulante la configuration adaptée.

Circuit de Reset

Lors de la phase d'alimentation du PIC plusieurs registres internes vont se positionner dans un état connu. C'est le cas notamment du compteur de programme (que nous avons vu lors d'une précédente leçon) qui se chargera avec la valeur 0 correspondant à votre première ligne de programme. Une broche spécifique nommée "MCLR/" va permettre de réinitialiser le pic à tout moment, cela est bien utile lorsque votre programme se trouve dans un état indéterminé que vous n'avez pas souhaité (en fait c'est un plantage).

Le PIC 16F84 possède en interne un dispositif de reset automatique qui s'enclenche à la mise sous tension, il suffit uniquement de relier la broche MCLR/ au +5v, si vous souhaitez par contre pouvoir reseter à tout moment votre PIC il sera nécessaire de réaliser un schéma tel que celui présente figure 10.

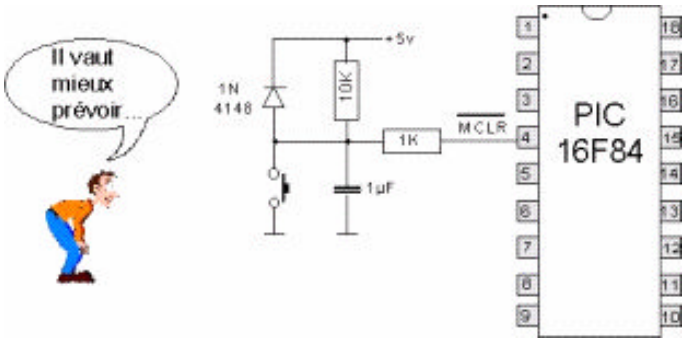
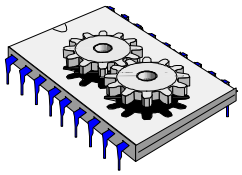


Figure 11

Pour conclure ce chapitre

Avec cette troisième partie nous avons vu les ports d'Entrées-Sorties un peu plus en détail ainsi que certaines instructions permettant de configurer le microcontrôleur. Nous allons aborder dans la prochaine leçon les trente cinq instructions de base du PIC 16F84 avant de commencer à écrire un premier programme.



A la découverte des microcontrôleurs PIC
Quatrième partie

C'est la rentrée pour beaucoup d'entres nous et comme promis dans notre dernière leçon nous allons aujourd'hui aborder les instructions qui régissent le fonctionnement d'un programme pour le PIC 16f84.Il est à noter que ces instructions que nous allons détailler sont communes à de nombreux PIC de la même famille et notamment au PIC 16F628 qui est compatible broche à broche avec le PIC 16F84.

Les instructions sont des commandes préprogrammées et figées dans le PIC, elles permettent à un programmeur de les enchaîner pour réaliser un programme. Il ne faut surtout pas oublier qu'un programme n'est ni plus ni moins qu'une succession d'ordres devant être exécutés par le microcontrôleur (**figure 1**). Dans une précédente leçon nous avons vu que le registre compteur de programme (CP) du PIC était chargé de pointer la case mémoire contenant l'instruction devant être exécutée, puis ce fameux registre CP s'incrémente automatiquement afin d'aller chercher la prochaine instruction et ainsi de suite jusqu'à la fin du programme, qui peut bien sûr se reboucler…
Comme nous l'avons vu dans une de nos dernières leçon, le PIC 16F84 ne dispose que de 35 instructions, ce qui peut paraître peu aux yeux de certain mais nous le verrons au cours de cette leçon, ce nombre est largement suffisant pour réaliser un programme, d'autant plus que certaines instructions élaborées peuvent générer plusieurs commandes successives. Il faut également savoir que toutes les instructions excepté les sauts se font en un cycle d'horloge d'ou une très grande rapidité d'exécution, avec un quartz de 4 Mhz on atteindra le MIPS (1 Million Instruction Par Seconde).

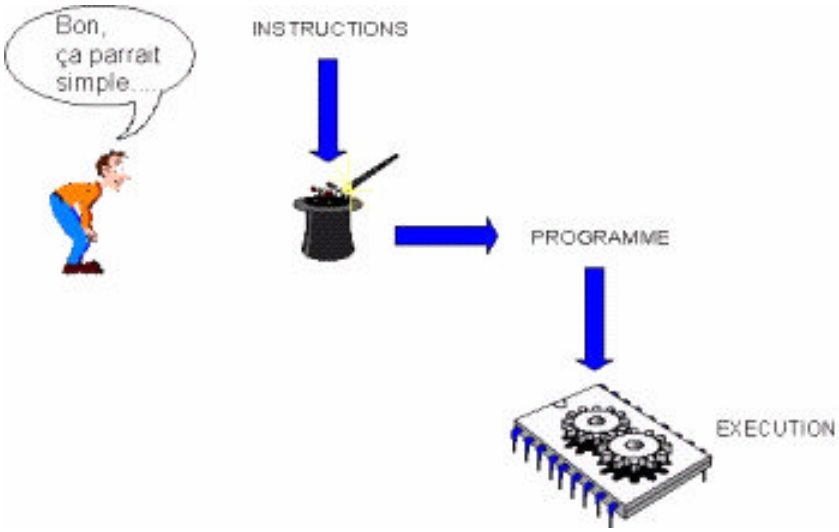


Figure 1 – Les instructions et le programme

Comme vous le savez le microcontrôleur ne travail qu'avec des "0" et des "1" donc il nous faudra un artifice pour convertir les instructions (appelées également mnémoniques) que vous allez sélectionner en langage binaire, ce sera le rôle du compilateur de réaliser cette fonction.

Avant de se lancer dans l'apprentissage de toutes les instructions, faisons un bref rappel sur la façon de réaliser un programme pour le PIC (**figure3**).
Nous allons travailler avec le logiciel MPLAB qui est gracieusement fournit par la société MICROCHIP, vous pouvez télécharger ce logiciel avec de nombreux exemples d'applications sur le site officiel du fabricant:

www.microchip.com

Le logiciel Mplab permet de saisir via un éditeur de texte le source d'un programme tel que nous allons en créer, puis de compiler celui-ci afin de le charger dans la mémoire du PIC concerné.
Une fois le fichier source compilé il va falloir le transférer dans la mémoire du PIC, pour cela il faut un programmateur, de nombreux montages de programmateur de PIC ont été publiés dans notre revue, d'autre part vous pouvez également trouver des kits chez nos annonceurs.
Un logiciel est également nécessaire pour le transfert du fichier compilé vers le PIC, le plus populaire est ICPROG, vous pouvez télécharger ce produit sur le site :

www.ic-prog.com

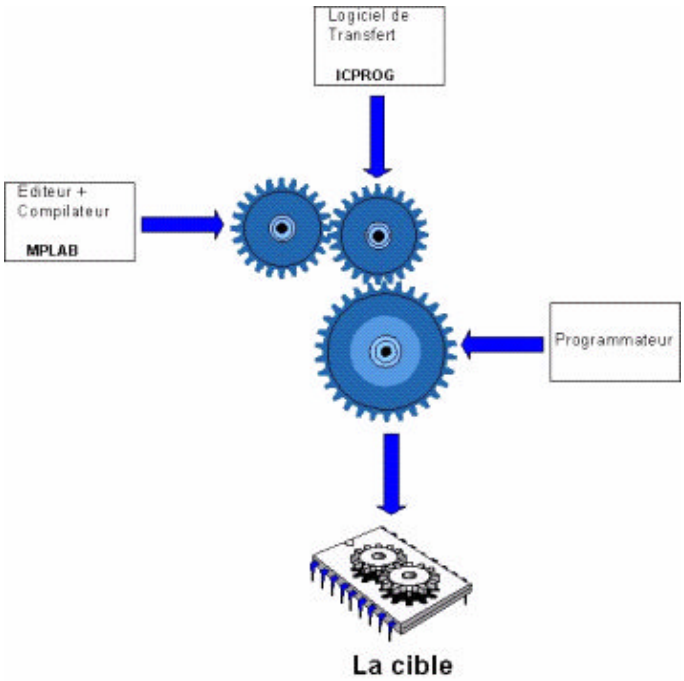


Figure 3 Cheminement de la programmation

Nous allons maintenant détailler les instructions en donnant un exemple typique de programmation pour chaque mnémonique.

Les instructions pourront avoir plusieurs écritures telles que :

Instruction	(exemple d'écriture: *RETURN*)
Instruction + valeur	(exemple d'écriture: *MOVLW 05*)
Instruction + source, destination	(exemple d'écriture: *ADDWF reg1,w*)
Instruction + source, numéro bit du registre	(exemple d'écriture: *BSF reg1,3*)

Classement des instructions du PIC

Pour simplifier nous allons classer arbitrairement (**figure 2**) les trente cinq instructions selon quatre critères qui sont :

- Instructions de branchement et de contrôle
- Instructions relatives au registre de travail w
- Instructions relatives aux registres "file" d'utilisation spéciale ou générale
-
- Instructions relatives au traitement sur un bit spécifique d'un registre

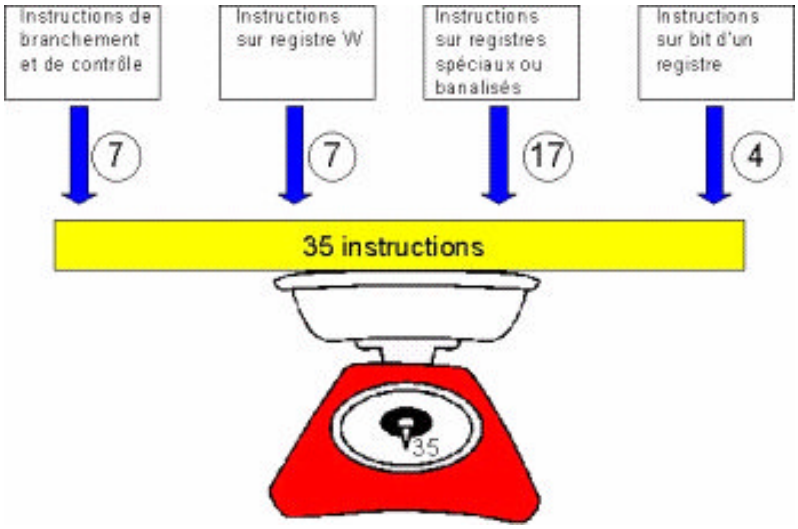


Figure 2 – Le classement des 35 instructions

Liste des instructions du PIC 16F84

Mnémonique	Fonction	Flags		
		Z	DC	C
ADDLW	registre w + valeur -> w	X	X	X
ADDWF	w + contenu registre f -> d	X	X	X
ANDLW	w and valeur -> w	X		
ANDWF	w and contenu registre f -> d	X		
BCF	0 -> (f (N° bit))	----------		
BSF	1 -> (f (N° bit))	----------		
BTFSC	saute la prochaine instr si N° bit registre f =0	----------		
BTFSS	saute la prochaine instr si N° bit registre f =1	----------		
CALL	PC+1 -> pile @saut -> PC	----------		
CLRF	0 -> registre f	X		
CLRW	0 -> registre w	X		
CLRWDT	RAZ temporisateur chien de garde	To		Pd
COMF	complément du registre f --> d	X		
DECF	contenu registre - 1 -> d	X		
DECFSZ	idem DECF et saute prochaine instruction si registre f = 0	----------		
GOTO	branchement à une adresse	----------		
INCF	contenu registre f + 1 -> d	X		
INCFSZ	idem INCF et saute prochaine instruction si registre f= 0	----------		
IORLW	registre w ou valeur -> w	X		
IORWF	registre w ou valeur registre f -> d	X		
MOVF	copie contenu registre f dans registre d	X		
MOVLW	chargement d'une valeur dans le registre w	----------		
MOVWF	copie contenu registre w dans registre f	----------		
NOP	aucune opération	----------		
RETFIE	retour d'interruption	----------		
RETLW	retour de sous-programme avec chargement valeur dans w	----------		
RETURN	retour de sous - programme	----------		
RLF	rotation à gauche de f au travers de la carry (bit c) -> d	----------		
RRF	rotation à droite de f au travers de la carry (bit c) -> d	----------		
SLEEP	mise en mode veille du microcontrôleur	----------		
SUBLW	valeur - contenu registre w -> w	X	X	X
SUBWF	contenu registre f - contenu registre w -> d	X		X
SWAPF	échange de quartets entre registre f et d	----------		
XORLW	contenu registre w xor valeur -> w	X		
XORWF	contenu registre w xor contenu registre f -> d	X		

Nota : si d=1 alors registre w sélectionné si d=0 alors registre f sélectionné
f = registre banalisé que vous déclarez
w = registre de travail interne du PIC

Tableau 1 Liste des instructions

Détail des instructions

1 - Instructions de branchement et de contrôle

Ces instructions concernent en particulier tous les types de branchement, ainsi que les retours de sous-programme et différents modes de fonctionnement tel que le mode sommeil.

Instruction : **CALL**

Rôle : Cette instruction sert à exécuter un sous programme. Dès que le PIC rencontre cette mnémonique le compteur de programme (CP) se charge avec l'adresse du sous programme à exécuter, auparavant il y a une sauvegarde de l'adresse courante du CP (dans la pile du PIC), en fin de sous programme une deuxième instruction (RETURN) que nous verrons plus loin sert à revenir à la position ou était le programme avant d'être dérouté (**figure 4**).

***Quand doit t'on utiliser* cette instruction:** Cette instruction sera utilisée lorsque dans votre programme vous avez réalisé un sous-programme et que vous voulez y accéder.

Syntaxe : CALL étiquette
Une étiquette correspond à un mot situé en début de sous programme. Sur MPLAB l'étiquette se place en début de ligne. L'étiquette correspond à une adresse dans le programme.

Nombre de cycle(s) d'horloge nécessaires : 2

Exemple d' utilisation: (Nota : les commentaires sont en italique et soulignés)

```
        call tempo              'on appel ici le sous-programme tempo
-
-
-
tempo                           'le sous-programme tempo commence ici
-
-
        return                  L'instruction return permet de revenir au
                                programme dérouté
```

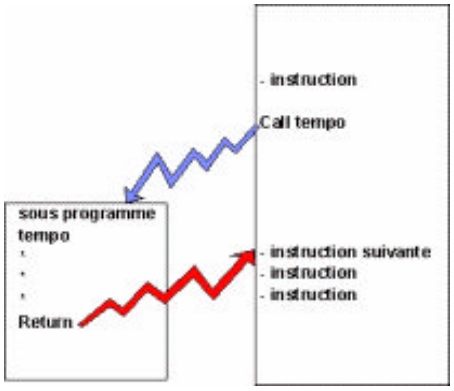


Figure 4 – L'instruction Call

Instruction : **CLRWDT**

Rôle : Cette instruction sert à mettre à zéro le watchdog, de sorte de ne pas effectuer un reset du PIC, nous verrons des exemples d'applications avec le watchdog dans nos programmes.

***Quand doit t' on utiliser* cette instruction:** Cette instruction sera utilisée lorsque vous utiliser le watchdog, celui-ci permet entre autre de vérifier le bon déroulement d'un programme. Si par exemple votre programme doit impérativement appeler une routine régulièrement et pour être sûr que cette routine est appelée, on place en général le reset du chien de garde dans ce sous-programme, ainsi si le programme est planté il ne passe pas par la routine, le watchdog n'est pas réarmé et déclenche alors le reset du microcontrôleur.

Syntaxe : CLRWDT

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation: (Nota : les commentaires sont en italique et soulignés)

```
-
-
        clrwdt          'on réarme ici la temporisation du watchdog
-
-
```

Instruction : **GOTO**

Rôle : Cette instruction sert à brancher le programme vers une autre adresse, le compteur de programme charge l'adresse du saut, attention la différence par rapport à l'instruction call est que l'adresse de retour n'est pas mémorisée dans la pile ce qui fait que le programme continu à l'endroit du saut (**figure 5**). Cette instruction sera utilisée principalement avec un branchement conditionnel qui forcera le saut (goto) si une condition est vraie ou fausse, nous l'utiliserons dans nos futurs programmes.

***Quand doit t' on utiliser* cette instruction:** Cette instruction sera utilisée lorsque vous testez une condition si cette condition est vraie alors on exécute le goto vers un traitement (branchement vers une étiquette) sinon on continue le traitement en cours.

Syntaxe : GOTO étiquette

Nombre de cycle(s) d'horloge nécessaires : 2

Exemple d' utilisation:

```
-
-
        Goto tempo          'on va au sous-programme tempo
-
-
-
tempo                       'le sous-programme tempo commence ici
-
-
```

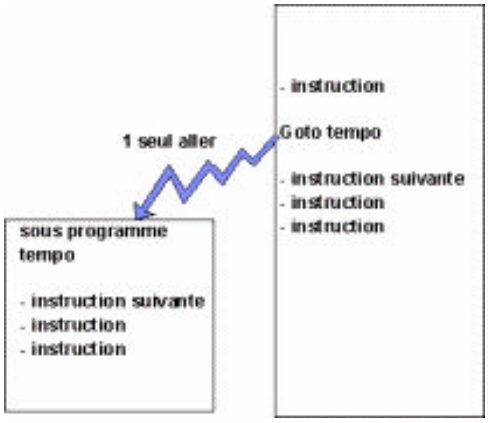


Figure 5 – l'instruction Goto

Instruction : **RETFIE (RETurn From IntErrupt)**

Rôle : Cette instruction doit être utilisée impérativement lorsque votre programme fonctionne sous interruption. Cette instruction se place en fin de sous-programme d'interruption , elle permet de revenir à l'endroit du programme juste avant le déclenchement de l'événement qui à provoqué l'interruption. Nous élaborerons ce genre de programme dont l'intérêt réside dans le fait qu'à n'importe quel endroit de votre programme un événement (une interruption) peut dérouter celui-ci pour

exécuter une routine (un sous-programme) prioritaire, avant de rendre la main (c'est le retour à l'endroit ou le programme était avant l'IT).Notez déjà que l'adresse du sous-programme d'interruption sera toujours l'adresse 4 sur le PIC 16F84, nous y reviendrons.

Quand doit t' on utiliser **cette instruction:** Uniquement si on utilise les interruptions, elle se place en fin de sous-programme

Syntaxe : RETFIE

Nombre de cycle(s) d'horloge nécessaires : 2

Exemple d' utilisation:

```
-
-
Sp_IT              'c'est une étiquette donnée au sous-programme d'interruption
-
-
-
        RETFIE                          'retour d'interruption
```

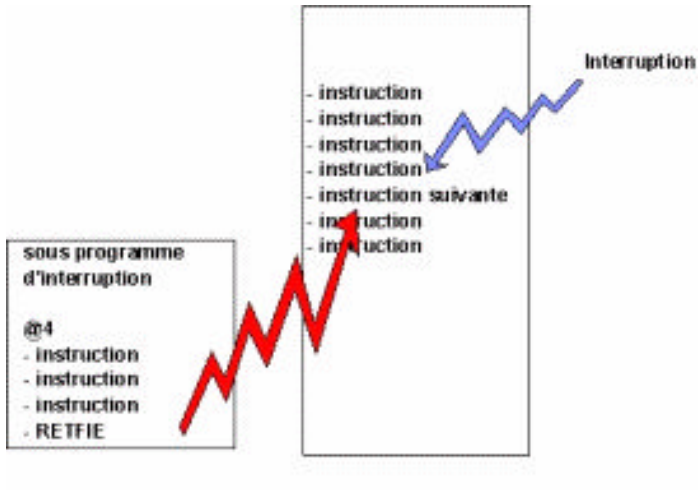


Figure 6 – l'instruction RETFIE

Instruction : **RETLW (RETurn with Literral in W)**

Rôle : Cette instruction est identique à l'instruction RETURN, elle sert en fin de sous-programme à faire revenir le compteur de programme à l'instruction suivant le saut à la routine (sous-programme). L'instruction RETLW permet en plus de charger une valeur dans le registre de travail w au moment ou elle est active.

***Quand doit t' on utiliser* cette instruction:** Dans un programme comportant un sous-programme et lorsque l'on souhaite sortir de la routine avec une valeur qui peut bien sûr dépendre du traitement dans le sous-programme.

Syntaxe : RETLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 2

Exemple d' utilisation:

```
        call tempo          'on appel ici le sous-programme tempo

tempo                       'le sous-programme tempo commence ici

        RETLW 05            L'instruction RETLW permet de revenir au
                            programme dérouté et le registre w contient la
                            valeur 5
```

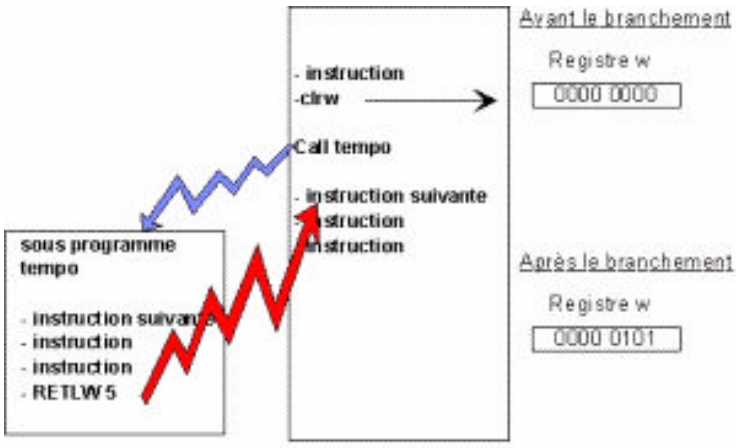


Figure 7 – l'instruction RETLW

Instruction : **SLEEP**

Rôle : Cette instruction met le microcontrôleur en mode sommeil ce qui signifie que l'oscillateur interne est bloqué et que le programme est arrêté. Pour sortir de ce mode il faut impérativement une interruption, un signal watchdog ou un reset du PIC.

***Quand doit t' on utiliser* cette instruction:** Dans un programme qui ne travail par exemple que sur interruption, on peut passer en mode basse consommation avec l'instruction sleep pour économiser la source d'énergie.

Syntaxe : SLEEP

Nombre de cycle(s) d'horloge nécessaires : 2

Exemple d' utilisation:

```
        - Instruction
        - Instruction
-
SLEEP                       'le PIC passe en mode sommeil
-
```

2 - Instructions concernant le registre de travail w

Ces instructions concernent tous les types de traitement appliqués sur le registre de travail w.

Instruction : **ADDLW**

Rôle : Cette instruction additionne le contenu du registre de travail w avec une valeur immédiate, le résultat est stocké dans w.

***Quand doit t' on utiliser* cette instruction:** Sur le PIC on doit souvent pour accéder à un registre banalisé (un registre banalisé peut être par exemple une variable que vous avez déclaré, il peut y en avoir 68 au maximum, c'est en fait la taille de la RAM) passer par le registre w. Si on veut additionner une valeur avec un

registre de son choix on utilisera l'instruction ADDLW puis on transférera le contenu dans le registre banalisé.

Syntaxe : ADDLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

```
- CLRW          'on fait une raz du registre w
- ADDLW 06      'on additionne le contenu de w avec 6 donc w=6
```

Instruction : **ANDLW**

Rôle : Cette instruction fait un "et logique" entre le contenu du registre de travail w et la valeur immédiate (opérande) , le résultat est stocké dans w.

***Quand doit t' on utiliser* cette instruction:** On utilise souvent le "ET LOGIQUE" lorsque l'on veut sélectionner un ou plusieurs bits d'un registre, on parle alors de masque.

Syntaxe : ANDLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

```
- CLRW          'on fait une raz du registre w
- ADDLW 06      'on additionne le contenu de w avec 6 donc w=6
- ANDLW 03            et logique entre 6 et 3 -> 0110  and   0011 donc w=
                0010 soit 2
```

Instruction : **IORLW**

Rôle : Cette instruction fait un "ou logique" entre le contenu du registre de travail w et la valeur immédiate (opérande) , le résultat est stocké dans w.

***Quand doit t' on utiliser* cette instruction:** On utilise cette instruction pour par exemple avoir deux entrées de commandes pour une sortie, nous ne

manquerons pas d'utiliser cette instruction.

Syntaxe : IORLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

- CLRW	*'on fait une raz du registre w*
- ADDLW 06	*'on additionne le contenu de w avec 6 donc w=6*
- ANDLW 03	*et logique entre 6 et 3 -> 0110 and 0011 donc w= 0010 soit 2*
- IORLW 01	*ou logique entre le contenu de w avec 1 -> 0010 ou 0001 w=0011*

Instruction : **MOVLW**

Rôle : Cette instruction charge le contenu du registre w avec une valeur immédiate, l'ancien contenu est alors écrasé.

***Quand doit t' on utiliser* cette instruction:** Cette instruction est énormément utilisée, car le registre w est comme une plaque tournante du PIC, c'est par lui que l'on accédera à tous nos registres banalisés (variables) que l'on aura déclaré à l'aide de la syntaxe (nous reviendrons sur toutes les directives d'assemblage) EQU.

Syntaxe : MOVLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

Compteur EQU 15	*'on déclare une variable appelée 'compteur' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 7	*'on met le registre w à 7*
- MOVWF Compteur	*'on transfert le contenu de w (ici c'est la valeur 7) dans la variable (ou registre banalisé) nommée Compteur*

Instruction : **SUBLW**

Rôle : Cette instruction soustrait le contenu du registre de travail w avec une valeur immédiate, le résultat est stocké dans w.

***Quand doit t' on utiliser* cette instruction:** Sur le PIC on doit souvent pour accéder à un registre banalisé (un registre banalisé peut être par exemple une variable que vous avez déclaré, il peut y en avoir 68 au maximum, c'est en fait la taille de la RAM) passer par le registre w. Si on veut soustraire une valeur du registre de son choix on utilisera l'instruction SUBLW puis on transférera le contenu dans le registre banalisé.

Syntaxe : SUBLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

```
mavariable EQU 16          'on déclare une variable appelée 'mavariable' qui
                           se trouve à l'adresse 16 en RAM (de 12 à 79 soit 68
                           variables utilisables)
 - CLRW                    'on fait une raz du registre w
 - ADDLW 06                'on additionne le contenu de w avec 6 donc w=6
 - SUBLW 01                'on sosutrait le contenu de w avec 1 donc w=5

-- MOVWF mavariable        'on transfert le contenu de w (ici c'est la valeur 5)
                           dans la variable (ou registre banalisé) nommée
                           mavariable
```

Instruction : **XORLW**

Rôle : Cette instruction fait un "ou exclusif " entre le contenu du registre de travail w et la valeur immédiate (opérande) , le résultat est stocké dans w.

***Quand doit t' on utiliser* cette instruction:** On utilise cette instruction pour par exemple comparer le contenu d'un registre avec une valeur, en effet avec un ou exclusif le résultat dans le registre w sera égal à 0000 0000 si et seulement si tous les bits de la comparaison sont identiques (exemple w= 0111 0111 et la valeur pour le ou exclusif = 0111 0111)

Syntaxe : XORLW + valeur

Nombre de cycle(s) d'horloge nécessaires : 1

Exemple d' utilisation:

```
- CLRW          'on fait une raz du registre w
- ADDLW 06      'on additionne le contenu de w avec 6 donc w=6
- ANDLW 05            et logique entre 6 et 3 -> 0110  and   0101 donc w=
                0100 soit 4
- XORLW 02            ou exclusif entre le contenu de w avec 2 -> 0100
                xor  0010 w=0110
```

Instruction : **CLRW**

Rôle : Cette instruction met le contenu du registre de travail w à zéro

Quand doit t' on utiliser **cette instruction:** Etant donné que l'on utilise souvent le registre de travail w il est dans certain cas nécessaire de partir avec une valeur nulle dans ce registre.

Syntaxe : CLRW

Nombre de cycle(s) d'horloge nécessaires : 1

```
- CLRW          'on fait une raz du registre w
```

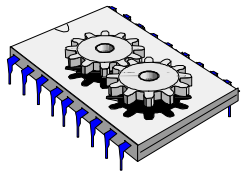
Pour vous laisser le temps de réfléchir sur les instructions que nous venons de voir et avant notre prochaine leçon qui terminera la suite voici un exemple type de programme. Dans ce source on utilise une bonne partie des instructions que nous venons de voir. Le rôle de ce programme est de réaliser un compteur binaire 8 bits et de l'afficher sur les leds du port B.

Exemple 1 – Programme type

Pour conclure ce chapitre

```
;-------- Exemple d'application avec un PIC : Un compteur binaire ---------
; Titre : Compteur binaire
; Date : 30 juillet 2004
; Auteur : Electronique Pratique
; PIC utilisé : PIC 16 F 84
; On réalise un compteur binaire sur les broches RB0 à RB7 d'un PIC 16 F 84 le quartz utilisé
; est de 4 Mhz , on effectue une tempo environ égale à 0.2 seconde.

;-------- Directive d'assemblage pour MPLAB ---------
list p=16f84A                ; type de PIC utilisé
#include p16f84A.inc          ; fichier d'adressage necessaire au compilateur
__config H3FF9               ; déclaration de l'oscillateur quartz et pas de code protect

;-------- Définition des constantes ---------
#define inter0 0   ; bouton marche

;-------- Définition des registres temporaires ---------
retard1    EQU    0x0C    ; le registre temporaire retard1 se trouve à l'adresse 0C
retard2    EQU    0x0D    ; le registre temporaire retard2 se trouve à l'adresse 0D
memo       EQU    0x0E    ; le registre memo tampon se trouve à l'adresse 0E

;-------- Init des ports A et B ---------
 ORG 0
 bsf STATUS,5                ; on met à 1 le 5eme bit du registre status pour accéder
                             ; à la 2eme page mémoire ( pour trisa et trisb )
 MOVLW B'00000000'           ;on met 00 dans le registre W
 MOVWF TRISB                 ; on met 00 dans le port B il est programmé en sortie
 MOVLW 0x1F                  ; on met 1F dans le registre W
 MOVWF TRISA                 ; on met 00 dans le port A il est programmé en entrée
 bcf STATUS,5                ; on remet à 0 le 5eme bit du registre status pour accéder
                             ; à la 1ere page mémoire

;-------- Programme principal ---------
Main

    btfss PORTA,inter0       ; interrupteur 0 ( marche ) appuyé ? si oui on continu sinon
    goto Main                ; on va à l'étiquette Main

    MOVLW  0xFF              ; on met 255 dans le registre W
    MOVWF  retard1           ; on charge retard1 avec 255 ( FFh contenu du registre W )
    MOVLW  0xFF              ; on met 255 dans le registre W
    MOVWF  retard2           ; on charge retard2 avec 255 ( FFh contenu du registre W )
    MOVF  memo, W            ; on met memo dans W
    MOVWF  PORTB             ; on met W sur le port B ( leds )
    CALL  tempo              ; on appel la temporisation
    MOVLW  0x01              ; on met 1 dans le registre W
    ADDWF  memo, F           ; on additionne memo + 1
    GOTO  Main               ; retour au début du programme

;-------- Programme de temporisation ( 0.2 s ) ---------

tempo
    DECFSZ retard1,F         ; on décrémente retard1 et on saute la prochaine instruction si
    GOTO  tempo              ; le registre retard1 = 0 sinon retour à tempo
    MOVLW 0xFF               ; on met 255 dans le registre W
    MOVWF retard1            ; on charge retard1 avec 255 ( FFh contenu du registre W )
    DECFSZ retard2,F         ; on décrémente retard1 et on saute la prochaine instruction si
    GOTO  tempo              ; le registre retard1 = 0 sinon retour à tempo
    RETURN                   ; retour au programme principal après l'instruction CALL

    END                      ; fin du programme
```

Avec cette quatrième partie nous avons passé en revue et détaillé une partie des instructions des PIC de la série 16Fxx et 16Fxxx, dans le prochain numéro nous terminerons cette présentation et nous détaillerons notre premier programme en utilisant le logiciel MPLAB.



A la découverte des microcontrôleurs PIC
Cinquième partie

Nous allons terminer dans ce numéro les dernières instructions qu'il nous restait à détailler, puis dans notre prochaine leçon nous analyserons un programme afin de se familiariser avec tout le jeu d'instructions que nous avons vu.

Les instructions suite… et fin

3 - Instructions concernant les registres banalisés

Ces instructions concernent tous les types de traitement appliqués sur un registre banalisé, un registre banalisé peut être par exemple une variable que vous avez déclaré, il peut y en avoir 68 au maximum, c'est en fait la taille de la RAM du PIC.
Chaque registre banalisé (ou variable que vous déclarez) possède une taille de 8 bits (1 octet).

Instruction : **ADDWF** *(ADD W and F)*

Rôle : Cette instruction additionne le contenu du registre de travail w avec le contenu d'un registre banalisé, le résultat est stocké dans w si l'opérande (voir syntaxe de la commande) d = 0 sinon le résultat sera stocké dans le registre banalisé

Quand doit t' on utiliser **cette instruction:** lorsque l'on on veut additionner une valeur avec un registre banalisé.

Syntaxe : ADDWF + f,d

f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

#define f 1	'la variable f vaut 1
#define w 0	'la variable w vaut 0
monregistre EQU 12	*'on déclare une variable appelée 'monregistre' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRW	*'on fait une raz du registre w*
- ADDLW 01	*'on additionne le contenu de w avec 1 donc w=1*
- ADDWF monregistre,f	*'on additionne w et le contenu du registre "monregistre" , le résultat est stocké dans "monregistre" car l'opérande "d" ici vaut 1 (f est déclarée à 1) en début de programme)*
- ADDWF monregistre,w	*'on additionne w et le contenu du registre "monregistre" , le résultat est stocké dans "w" car l'opérande "d" ici vaut 0 (w est déclarée à 0 en début de programme)*

-

Instruction : **ANDWF (AND W and f)**

Rôle : Cette instruction fait un "et logique" entre le contenu du registre de travail w et le contenu du registre spécifié f, selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** On utilise souvent le "ET LOGIQUE" lorsque l'on veut sélectionner un ou plusieurs bits d'un registre, on parle alors de masque.

Syntaxe : ANDWF f,d

f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

#define f 1	'la variable f vaut 1
monregistre EQU 12	*'on déclare une variable appelée 'monregistre' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRW	*'on fait une raz du registre w*
- ADDLW 01	*'on additionne le contenu de w avec 1 donc w=1*
- ANDWF monregistre,f	*'on fait un "et logique" entre le contenu de w et le contenu du registre "monregistre" , le résultat est stocké dans "monregistre" car l'opérande "d" ici vaut 1 (f est déclarée à 1) en début de programme)*

Instruction : **CLRF (Clear f)**

Rôle : Cette instruction met le contenu du registre spécifié d'adresse f à zéro

***Quand doit t' on utiliser* cette instruction:** Etant donné que l'on utilise souvent les registres banalisés il est dans certain cas nécessaire de partir avec une valeur nulle dans ce registre.

Syntaxe : CLRF f
f= adresse du registre banalisé

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRF tempo	*'on fait une raz du registre tempo déclaré ci-dessus*

Instruction : **COMF (Complement f)**

Rôle : Cette instruction fait une opération de complément sur le contenu du registre spécifié d'adresse f , en fait les 0 sont remplacés par des 1 et vice versa , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** lorsque l'on doit réaliser une opération de complément (inversion des tous les bits) sur un registre.

Syntaxe : COMF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d'utilisation:

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0x55	*'on met le registre w à 55 (0101 0101)*
- MOVWF tempo	*'on transfert le contenu de w (ici c'est la valeur 55) dans la variable (ou registre banalisé) nommée tempo*
- COMF tempo,1	*'on complémente le contenu de tempo (ici c'est la valeur 55) et on stocke le résultat AA (1010 1010) dans la variable (ou registre banalisé) nommée tempo puisque l'opérande d = 1*

Instruction : **DECF (Decrement f)**

Rôle : Cette instruction fait une décrémentation (contenu – 1) sur la valeur du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** lorsque l'on doit réaliser une soustraction unitaire sur un registre, pour réaliser une temporisation ou un décomptage par exemple

Syntaxe : DECF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Dans l'exemple qui suit nous allons réaliser une temporisation et pour se faire nous utilisons l'instruction BTFSS (qui est détaillée plus loin) ainsi que le flag Z du registre STATUS (voir les premières leçons) ce flag (ou bit) indique s'il vaut 1 que le résultat de la dernière opération effectuée vaut 0 et vice versa.

Exemple d' utilisation:

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0x55	*'on met le registre w à 55 (0101 0101)*
- MOVWF tempo	*'on transfert le contenu de w (ici c'est la valeur 55) dans la variable (ou registre banalisé) nommée tempo*
loop	
- DECF tempo,1	*'on décrémente le contenu de tempo (valeur initiale = 55) et on stocke le résultat dans la variable (ou registre banalisé) nommée tempo puisque l'opérande d = 1*
-BTFSS STATUS,2	*'On test le bit Z (zéro) du registre status, si Z=1 alors on saute la prochaine instruction(goto loop) sinon si le flag Z = 0 alors on revient à l'étiquette 'loop et on redécrémente le registre tempo ceci permet de faire une boucle (qui est en fait une temporisation), cette boucle s'effectuera tant que le contenu du registre tempo ne vaut pas zéro*
- GOTO loop	*'retour à l'étiquette 'loop'*

Instruction : **DECFSZ (Decrement f , skip if zéro)**

Rôle : Cette instruction est double, elle fait une décrémentation (contenu – 1 comme l'instruction DECF) sur la valeur du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1), puis teste ensuite si le contenu du registre spécifié vaut zéro et dans ce cas saute la prochaine instruction.

***Quand doit t' on utiliser* cette instruction:** Cette instruction est souvent utilisée pour réaliser une temporisation ou un décomptage par exemple

Syntaxe : DECFSZ f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1 ou 2 selon résultat du test

Exemple d' utilisation:
Reprenons l'exemple de la temporisation réalisée avec l'instruction DECF, nous voyons ici qu'il y a une ligne en moins pour réaliser le même programme.

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0x55	*'on met le registre w à 55 (0101 0101)*
- MOVWF tempo	*'on transfert le contenu de w (ici c'est la valeur 55) dans la variable (ou registre banalisé) nommée tempo*
delay	
- DECFSZ tempo,1	*'on décrémente le contenu de tempo (valeur initiale = 55) et on stocke le résultat dans la variable (ou registre banalisé) nommée tempo puisque l'opérande d = 1puis on saute la prochaine instruction (goto delay) si le contenu de tempo vaut zéro, cela permet de faire une boucle comme dans l'exemple avec DECF*

- GOTO delay *'retour à l'étiquette 'delay*

Instruction : **INCF (INCrement f)**

Rôle : Cette instruction fait une incrémentation (contenu + 1) sur la valeur du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** lorsque l'on doit réaliser une addition unitaire sur un registre, pour réaliser un compteur ,.on peut ainsi déterminer par exemple le nombre de défauts constatés sur une machine

Syntaxe : INCF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0x55	*'on met le registre w à 55 (0101 0101)*
- MOVWF tempo	*'on transfert le contenu de w (ici c'est la valeur 55) dans la variable (ou registre banalisé) nommée tempo*
- INCF tempo,1	*'on incrémente le contenu de tempo qui passe à 56 (hexa) et on stocke le résultat dans la variable (ou registre banalisé) nommée tempo puisque l'opérande d = 1*

Instruction : **INCFSZ (Increment f , skip if zéro)**

Rôle : Cette instruction est semblable à DECFSZ , elle fait une incrémentation (contenu + 1 comme l'instruction INCF) sur la valeur du registre spécifié d'adresse f ,

selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1), puis teste ensuite si le contenu du registre vaut zéro et dans ce cas saute la prochaine instruction.

***Quand doit t' on utiliser* cette instruction:** Cette instruction est souvent utilisée pour réaliser une temporisation ou un comptage par exemple

Syntaxe : INCFSZ f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1 ou 2 selon résultat du test

Exemple d' utilisation: Dans cette exemple nous allons faire une boucle (ou une temporisation) de 256 tops d'horloge, puisque quand le contenu du registre (8 bits) est à 255 et qu'on lui ajoute + 1 il revient à zéro.

tempo EQU 12	*'on déclare une variable appelée 'tempo' qui se trouve à l'adresse 12 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRF tempo	*'on fait une raz du contenu du registre banalisé nommée tempo*
delay	
- INCFSZ tempo,1	*'on incrémente le contenu de tempo (valeur initiale = 0) et on stocke le résultat dans la variable (ou registre banalisé) nommée tempo puisque l'opérande d = 1puis on saute la prochaine instruction (goto delay) si le contenu de tempo vaut zéro, cela permet de faire une boucle comme de 256 tops d'horloge*
- GOTO delay	*'retour à l'étiquette 'delay*

Instruction : **IORWF (Inclusive OR W with F)**

Rôle : Cette instruction fait un "ou logique" entre le contenu du registre de travail w et le contenu du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** On utilise cette instruction pour réaliser par exemple un ou logique entre le registre w et une valeur qui n'est pas figée et qui dépend du contenu d'un registre.

Syntaxe : IORLW f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

tampon EQU 15	*'on déclare une variable appelée 'tampon' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRW	*'on fait une raz du registre w*
- IORWF tampon,0	'*ou logique entre le contenu de w et le contenu du registre nommé 'tampon' le résultat est stocké dans w puisque l'opérande d = 0, en fait c'est une copie du registre 'tampon' dans le registre w puisque qu'auparavant nous avons fais une raz de w (CLRW) -> 0 ou contenu registre tampon = registre tampon*

Instruction : **XORWF (eXclusive OR W with F)**

Rôle : Cette instruction fait un "ou exclusif " entre le contenu du registre de travail w et le contenu du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** Souvent pour comparer le contenu de deux registres on utilise un ou exclusif, si le résultat de l'opération vaut 0 alors les contenus des registres sont identiques

Syntaxe : XORLW f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

tampon EQU 15	*'on déclare une variable appelée 'tampon' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
- XORWF tampon,0	*ou exclusif entre le contenu de w et le contenu du registre nommé 'tampon' le résultat est stocké dans w puisque l'opérande d = 0, il suffit ensuite de tester le contenu du registre w, si celui-ci vaut zéro alors les deux registres (w et tampon) étaient identiques avant la comparaison.*

Instruction : **MOVF (Move f)**

Rôle : Cette instruction copie le contenu du registre spécifié d'adresse f sur lui même ou bien dans le registre w (selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1)), le fait de copier un registre sur lui – même peut paraître absurde mais cela permet entre autre de tester le flag z (zéro) du registre d'état et ainsi permettre de savoir si le contenu du registre vaut zéro.

***Quand doit t' on utiliser* cette instruction:** Une utilisation peut être envisagée lorsque l'on veut copier un registre banalisé vers un deuxième registre du même type, dans un premier temps on copie le registre dans le registre de travail w (MOVF registre , w) puis ensuite on transfert le contenu du registre de travail w vers le registre destination avec l'instruction MOVWF (que l'on verra plus loin).

Syntaxe : MOVF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

Compteur EQU 15	*'on déclare une variable appelée 'compteur' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
recharge EQU 16	*'on déclare une variable appelée 'recharge' qui se trouve à l'adresse 16 en RAM (de 12 à 79 soit 68 variables utilisables)*
#define w, 0	*'Pour le compilateur la lettre w vaudra 0 dans le reste du programme*
- MOVF recharge, w	*'on copie le contenu du registre recharge dans le registre de travail w*
- MOVWF Compteur	*'on transfert le contenu de w (c'est le contenu du registre recharge) dans la variable (ou registre banalisé) nommée Compteur*

Instruction : **MOVWF (Move w to f)**

Rôle : Cette instruction copie le contenu du registre de travail w vers le registre spécifié d'adresse f

***Quand doit t' on utiliser* cette instruction:** Cette instruction sera souvent utilisée c'est elle qui permettra de charger une valeur dans un registre. Encore une fois nous voyons que le registre de travail w a une place importante dans le PIC.

Syntaxe : MOVWF f
f= adresse du registre banalisé

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation: Voir exemple précédent (instruction MOVF)

Instruction : **NOP (No oPération)**

Rôle : Cette instruction ne réalise aucune opération.

Quand doit t' on utiliser **cette instruction:** Cette instruction est particulièrement utilisée lorsque l'on veut introduire un retard dans un programme, tout en sachant que l'instruction NOP prend un cycle machine.

Syntaxe : NOP

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

Notre application est cadencée avec un quartz de 4 MHz, il faut savoir qu'en interne le PIC divise la fréquence du quartz par 4 ce qui fait une horloge à 1 MHz, donc un cycle machine dure 1 / 1 10^{-6} soit 1 micro seconde (1 µS). Nous souhaitons avoir une temporisation (ou un retard) dans une boucle de programme de 5 µS, pour ce faire nous allons insérer 5 instructions de type NOP.
Toutefois pour des temporisations très longues il est bien certain que l'on n'utilisera pas ce genre d'artifice, il faudra réaliser une temporisation avec le décomptage d'un registre par exemple.

```
mavariable EQU 16        'on déclare une variable appelée  'mavariable' qui se
                         trouve à l'adresse 16 en RAM (de 12 à 79 soit 68
                         variables utilisables
)
 - CLRW                  'on fait une raz du registre w

 - ADDLW 06                    'on additionne le contenu de w avec 6 donc w=6

 -    NOP                 'on introduit ici un retard de 5 µS
 -    NOP
 -    NOP
 -    NOP
 -    NOP
 -
 -- MOVWF mavariable      'on transfert le contenu de w (ici c'est la valeur 6) dans la
                         variable (ou registre banalisé) nommée mavariable
```

Instruction : **RLF (Rotate Left f throught carry)**

Rôle : Cette instruction exécute une rotation à gauche du contenu du registre spécifié d'adresse f . Le bit situé le plus à gauche est transféré dans la carry et la valeur précédente de la carry est transférée dans le bit 0 du registre .(selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

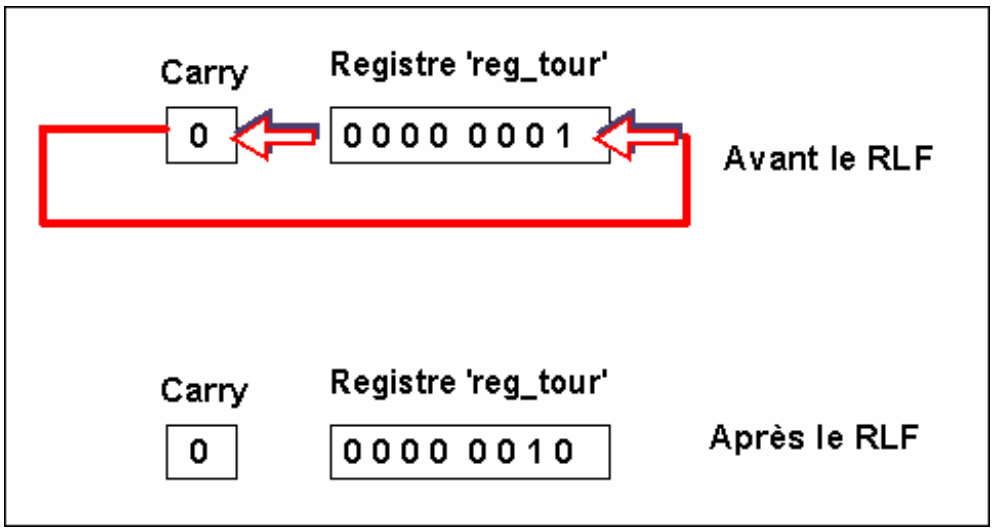
***Quand doit t' on utiliser* cette instruction:** Il y a de nombreuses applications pour cette instruction, allant d'un simple chenillard à une multiplication par 2 par exemple.

Syntaxe : RLF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

reg_tour EQU 15	*'on déclare une variable appelée 'reg_tour' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
#define f, 1	*'Pour le compilateur la lettre f vaudra 1 dans le reste du programme*
- CLRW	*'on fait une raz du registre w*
- ADDLW 01	*'on additionne le contenu de w avec 1 donc w=1*
- MOVWF reg_tour	*'on transfert le contenu de w (c'est le contenu du registre recharge) dans la variable (ou registre banalisé) nommée reg_tour*
- RLF reg_tour,f	*'On réalise une rotation à gauche du contenu du registre Nommé 'reg_tour' . Si on considère que la carry avant le RLF était positionnée à 0, la valeur du registre 'reg_tour' au départ valait 1 après la rotation il vaut 2*



Instruction : **RRF (Rotate right f throught carry)**

Rôle : Cette instruction exécute une rotation à droite du contenu du registre spécifié d'adresse f . Le bit situé le plus à gauche est transféré dans la carry et la valeur précédente de la carry est transférée dans le bit 0 du registre .(selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

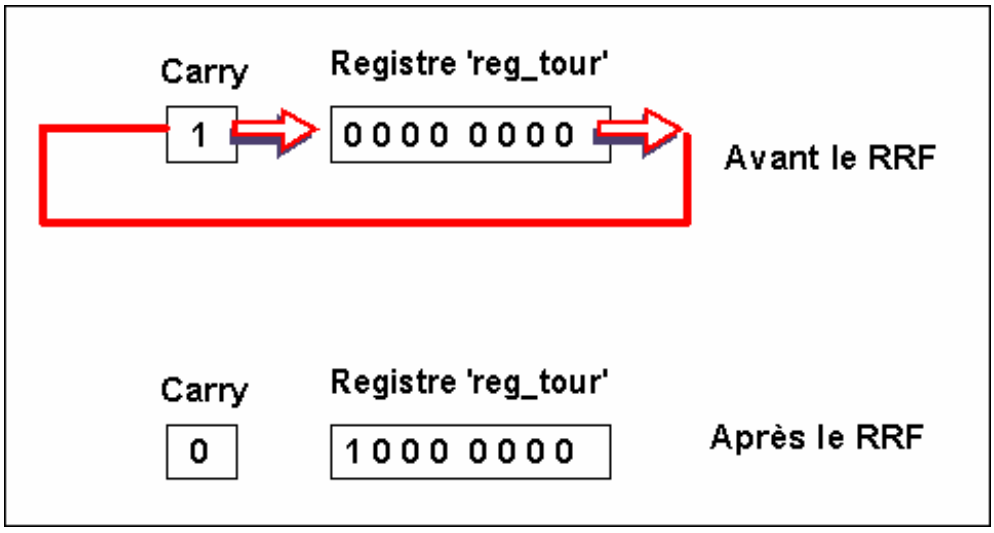
***Quand doit t' on utiliser* cette instruction:** Il y a de nombreuses applications pour cette instruction, allant d'un simple chenillard à une division par 2 par exemple.

Syntaxe : RRF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

reg_tour EQU 15	*'on déclare une variable appelée 'reg_tour' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
#define f, 1	*'Pour le compilateur la lettre f vaudra 1 dans le reste du programme*
- MOVLW 0xFF	*'on met le registre w à 255 ou FF en héxa (1111 1111)*
- ADDLW 01	*'on additionne le contenu de w avec 1 donc w=w+1(la carry est positionnée et w=0)*
- MOVWF reg_tour	*'on transfert le contenu de w (c'est le contenu du registre recharge) dans la variable (ou registre banalisé) nommée reg_tour*
- RRF reg_tour,f	*'On réalise une rotation à droite du contenu du registre Nommé 'reg_tour' ., la valeur du registre 'reg_tour' au départ valait 0 après la rotation il vaut 80 héxa (1000 0000)*



Instruction : **SUBWF (SUBstract W from F)**

Rôle : Cette instruction exécute une soustraction entre le contenu du registre spécifié d'adresse f et le contenu du registre w , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** On utilise cette instruction pour réaliser par exemple une soustraction conditionnelle, la valeur à soustraire est dynamique car est peut évoluer, c'est en fait le contenu d'un registre.

Syntaxe : SUBWF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

mavariable EQU 16	*'on déclare une variable appelée 'mavariable' qui se trouve à l'adresse 16 en RAM (de 12 à 79 soit 68 variables utilisables)*
- CLRW	*'on fait une raz du registre w*
- ADDLW 06	*'on additionne le contenu de w avec 6 donc w=6*
- SUBWF mavariable,0	*'on soustrait le contenu du registre 'mavariable' avec le contenu de w, le résultat est stocké dans w car l'opérande d=0*

Instruction : **SWAPF (SWAP nibbles in F)**

Rôle : Cette instruction permet d'échanger les quatre bits de poids faible par les quatre bits de poids fort du contenu du registre spécifié d'adresse f , selon l'opérande d, le résultat est soit stocké dans w (d=0), soit stocké dans le registre f (d=1).

***Quand doit t' on utiliser* cette instruction:** On utilise souvent cette instruction lorsque l'on travail sous interruption afin de sauvegarder le registre STATUS (registre d'état) car l'instruction SWAPF à la particularité de n'affecter aucun flag du registre d'état.

Syntaxe : SWAPF f,d
f= adresse du registre banalisé
d= direction pour le stockage du résultat (0 = résultat dans W , 1 = résultat dans le registre banalisé dont l'adresse est f)

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

- CLRW	*'on fait une raz du registre w*
- SWAPF STATUS,0	*'on sauvegarde le registre STATUS (registres d'état qui contient tous les flags) dans le registre de travail w puisque l'opérande d=0*

4 - Instructions sur un bit d'un registre

Instruction : **BCF (Bit Clear f)**

Rôle : Cette instruction permet de faire une RAZ du bit indiqué (0 à 7) du registre spécifié d'adresse f .

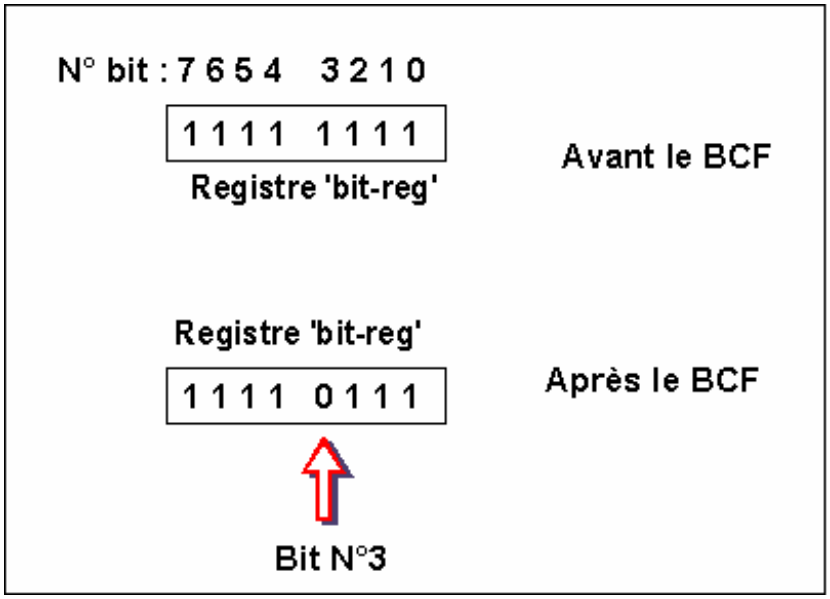
***Quand doit t' on utiliser* cette instruction:** Cette instruction est très utilisée elle permet de travailler rapidement sur un bit d'un registre sans modifier le reste du registre.

Syntaxe : BCF f,b
f= adresse du registre banalisé
b= numéro du bit à remettre à zéro, ce nombre est compris entre 0 et 7.

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

bit-reg EQU 15	*'on déclare une variable appelée bit-reg' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0xFF	*'on met le registre w à 255 ou FF en héxa (1111 1111)*
- MOVWF bit-reg	*'on transfert le contenu de w (FF ou 255) dans la variable (ou registre banalisé) nommée bit-reg*
- BCF bit-reg,3	*'On fait une raz du 3eme bit du registre nommé 'bit-reg' ., la valeur du registre 'bit-reg' au départ valait 1111 1111 après le BCF il vaut 1111 0111 (3eme bit à zéro)*



Instruction : **BSF (Bit Set f)**

Rôle : Cette instruction permet de faire une mise à "1" du bit indiqué (0 à 7) du registre spécifié d'adresse f .

***Quand doit t' on utiliser* cette instruction:** Cette instruction est très utilisée elle permet de forcer rapidement à "1" la valeur d'un bit d'un registre sans modifier le reste du registre.

Syntaxe : BSF f,b
f= adresse du registre banalisé

b= numéro du bit à mettre à "1", ce nombre est compris entre 0 et 7.

Nombre de cycle(s) d'horloge nécessaire(s): 1

Exemple d' utilisation:

bit-reg EQU 15	*'on déclare une variable que l'on appelle bit-reg' qui se trouve à l'adresse 15 en RAM (de 12 à 79 soit 68 variables utilisables)*
- MOVLW 0x00	*'on met le registre w à 0 (0000 0000)*
- MOVWF bit-reg	*'on transfert le contenu de w (00) dans la variable (ou registre banalisé) nommée bit-reg*
- BSF bit-reg,7	*'On fait une mise à "1" du 7eme bit du registre nommé 'bit-reg' ., la valeur du registre 'bit-reg' au départ valait 0000 0000 après le BSF il vaut 1000 0000 (7eme bit à un)*

Instruction : **BTFSC (Bit Test F Skip if Clear)**

Rôle : Cette instruction permet de faire un test du bit indiqué (0 à 7) du registre spécifié d'adresse f et de sauter la prochaine instruction si la valeur du bit vaut "0".

***Quand doit t' on utiliser* cette instruction:** Cette instruction très puissante est utilisée pour faire des branchements conditionnels, on oriente le programme selon la valeur du bit testé. Par exemple on peut tester l'état d'un bit d'un port configuré en entrée, sur lequel est connecté un interrupteur, tant que cet interrupteur est actionné le programme ne démarre pas (interrupteur type "coup de poing" par exemple).

Syntaxe : BTFSC f,b
f= adresse du registre banalisé
b= numéro du bit à remettre à zéro, ce nombre est compris entre 0 et 7.

Nombre de cycle(s) d'horloge nécessaire(s): 1 ou 2 (selon résultat du test)

Exemple d' utilisation: Un interrupteur est connecté sur le port A sur la broche RA0 (bit 0 du port A) on démarre le programme uniquement si l'inter est ouvert .

```
#define inter, 0          'Pour le compilateur le mot inter représente la valeur 0 dans le reste du programme

debut

- BTFSC PORTA,inter      'on teste le bit "0" du port A et si celui-ci vaut "0" alors on saute la prochaine instruction qui reboucle au début du programme

 - goto debut            'on retourne à l'étiquette 'debut' tant que la valeur de l'inter vaut "1"

-   instruction          'suite du programme
-   instruction
```

Instruction : **BTFSS (Bit Test F Skip if Set)**

Rôle : Cette instruction permet de faire un test du bit indiqué (0 à 7) du registre spécifié d'adresse f et de sauter la prochaine instruction si la valeur du bit vaut "1".

***Quand doit t' on utiliser* cette instruction:** Cette instruction très puissante est utilisée pour faire des branchements conditionnels, on oriente le programme selon la valeur du bit testé. Par exemple on peut tester l'état d'un bit d'un port configuré en entrée, sur lequel est connecté un interrupteur, tant que cet interrupteur n'est pas actionné le programme ne démarre pas.

Syntaxe : BTFSC f,b
f= adresse du registre banalisé
b= numéro du bit à remettre à zéro, ce nombre est compris entre 0 et 7.

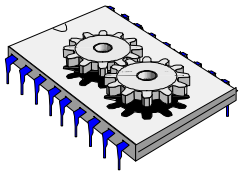
Nombre de cycle(s) d'horloge nécessaire(s): 1 ou 2 (selon résultat du test)

Exemple d' utilisation: Un interrupteur est connecté sur le port A sur la broche RA0 (bit 0 du port A) on démarre le programme uniquement si l'inter est actionné .

```
#define inter, 0          'Pour le compilateur le mot inter représente la valeur 0 dans le reste du programme

debut

- BTFSS PORTA,inter      'on teste le bit "0" du port A et si celui-ci vaut "1" alors on saute la prochaine instruction qui reboucle au début du programme

 - goto debut            'on retourne à l'étiquette 'debut' tant que la valeur de l'inter vaut "0"

-   instruction          'suite du programme
-   instruction
```

Pour conclure ce chapitre

Avec cette cinquième partie nous avons terminé de détailler les instructions des PIC de la série 16Fxx et 16Fxxx, dans le prochain numéro nous détaillerons et réaliserons notre premier programme en utilisant le logiciel MPLAB.



A la découverte des microcontrôleurs PIC
Sixième partie

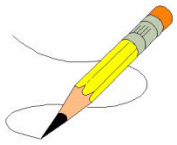
Dans ce numéro nous allons mettre en œuvre les instructions que nous avons détaillés dans les leçons précédentes, pour ce faire nous utiliserons le logiciel MPLAB de fourniture MICROCHIP™, celui-ci va nous permettre de réaliser un programme, de le compiler et de le simuler.

Le programme que nous allons analyser est très rudimentaire puisqu'il s'agit de faire tout simplement clignoter une led sur une des broches du pic déclarée en sortie. Ce programme sans prétention va nous permettre d'aborder de manière pédagogique la conception d'une application. Une fois ce programme détaillé nous procéderons à l'étape suivante qui consistera à l'intégration sous MPLAB avec bien sûr des explications relatives à cet atelier logiciel.

Figure 1 – Un exemple simple de programme

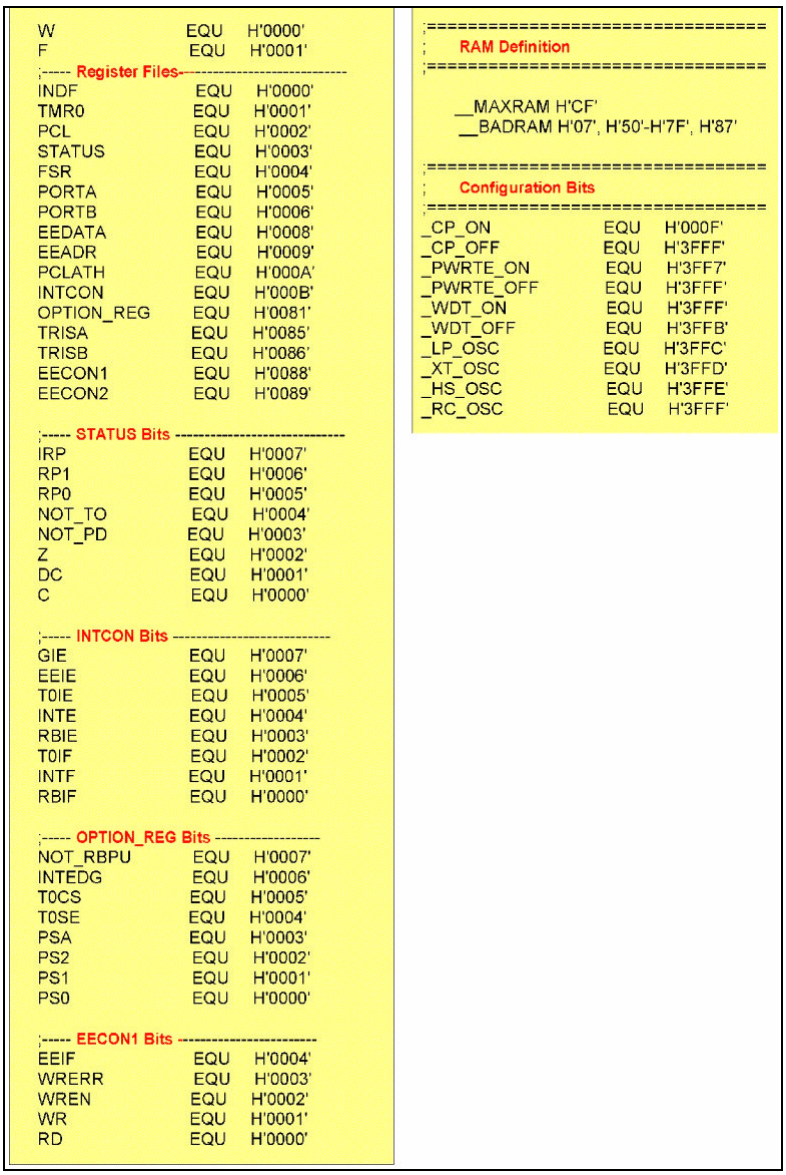
Présentation du programme et analyse

Notre premier programme (figure 2) a pour but de faire clignoter une led. Avant de rentrer dans le source nous allons voir quelques directives nécessaires au compilateur ainsi que la façon d'écrire un programme. Notez bien dans un premier temps que tout commentaire commence par un point virgule (;). Il est vivement conseillé de mettre des commentaires, ne serait- ce que pour vous permettre de comprendre et de reprendre le programme ultérieurement. Prenez donc le temps de réfléchir et choisissez des mots clé ainsi que des phrases explicites. Dans les commentaires on retrouve souvent un titre avec une ou deux phrases sur ce qu'est censé faire notre programme, ainsi que la date, la version du source, l'auteur et le PIC utilisé.



```
;------------  Exemple d'application avec un PIC : Un clignotant  ---------------

; Titre : Clignotant
; Date : 01 OCTOBRE 2004
; Auteur : P.M
; PIC utilisé : PIC 16 F 84
; On réalise un clignotant sur la broche RB0  d' un PIC 16 F 84 le quartz est de 4
; Mhz , on effectue une temporisation environ égale à 0.2 seconde.

;------------  Directive d' assemblage pour MPLAB  ---------------

 list p=16f84A              ;on indique ici le type de PIC utilisé
 #include p16f84A.inc       ;c'est le fichier qui contient la liste des adresses des registres internes
 __config H'3FF9'           ; définition des fusibles de configuration

;------------  Définition des constantes  ---------------

#define inter 0  ; bouton marche

;------------  Définition des registres temporaires  ---------------

retard1     EQU     0x0C     ; le registre temporaire retard1 se trouve à l' adresse 0C
retard2     EQU     0x0F     ; le registre temporaire retard2 se trouve à l' adresse 0F

;------------  Init des ports A et B  ---------------

        ORG     0                   ; notre programme commencera à l'adresse 0 en mémoire

        BSF STATUS,5                ; on met à 1 le 5eme bit du registre status pour accéder
                                    ; à la 2eme page mémoire ( pour trisa et trisb )

        MOVLW 0x00                   ; on met 00 dans le registre W
        MOVWF TRISB                  ; on met 00 dans TRISB le port B est programmé en sortie

        MOVLW 0x1F                   ; on met 1F dans le registre W
        MOVWF TRISA                  ; on met 1F dans TRISA le port A est programmé en entrée

        BCF STATUS,5                 ; on remet à 0 le 5eme bit du registre status pour accéder
                                     ; à la 1eme page mémoire

;------------  raz de la led et du PORTB  ---------------

       CLRF  PORTB                  ; RAZ du port B ( on éteint la led )

;------------  Programme principal  ---------------

debut
        btfss PORTA,inter      ; interrupteur  ( marche ) appuyé ? si oui on continu sinon
        goto debut             ; on retourne à l'étiquette debut

       COMF PORTB,F       ; On effectue le complément du PORTB
       MOVLW 0xFF         ; On charge le registre w avec la valeur FF (255)
       MOVWF retard1      ; On copie le contenu de w (FF) dans le registre retard1
       MOVWF retard2      ; On copie le contenu de w (FF) dans le registre retard2
       CALL  tempo        ; on appel la temporisation
       GOTO  debut        ; retour au début du programme

;------------  Programme de temporisation  ---------------

tempo
        DECFSZ retard1,F     ; on décrémente retard1 et on saute la prochaine instruction si
        GOTO  tempo          ; le registre retard1 = 0 sinon retour à tempo

        movlw 0xFF           ; on recharge retard1
        movwf retard1

        DECFSZ retard2,F     ; on décrémente retard2 et on saute la prochaine instruction si
        GOTO  tempo          ; le registre retard2 = 0 sinon retour à tempo

        RETURN               ; retour au programme principal après l 'instruction CALL
        END
```

FIGURE 2 – Notre premier programme



```
W                   EQU   H'0000'
F                   EQU   H'0001'
;----- Register Files----------
INDF                EQU   H'0000'
TMR0                EQU   H'0001'
PCL                 EQU   H'0002'
STATUS              EQU   H'0003'
FSR                 EQU   H'0004'
PORTA               EQU   H'0005'
PORTB               EQU   H'0006'
EEDATA              EQU   H'0008'
EEADR               EQU   H'0009'
PCLATH              EQU   H'000A'
INTCON              EQU   H'000B'
OPTION_REG          EQU   H'0081'
TRISA               EQU   H'0085'
TRISB               EQU   H'0086'
EECON1              EQU   H'0088'
EECON2              EQU   H'0089'

;----- STATUS Bits --------------------
IRP                 EQU   H'0007'
RP1                 EQU   H'0006'
RP0                 EQU   H'0005'
NOT_TO              EQU   H'0004'
NOT_PD              EQU   H'0003'
Z                   EQU   H'0002'
DC                  EQU   H'0001'
C                   EQU   H'0000'

;----- INTCON Bits --------------------
GIE                 EQU   H'0007'
EEIE                EQU   H'0006'
T0IE                EQU   H'0005'
INTE                EQU   H'0004'
RBIE                EQU   H'0003'
T0IF                EQU   H'0002'
INTF                EQU   H'0001'
RBIF                EQU   H'0000'

;----- OPTION_REG Bits ----------------
NOT_RBPU            EQU   H'0007'
INTEDG              EQU   H'0006'
T0CS                EQU   H'0005'
T0SE                EQU   H'0004'
PSA                 EQU   H'0003'
PS2                 EQU   H'0002'
PS1                 EQU   H'0001'
PS0                 EQU   H'0000'

;----- EECON1 Bits --------------------
EEIF                EQU   H'0004'
WRERR               EQU   H'0003'
WREN                EQU   H'0002'
WR                  EQU   H'0001'
RD                  EQU   H'0000'
```

```
;***********************************
;     RAM Definition
;***********************************

    __MAXRAM H'CF'
    __BADRAM H'07', H'50'-H'7F', H'87'

;***********************************
;     Configuration Bits
;***********************************
_CP_ON              EQU   H'000F'
_CP_OFF             EQU   H'3FFF'
_PWRTE_ON           EQU   H'3FF7'
_PWRTE_OFF          EQU   H'3FFF'
_WDT_ON             EQU   H'3FFF'
_WDT_OFF            EQU   H'3FFB'
_LP_OSC             EQU   H'3FFC'
_XT_OSC             EQU   H'3FFD'
_HS_OSC             EQU   H'3FFE'
_RC_OSC             EQU   H'3FFF'
```

Figure 3 Le fichier p16f84A.inc

Quelques directives d'assemblage

On entend par directives d'assemblage des ordres qui seront exécutés par le compilateur lorsque vous demanderez de transformer votre programme source en un fichier binaire compréhensible par le microcontrôleur.

Une des premières directives indique au compilateur le type de PIC utilisé, cette information est représentée par la ligne : **list p=16f84A** (on à prévu un montage avec un pic 16f84A).

La deuxième directive présente dans notre programme est : **#include p16f84A.inc** , le #include signifie que le compilateur intégrera au programme le fichier qui suit (le fichier p16f84A.inc est fournit avec le logiciel MPLAB) ,ici dans notre cas se sera le fichier p16f84A (**figure 3**)celui-ci définit les emplacements mémoire des registres internes ainsi que les valeurs des fusibles de configuration. C'est notamment grâce à ce fichier que l'on pourra utiliser dans notre programme des noms de registres (exemples: *PORTA, TRISB, STATUS*), ces noms figurent dans ce fichier de configuration.

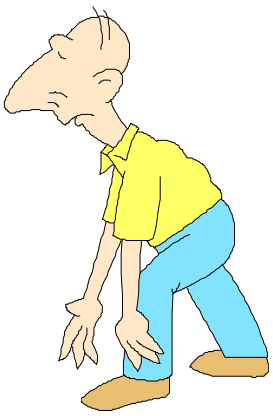


Figure 4 – Les directives d'assemblage

La troisième directive est : **__config H '3FF9'** ,celle-ci va définir la position de certains fusibles qui gèrent les mode de fonctionnement souhaités pour l'application
Prenez le temps de regarder le fichier p16f84A.inc (**figure 3**) et notamment la rubrique 'Configurations Bits' :

- Si vous ne voulez pas utiliser le "code protect" , ce bit qui interdit toute relecture de votre programme se trouvant dans la mémoire flash du pic , alors il faut placer _CP_OFF dans la variable __config donc il faut mettre 3FFF (hexadécimal) tel que précisé dans le fichier p16f84A.inc .

- - Puis si vous ne voulez pas utiliser la fonction power up timer (pwrte est une temporisation de 72 mS activée au démarrage du PIC) il faut mettre PWRTE_OFF donc encore 3FFF dans la variable __config.

- - Si vous ne voulez pas utiliser le watchdog (processus interne permettant de surveiller votre programme) il faut mettre _WDT_OFF donc 3FFB dans __config.

- - Si vous utilisez un quartz de moyenne vitesse (4Mhz) vous devez sélectionner l 'oscillateur _XT_OSC donc on doit mettre 3FFD dans __config.

quand nous faisons un "et" logique de tout cela nous avons 3FFF & 3FFF & 3FFB & 3FFD nous obtenons la valeur 3FF9

Au lieu de 3FF9 , nous pouvons également écrire la ligne suivante :

__Config _CP_OFF & _WDT_OFF & PWRTE_OFF & _XT_OSC

Ce qui est peut être un peu plus explicite…

Dans notre programme nous utilisons également la ligne suivante : **#define inter,0** (cette ligne est optionnelle) cela à pour intérêt de pouvoir utiliser (toujours dans un souci de clarté et de facilité de compréhension) le mot inter qui désigne comme son nom l'indique un interrupteur (bouton marche de notre application). Quand le compilateur lors de la phase de compilation rencontrera le mot inter dans le source il le remplacera par 0, ainsi cela clarifie notre programme et le compilateur n'est pas perdu…
Si vous ne souhaitez pas utiliser de #define il faudra alors travailler avec des valeurs dans votre programme ainsi la ligne suivante :

BTFSS PORTA,inter

Devra être remplacée par :

BTFSS PORTA, 0

Et le fonctionnement sera identique.

Analyse du programme

Nous allons utiliser dans notre programme une subroutine (un sous-programme) de temporisation, pour réaliser celle-ci nous allons décrémenter en cascade deux registres internes du PIC (registres 8 bits). Dans un premier temps nous déclarons les deux registres que nous allons utiliser. Nous appellerons arbitrairement ces registres "retard1" et "retard2" et nous indiquons l'adresse à laquelle chacun d'eux se situe dans le PIC. Sachant que les registres utilisateurs (ou variables mémoire) sont disponibles à partir de l'adresse 0C en hexadécimal (soit 12 en décimal) et jusqu'à l'adresse 4F en hexadécimal (soit 79 en décimal) ce qui représente 68 adresses , donc 68 registres utilisables (**Figure 5**).

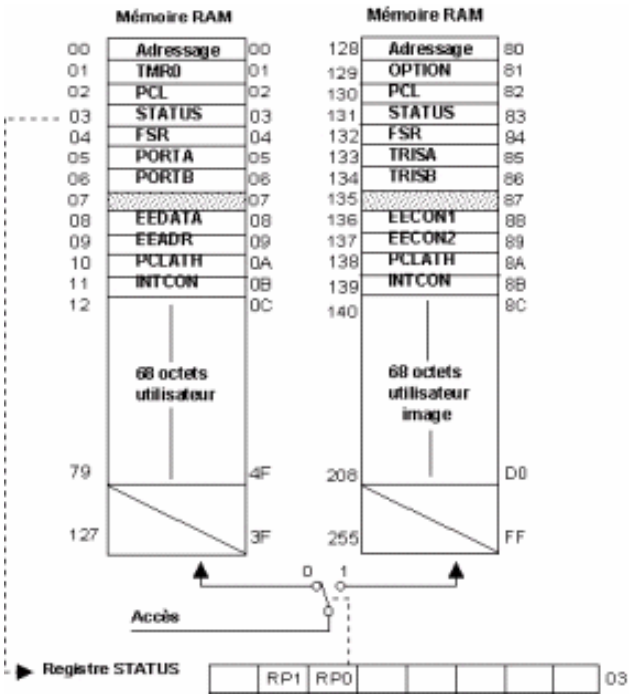


FIGURE 5 – Rappel du Mapping Mémoire

Les deux lignes suivantes permettent de déclarer ces deux registres, la directive EQU (comme équivalence) attribue une adresse mémoire au nom indiqué (retard1 sera à l'adresse 12 et retard2 sera à l'adresse 15). Vous pouvez choisir bien sûr d'autres adresses que celles choisies, en restant toutefois dans la plage autorisée (12 à 79). Dans l'écriture 0xC par exemple le "x" signifie que l'on est en hexadécimal.

```
Retard1     EQU   0xC
Retard2     EQU   0xF
```

Une fois les registres déclarés nous avons une instruction du type **ORG 0**, celle-ci indique au compilateur que le programme devra être logé dans la mémoire flash du PIC à l'adresse 0, sachant que cette mémoire à une capacité de 1k (ou plus exactement de 1024 instructions, chaque instruction étant codée sur 14 bits) . Dans notre exemple nous aurions pu loger notre programme à une autre adresse (adresse 100 par exemple), mais il y aura des cas (par exemple si l'on utilise les interruptions), ou il faudra gérer correctement l'endroit ou débute le programme, sachant que pour les interruptions par exemple l'adresse 4 est réservée ou plus précisément à chaque fois qu'une interruption se produira le Compteur de programme (ou compteur ordinal) se chargera avec l'adresse 4, nous y reviendrons ultérieurement lorsque nous réaliserons un programme avec des interruptions (**figure 6 et figure 7**).

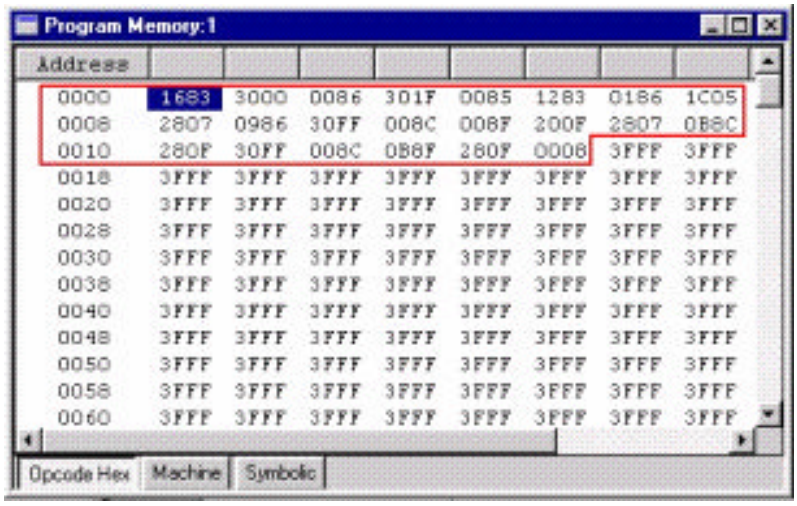


Figure 6 – Programme compilé à l'adresse 0 (org 0)

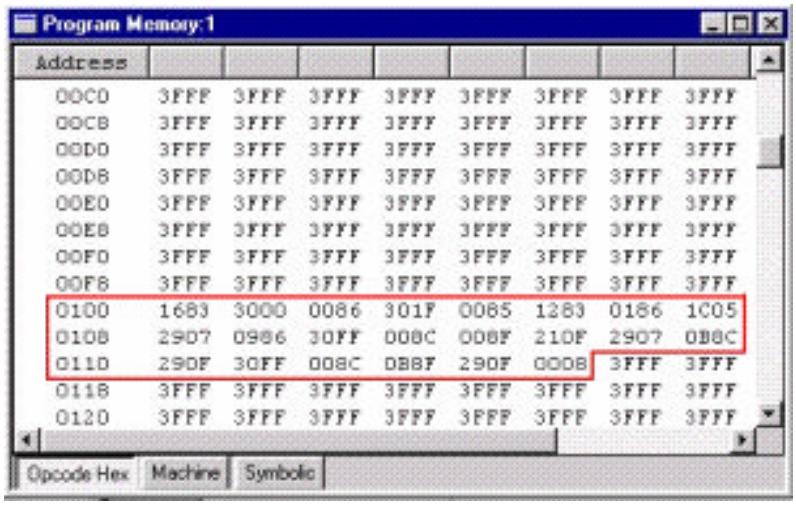


Figure 7 – Le même programme compilé à l'adresse 100 (org 100)

Nous allons dans la suite du programme déclarer les broches du PIC (port) soit en entrée soit en sortie et ceci en plaçant une valeur dans les registres de direction TRISA (pour la configuration du port A)et TRISB (pour la configuration du port B).
Si l'on regarde la **figure 5** on s'aperçoit que les registres TRISA et TRISB sont dans la deuxième page mémoire.

L'instruction suivante **BSF STATUS, 5** permet de passer à 1 le cinquième bit du registre STATUS (bit RP0), ce bit s'il est à 1 permet d'accéder à la deuxième page mémoire dans laquelle se situe justement les registres de direction des ports A et B, sur lesquels nous devons effectuer une opération de chargement.

Une fois la 2eme page sélectionnée, la suite consiste à charger le registre de travail w avec la valeur indiquant la direction des ports (0 pour une sortie et 1 pour une entrée) puis à transférer le registre de travail w dans TRISA ou TRISB.

Pour le port B on ne veut que des sorties donc TRISB vaudra 0 (en binaire 0000 0000 donc les huit broches RB0 – RB7 sont en sortie)

MOVLW 0x00
MOVWF TRISB

Pour le port A on ne veut que des entrées donc TRISA vaudra 1F (en binaire 0001 1111 donc les cinq broches RA0 – RA4 sont en entrée)

MOVLW 0x1F
MOVWF TRISA

Puis on repasse le bit RP0 à 0 pour revenir à la page mémoire 1 dans laquelle se trouve notamment les port A et B sur lesquels nous devons travailler.
L'instruction suivante **BCF STATUS,5** permet de passer à 0 le cinquième bit du registre STATUS (bit RP0) , ce bit s'il est à 0 permet d'accéder à la première page mémoire.

Ensuite vient la ligne **CLRF PORTB** qui a pour rôle de faire une RAZ (tous les bits à zéro) du port B, cette instruction n'est pas obligatoire, mais elle permet de commencer le programme de manière identique à chaque mise sous tension (on est sûr que la led connectée sur le port B sera éteinte).

Le début du programme

Nous avons choisit dans notre montage de mettre un bouton marche arrêt pour tester si ce bouton est en position "on" ou "off" nous allons utiliser les instructions suivantes :

```
debut
        BTFSS PORTA,inter
        Goto debut
```

Notez déjà dans un premier temps que les étiquettes (étiquette "début" dans notre exemple) sont toujours en début de ligne, alors que les instructions sont décalées, il faudra toujours respecter cette règle.
L'instruction BTFSS PORTA, inter va tester si le bit 0 (c'est la broche RA0 ou est connecté l'interrupteur) du port A vaut 1 et dans ce cas sautera la prochaine instruction, celle-ci goto debut fait un branchement inconditionnel à l'étiquette "début". En fait on ne pourra commencer à exécuter le programme que si l'interrupteur est actionné (c'est exactement ce que l'on veut non?).

Si nous considérons que la led est éteinte, le but de notre programme étant de faire clignoter cette led, il va falloir l'allumer. Il y aurait plusieurs moyens d'allumer la led , (exemple BSF PORTB,0 si la led est connectée à RB0 **Figure 8**) l'instruction **COMF PORTB,F** fait un complément (on inverse la valeur de tous les bits 0=>1 et 1=>0) du PORTB. Sachant que celui-ci était dans notre initialisation à 0 il passe à FF (1111 1111) et la led s'allume (on peut donc câbler cette led sur n'importe quelle broche de sortie du port B RB0 à RB7 et cela fonctionnera). La prochaine fois que le programme passera par cette instruction il fera l'inverse (tous les bits à zéro) et ainsi de suite.

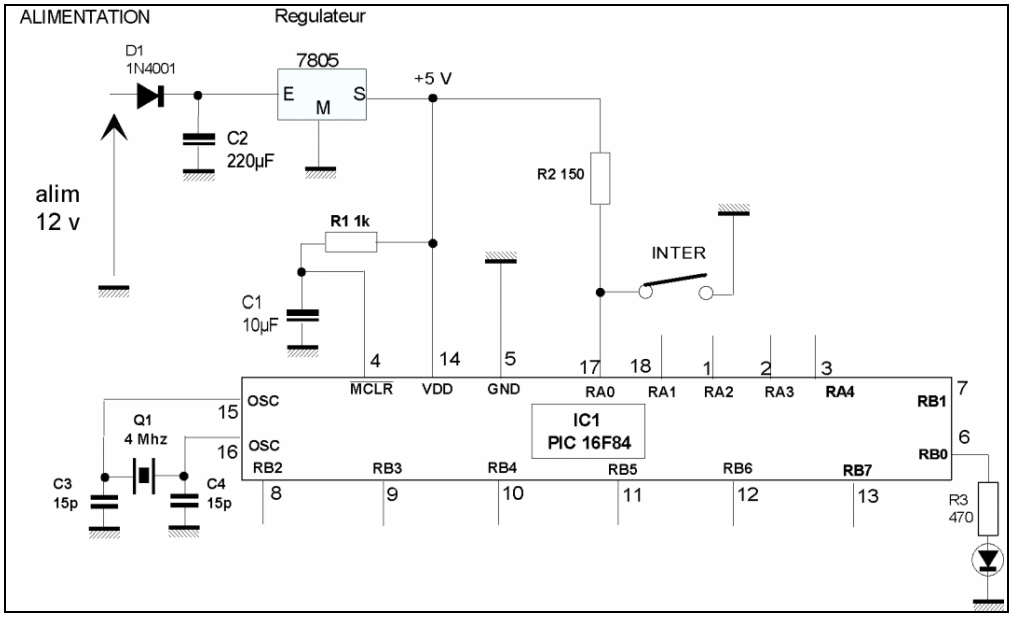


Figure 8 Schéma de principe

Si nous en restions là et que l'on rebouclerait le programme cela fonctionnerait, mais il faudrait être *superman* pour voir la led clignoter, car la fréquence serait trop élevée pour notre pauvre rétine…
Vous l'avez donc compris il va falloir réaliser une temporisation.

Dans un premier temps nous allons charger des valeurs dans deux registres (retard1 et retard2) puis nous les décrémenterons , notez que si il y a deux registres c'est que la fréquence serait encore trop élevée même si on utilisait un seul registre.
Pour charger les valeurs nous passons dans un premier temps par le registre de travail w, celui-ci distribuera ensuite ces valeurs vers les registres retard1 et retard2.
L'instruction **MOVLW 0xFF** charge le contenu de w avec la valeur ff (255).
Puis la copie dans les registres se fait par les instructions **MOVWF retard1** et **MOVWF retard2.** Les deux registres sont maintenant chargés, nous pouvons faire l'appel à la temporisation : l'instruction **Call tempo** fait un branchement à l'étiquette tempo.
L'instruction qui suit le Call tempo est **Goto debut,** cette instruction permet de reboucler le programme à l'étiquette début, endroit où l'on teste l'état de l'interrupteur marche –arrêt.

Programme de temporisation

Nous commençons dans ce sous-programme par décrémenter le registre retard1 qui était chargé précédemment (avant l'appel de la temporisation) à 255 (FF). L'instruction **DECFSZ retard1,F (décrément f skip if 0)** permet de réaliser la décrémentation et ceci tant que la valeur du registre retard1 ne vaut pas 0, si le registre vaut 0 alors on saute la prochaine instruction (goto tempo) donc nous allons effectuer 255 fois cette boucle.

Une fois cette première boucle effectuée on recharge la valeur de retard1, ceci pour faire une imbrication des décomptages des deux registres. L'instruction **MOVLW 0xFF** charge le registre w avec 255 (FF) et l'instruction **MOVWF retard1** copie le contenu de w dans retard1.

Ensuite nous allons décrémenter le registre retard2 avec l'instruction **DECFSZ retard2,F (décrément f skip if 0)**, tout comme pour le registre retard1 nous allons effectuer 255 fois cette opération. Remarquez qu'à chaque fois que l'on fait retard2 – 1 on retourne à l'étiquette tempo donc on repasse par la décrémentation du registre retard1, ce qui implique qu'en fait on réalise 256 x 255 décomptages du registre retard1 soit 65280, ce qui au total nous donnera une temporisation de 0.2 seconde environ.

Détails simplifiés du calcul :
Toutes les instructions dans la boucle prennent 1 cycle d'horloge excepté l'instruction goto (2 cycles). Pour le premier registre que l'on décrémente 256 x 255 on a un temps machine de 1µs x 65280 x 3 tops (1top pour DECFSZ et 2 tops pour goto) soit environ 196 ms (avec un quartz de 4 MHz sachant que l'horloge est divisée par 4 en interne, donc un top d'horloge = 1 µ seconde) et pour le reste de la boucle on passera 255 fois avec 5 tops d'horloges (2 pour goto et 3 pour MOVLW ,MOVWF et DECFSZ) donc 5 x 255 = 0.12 ms. Au total on aura donc 196 mS + 0.12 mS $\cong$ 0.2 seconde

Vient ensuite la dernière instruction, l'instruction **Return** celle-ci permet de revenir au programme principal et le cycle recommence…tant que vous n'avez pas remis l'interrupteur sur arrêt.

Il est maintenant temps de vous présenter le logiciel MPLAB qui va permettre de réaliser notre premier programme et de le compiler.
Les vues d'écran sont issues de la version 6.61 de MPLAB (octobre 2004).
La première chose à faire est bien sûr de télécharger cette version gratuite de MPLAB sur le site de Microchip (www.microchip.com).
Une fois le fichier MP661.zip (attention 38 Mo) correspondant à MPLAB version 6.61 téléchargé et installé, il vous reste à lancer l'exécutable.
A la **figure 9** le premier écran que vous devez avoir.

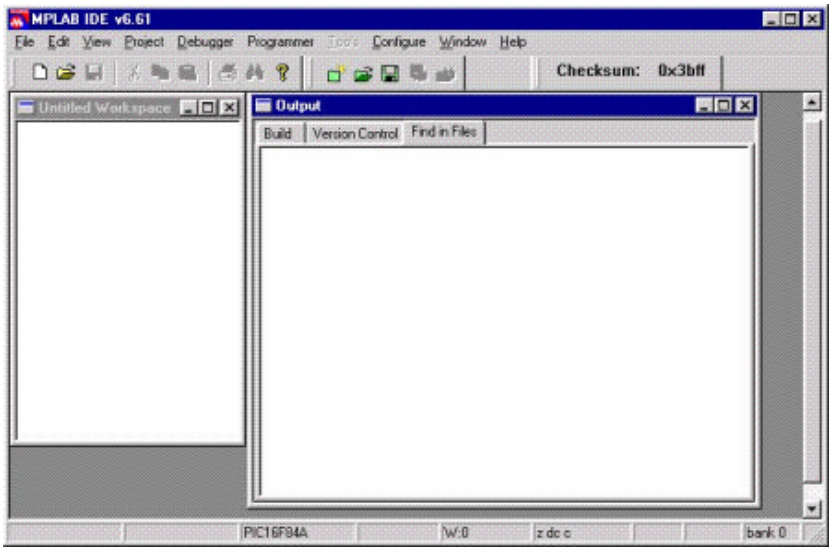
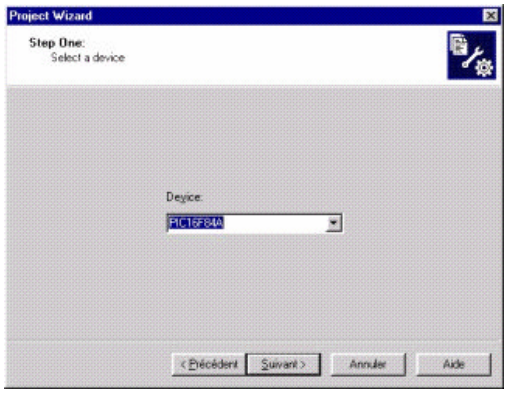


Figure 9 – MPLAB le premier écran

La version 6.61 intègre un fichier d'aide à la mise en route (un WIZARD) qui est bien pratique, il est à noter que l'on peut bien sûr créer un projet sans utiliser le Wizard. Pour lancer le wizard cliquez dans le menu "Project" puis "Projet Wizard". Cliquez sur "suivant" puis ensuite sélectionnez le type de PIC utilisé, pour nous se sera le PIC 16F84A (figure 10), une fois sélectionnez cliquez sur "suivant".



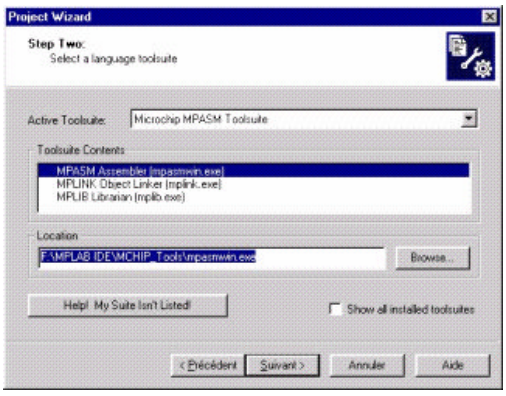


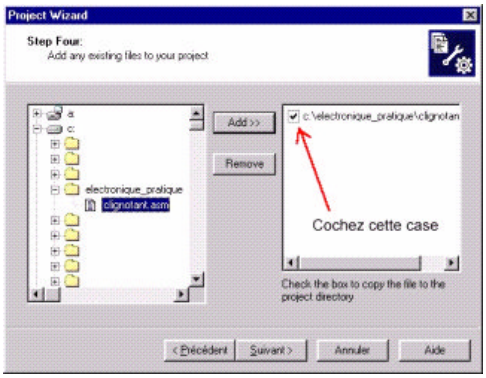
Figure 10 – Sélection du PIC à utiliser ***Figure 11 – Sélection du langage***

Sélectionnez le langage tel que représenté sur la **figure 11.** Ensuite il vous reste à donner un nom à votre projet (par exemple "clignotant") et d'indiquer à quel endroit les fichiers doivent être stockés sur le disque dur de votre ordinateur (par exemple c:\clignotant - **Figure 12**)
Ensuite le Wizard vous propose d'ajouter un fichier assembleur à votre projet, si vous avez par exemple téléchargé le fichier "clignotant.asm" présent sur le site de la revue (www.electroniquepratique.com) ou sur le site de l'auteur (voir à la fin de l'article

) vous pouvez alors le joindre en faisant bien attention de valider la case à cocher présente sur la partie droite de la fenêtre permettant de copier le fichier sélectionné dans le répertoire que vous avez définit **(figure 13)**, sinon cliquez sur suivant (vous pouvez rajouter ou créer ce fichier assembleur plus tard)

Ensuite appuyez sur le bouton "terminer", c'est la fin du Wizard qui peut paraître un peu long mais avec l'expérience cela va beaucoup plus vite par la suite.

Si vous avez respecté les vues précédentes voilà ce que vous devriez obtenir en **figure 14.**



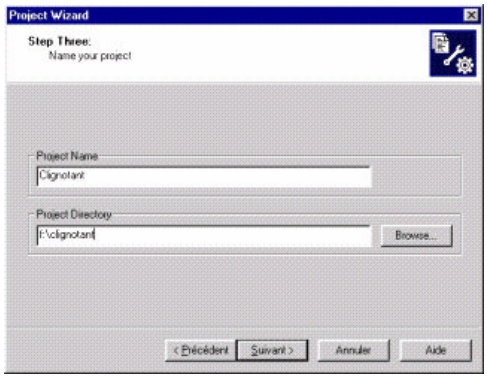


Figure 12 – Un nom pour le projet fichier .ASM

Figure 13 – Insertion d'un

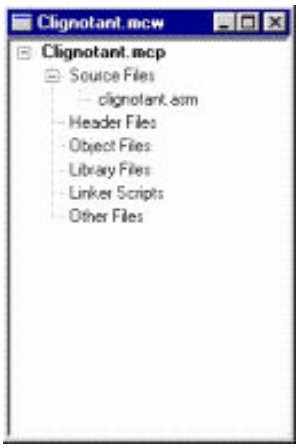
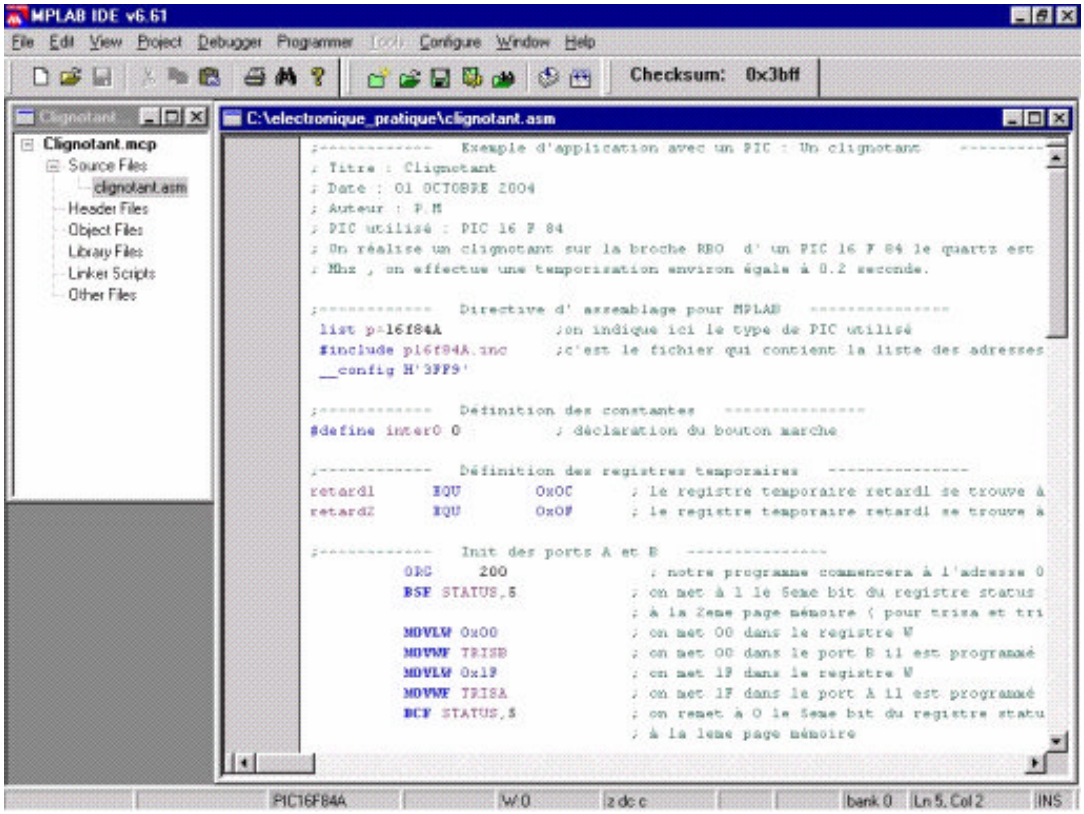


Figure 14 – Le projet avec le fichier asm

Dans le cas ou vous avez téléchargé le fichier clignotant.asm double cliquez sur le nom de ce fichier dans la fenêtre représentée **figure 14**, vous devez alors voir la fenêtre de l'éditeur avec le source (**figure 15**).

Si vous n'avez pas téléchargé le fichier clignotant.asm il vous suffit d'aller dans le menu "File" puis "New", une nouvelle fenêtre est alors à l'écran, tapez votre source tel que représenté **figure 2** puis depuis le menu "File" puis "Save as…" enregistrez ce fichier sous "clignotant.asm" dans la directorie (répertoire) que vous avez déclarée dans le Wizard (exemple c:\clignotant). Une fois enregistré, faire un click droit sur le texte "Source Files" (**figure 16**) et sélectionnez " Add Files" puis le fichier "clignotant.asm" que vous venez de créer. Une fois ces actions réalisées vous devez alors avoir la fenêtre représentée en Figure 14 puis si vous double-cliquez sur le nom "clignotant.asm" être au même point que précisé sur la figure 15.



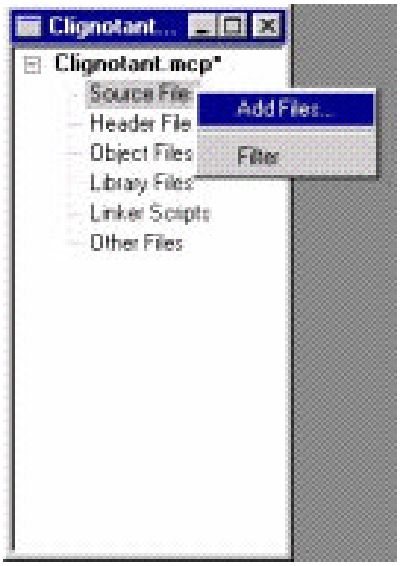


Figure 15 – Le fichier source

Figure 16 – Pour ajouter un fichier source au projet

Pour compiler le source

Nous voici arrivés à la fin de cet article et nous allons conclure avec la compilation de notre programme, si cette compilation à bien fonctionné, un fichier nommé

"clignotant.hex" sera crée dans le répertoire que vous avez indiqué dans le Wizard (c:\clignotant par exemple), ce fichier .hex correspond au fichier binaire qu'il faudra transférer vers la mémoire du PIC, pour ce faire il faudra un programmateur et un logiciel de transfert. Pour les lecteurs possédant un programmateur reconnu par le logiciel (exemple le PICSTART PLUS de Microchip), le transfert se fera depuis MPLAB.
Pour la compilation allez dans le menu "project" puis "Built All", une fenêtre de compilation (**figure 17**) apparaît alors, celle-ci vous indique le pourcentage de compilation. Si des erreurs sont détectées alors une fenêtre de debug vous indique le numéro de la ligne ou il y a une erreur, afin que vous la corrigiez (exemple **figure 18** ou l'on a volontairement enlevé le "F" de l'instruction BSF STATUS, 5 ligne 23).

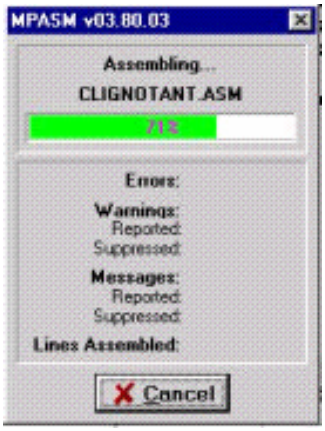


Figure 17 – La fenêtre de compilation

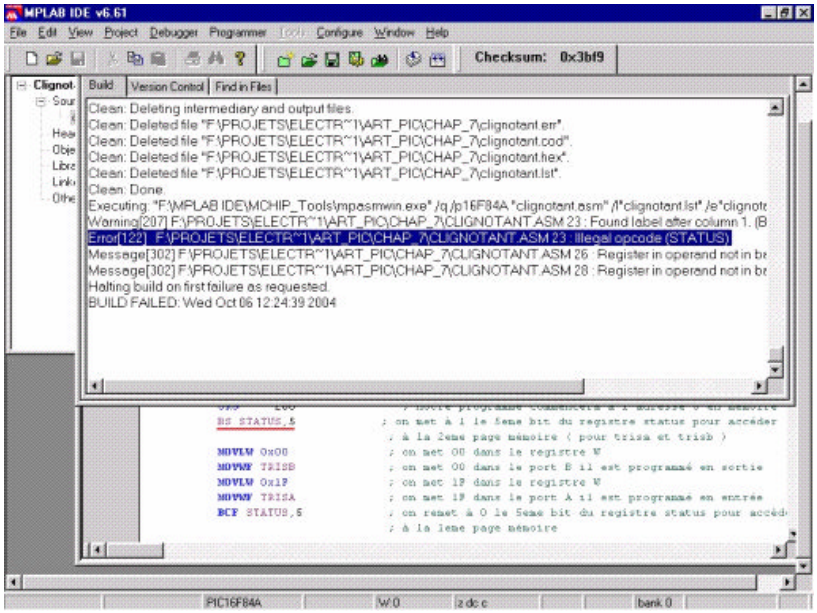


Figure 18 – L'erreur détectée par MPLAB

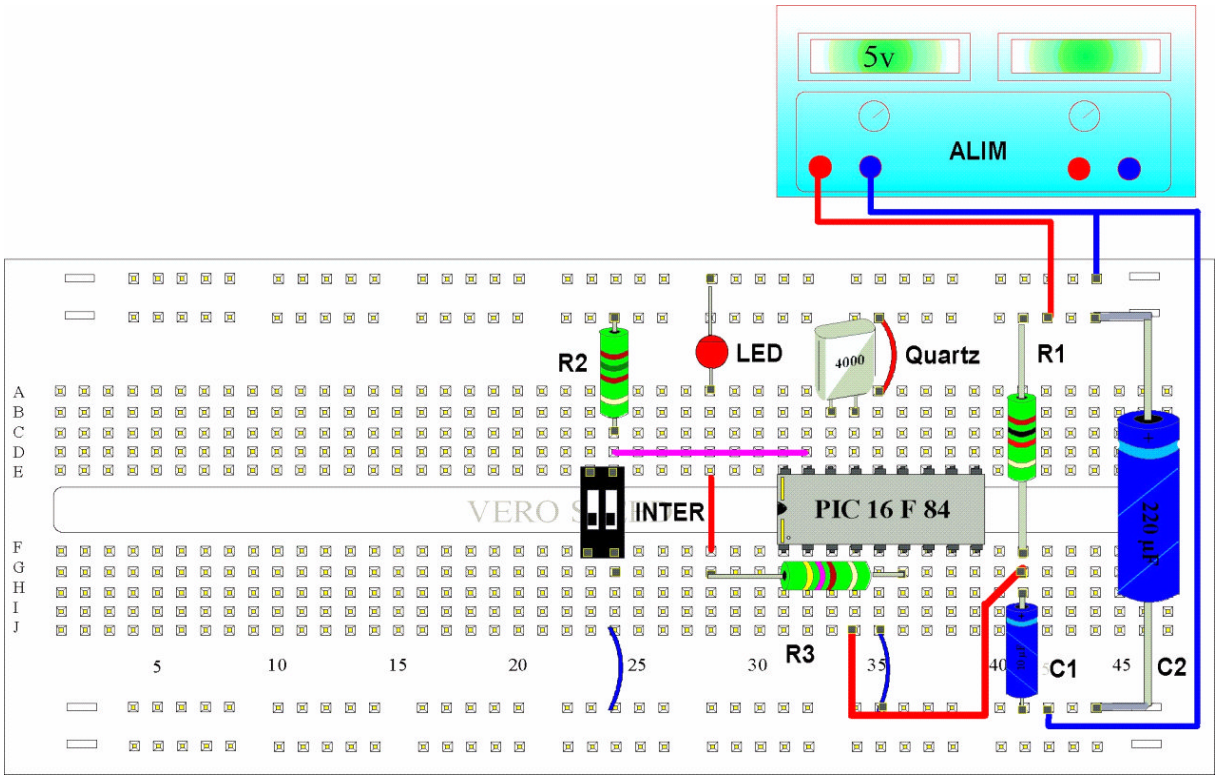
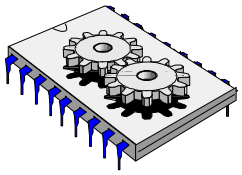


Figure 19 – Le montage réalisé à l'aide d'une platine d'essai type LABDEC

Pour conclure ce chapitre

Avec cette sixième partie nous avons analysé et compilé notre premier programme sous MPLAB, dans le prochain numéro nous aborderons les aspects programmateur afin de pouvoir transférer le fichier binaire vers le PIC ainsi que la partie simulation.
En attendant de réaliser une platine pour tester nos futurs programmes et pour les lecteurs désireux de réaliser le clignotant présenté dans cet article, voici **Figure 19** le montage réalisé avec une Platine de type LABDEC.



A la découverte des microcontrôleurs PIC Septième partie

Dans ce numéro nous allons détailler le transfert d'un fichier compilé vers la mémoire d'un PIC en nous intéressant plus particulièrement à l'aspect matériel et aux programmateurs et nous reviendrons sur la simulation avec le logiciel MPLAB.

Nous avons dans le dernier numéro détaillé puis compilé un programme. Nous allons aborder aujourd'hui un aspect intéressant, celui de la simulation. L'intérêt de simuler un programme est bien sûr très pédagogique car comme nous le verrons nous allons pouvoir faire du "pas à pas " et visualiser le fonctionnement de tous les registres constituant le PIC. De plus nous allons pouvoir vérifier le fonctionnement du programme avant de le transférer vers la mémoire du microcontrôleur, cela nous permettra d'économiser du temps et également d'augmenter la durée de vie de notre PIC, puisque la mémoire flash de celui-ci n'est pas comme vous le savez éternelle...

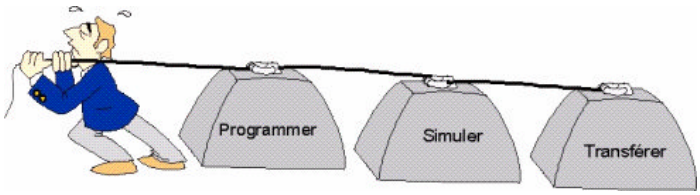


Figure 1 – phases de développement

Nous allons dans l'exemple simuler le clignotant, programme que nous avons détaillé dans la dernière leçon.

Pour passer un programme en mode simulation vous devez depuis le logiciel MPLAB ouvrir votre projet tel que nous l'avons vu dans notre dernière leçon puis depuis le menu " "Debugger", sélectionnez "Select Tools" et "MPLAB SIM" (Figure 2).

Une fois le mode simulation déclaré il faut placer les stimuli qui correspondent aux entrées du PIC, pour se faire allez dans le menu "Debugger", sélectionnez "Stimulus Controller", vous devez alors avoir une fenêtre supplémentaire correspondant à la Figure 3, c'est dans cette fenêtre que nous allons déclarer les entrées qui seront alors modifiables par un clic de souris.

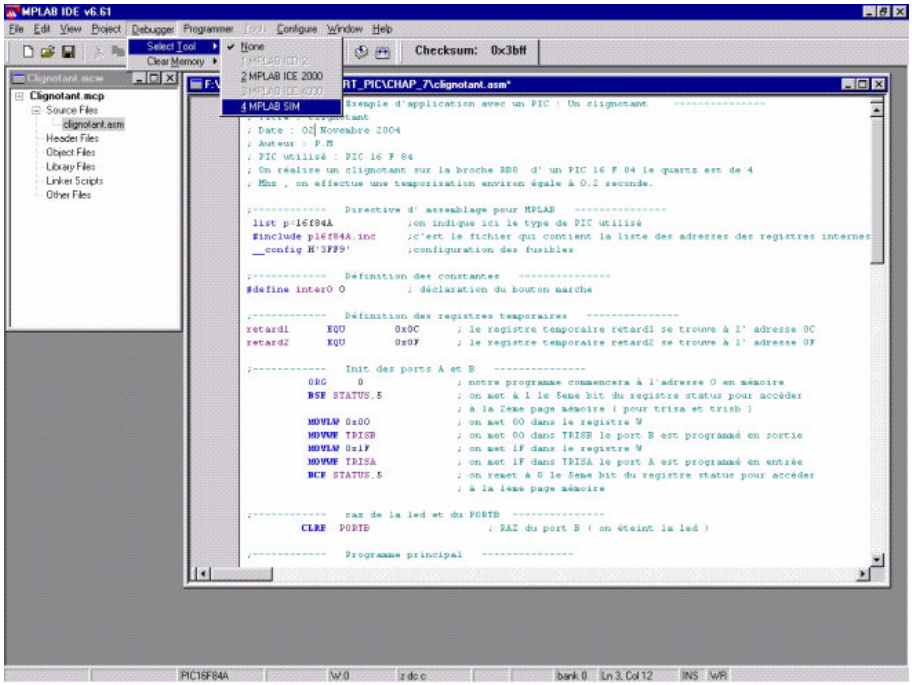


Figure 2 – Sélection du mode simulation

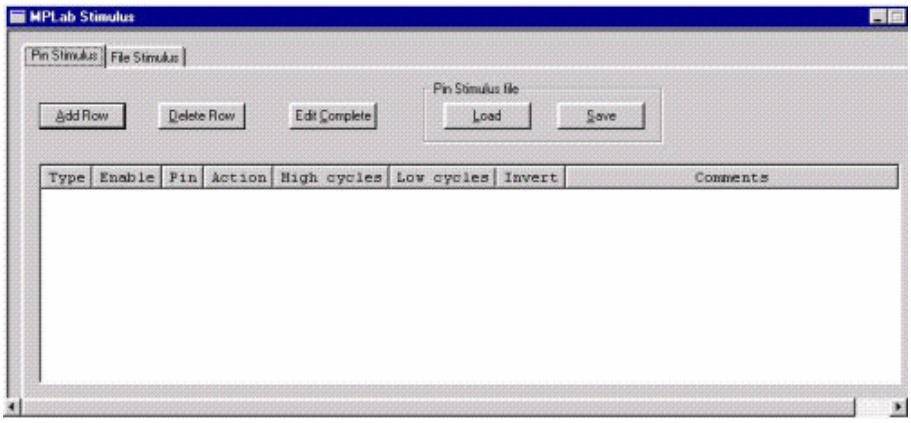


Figure 3 – Fenêtre de déclaration des stimuli

Dans notre programme "clignotant" nous avons en fait qu'une seule entrée à piloter, c'est le bouton marche-arrêt dont l'état permet au programme soit de se lancer soit de se reboucler sur le test de cet interrupteur.

C'est la broche RA0 qui à pour rôle de recevoir cet interrupteur, nous allons donc déclarer celle-ci dans la fenêtre de stimuli.

Pour déclarer la broche RA0 en entrée de stimuli cliquez sur le bouton "ADD ROW" puis dans le type sélectionnez "Asynch" ainsi que la broche concernée "RA0" puis dans la dernière liste déroulante l'option "Toggle". Le bouton "Fire" est là pour valider désormais vos click de souris, sachant que nous avons pris l'option "Toggle", chaque

appui sur "Fire" fera changer l'état de la broche RA0. Bien sûr si nous avions plusieurs entrées à simuler il suffirait de refaire un " Add Row" et de recommencer les étapes détaillées ci-dessus (Figure 4).

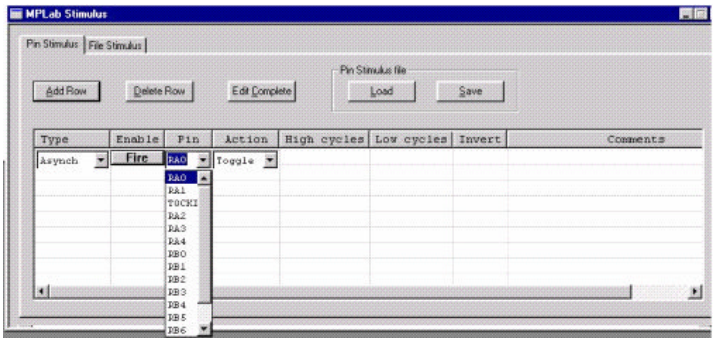


Figure 4 – Sélection des entrées de stimuli

Nous venons de mettre en place notre entrée de validation RA0, nous pouvons dès lors commencer la simulation. Pour ce faire et afin de visualiser l'action sur le bouton "Fire" qui simulera notre entrée RA0, nous allons ajouter une vue d'écran, cliquez dans le menu "View" puis sélectionnez "Spécial Function Registers" afin de visualiser l'état du port A (donc pour visualiser l'entrée RA0 entre autre) cet écran est représenté figure 5.

Sélectionnez maintenant le menu "Debugger" puis cliquez sur "Animate" dans votre fenêtre où il y a le source du programme apparaît une flèche verte qui se positionne au rythme de l'horloge sur l'instruction en cours d'exécution (Figure 5) , si vous n'avez pas appuyé encore sur le bouton "Fire" la flèche oscille entres les deux instructions :

btfss PORTA,inter0 ; interrupteur 0 (marche) appuyé ? si oui on continu sinon on retourne à l'étiquette debut

goto debut ;

Appuyez maintenant sur le bouton "Fire" et notez au passage le changement d'état du registre PORTA de la fenêtre Spécial Function Registers qui passe de 0000 0000 à 0000 0001 puisque la broche RA0 vient de passer à 1 par le click de souris. A ce moment le programme continu de se dérouler, l'appui sur le bouton marche-arrêt vient d'être simulé, pratique non ?

Pour arrêter le mode animation appuyez sur la touche F5 ou bien allez dans le menu "Debugger" puis cliquez sur "Halt".

Pour faire un reset de l'animation appuyez sur la touche F6 ou bien allez dans le menu "Debugger" puis cliquez sur "Reset" et sur "Processor Reset"

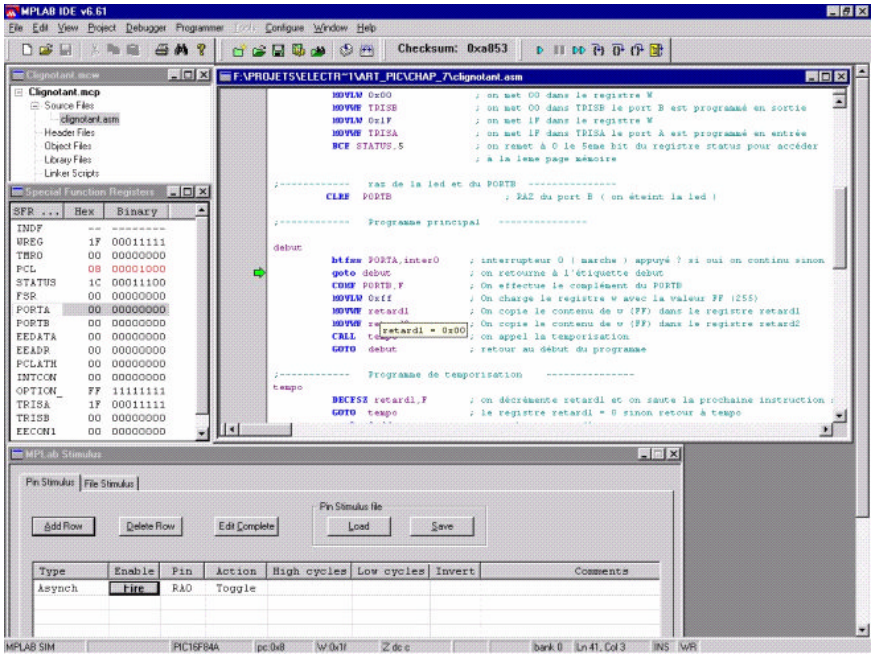


Figure 5 – Phase de simulation

Depuis le menu view vous pouvez ajouter des écrans composés de différents registres internes du PIC, y compris ceux que vous avez déclaré dans votre programme. Par exemple depuis le menu "View" cliquez sur "Watch" puis dans la liste déroulante "Add Symbol" sélectionnez le registre "retard1" et validez sur le bouton "Add Symbol". Comme vous l'avez constaté vous pouvez visualiser tous les registres internes au PIC ce qui est très pédagogique pour en comprendre le fonctionnement (figure 6).

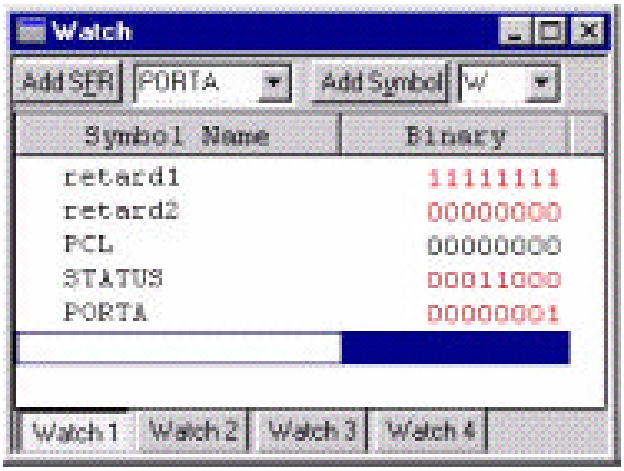


Figure 6 – Visualisation de différents registres internes

Nous voici arrivés au stade ou nous avons écrit notre programme, nous l'avons simulé et tout ce passe bien, la simulation correspond à nos attentes…
Nous pouvons maintenant transférer le fichier compilé (clignotant.hex) vers la mémoire du PIC.

Phase de programmation

Trois signaux sont nécessaires pour programmer un PIC, en effet la programmation nécessite un signal d'horloge, celui-ci sera appliqué sur la broche RB6 du PIC, quant aux données elles transiteront sur la broche RB7, il est à noter que cette broche sera bidirectionnelle puisque l'on pourra également lire le programme contenu dans la mémoire flash du PIC ainsi que dans la mémoire EEPROM.
Lors de la programmation une tension de 12 V est appliquée par le programmateur sur la broche MCLR/ indiquant à la logique interne que le mode programmation est demandé, un reset du PIC pour initialiser le compteur ordinal est alors effectué.
Un chronogramme simplifié des signaux est représenté **figure 7** .

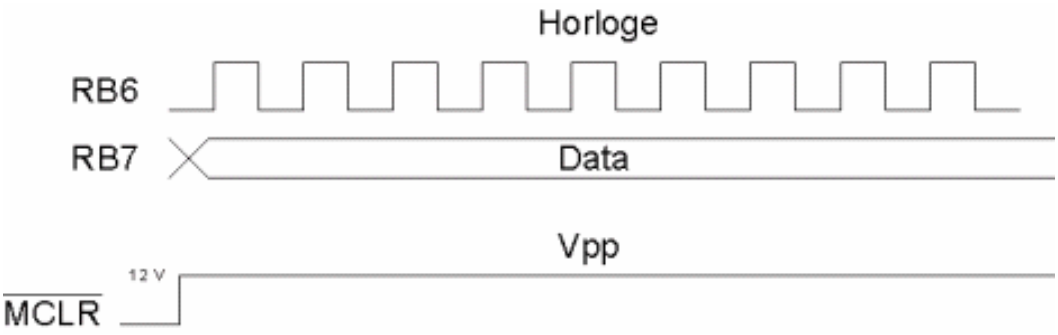


Figure 7 – Les signaux de programmation

Programmation ICP

La programmation du PIC nécessite soit d'enlever le PIC du montage sur lequel il est implanté et de l'insérer dans le programmateur soit de réaliser la programmation in-situ (directement sur la platine d'application), on parlera alors de programmation ICP (In-Circuit Programming). Le mode ICP est très pratique puisque l'on a pas besoin d'enlever le PIC du montage pour le programmer ce qui est appréciable lorsque l'on fait de la mise au point, il y a quand même quelques précautions à prendre, notamment l'isolation des broches RB6 et RB7 qui sont utilisées lors de la phase de programmation, un schéma de principe est donné Figure 8**.**

Les fils de connexion entre la platine et le programmateur devront être le plus court possible pour éviter les effets de capacité parasite qui pourraient nuire à la programmation. Il est à noter que la masse doit être reliée entre le programmateur et la platine ainsi que le +5V.

Mode LVP
Sur d'autres PIC de la même famille tel que le PIC 16F628 il est possible de réaliser une programmation en mode basse tension (5V) et ceci suivant la configuration d'un bit nommé LVP (low voltage programming).
Si ce bit est positionné à 0 alors la programmation se fait en +12V (appliqué toujours sur la broche MCLR) ce qui permet d' être compatible avec le PIC 16F84 dans le cas du PIC 16F628 ou bien si ce bit est positionné à "1" la programmation se réalise alors en 5v.

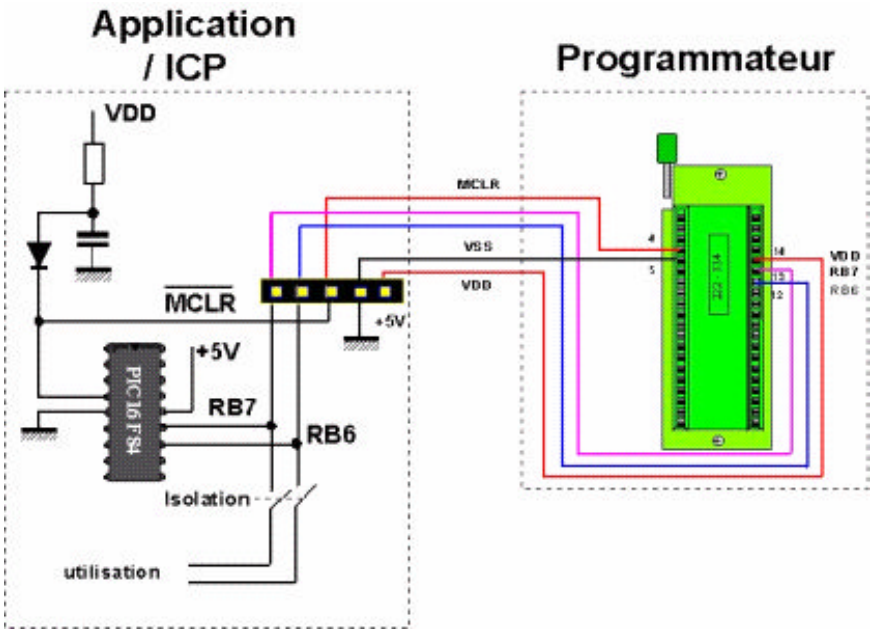


Figure 8 – Programmation in-situ

Programmateurs

Il existe de nombreux programmateurs sur le marché qu'ils soient sous forme de kit à monter ou bien déjà câblés.MICROCHIP propose entre autre le PICSTART PLUS dont l'intérêt est d'être reconnu par le logiciel MPLAB qui sert de développement, ainsi le programmeur reste sur le même environnement du début jusqu'à la fin de l'application (Figure **9**).
Si vous utilisez un programmateur fait "maison" ou bien en kit (Figure **10**) il vous faudra un logiciel permettant de transférer le fichier compilé vers la mémoire du PIC, le logiciel le plus populaire est sans doute ICPROG (Figure **11**), vous pouvez télécharger une version de ce logiciel sur le site WWW.IC-PROG.COM.

Lors de l'achat du programmateur préférez les programmateurs pilotés sur port série ou USB, sachant que le port parallèle est de moins en moins utilisé (surtout avec les versions récentes de WINDOWS).

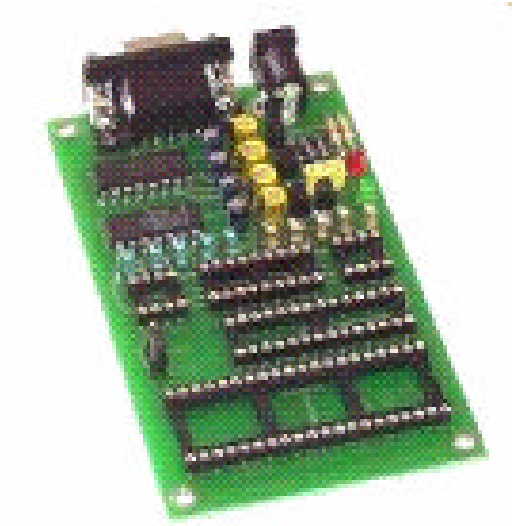


Figure 10 – Programmateur de PIC

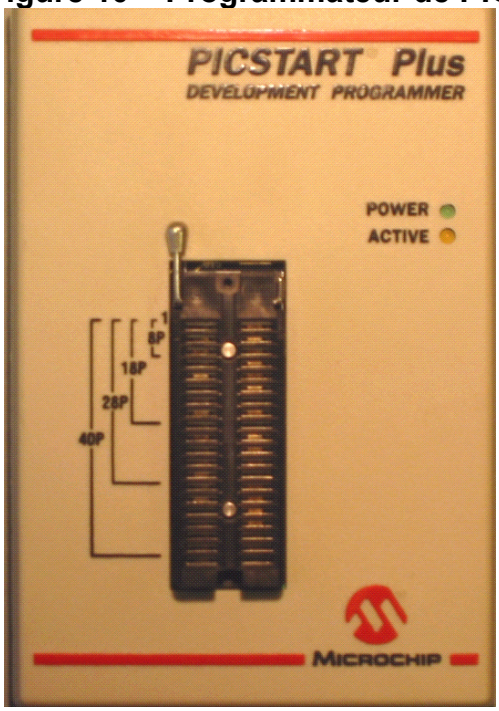


Figure 9 – Le PICSTART PLUS

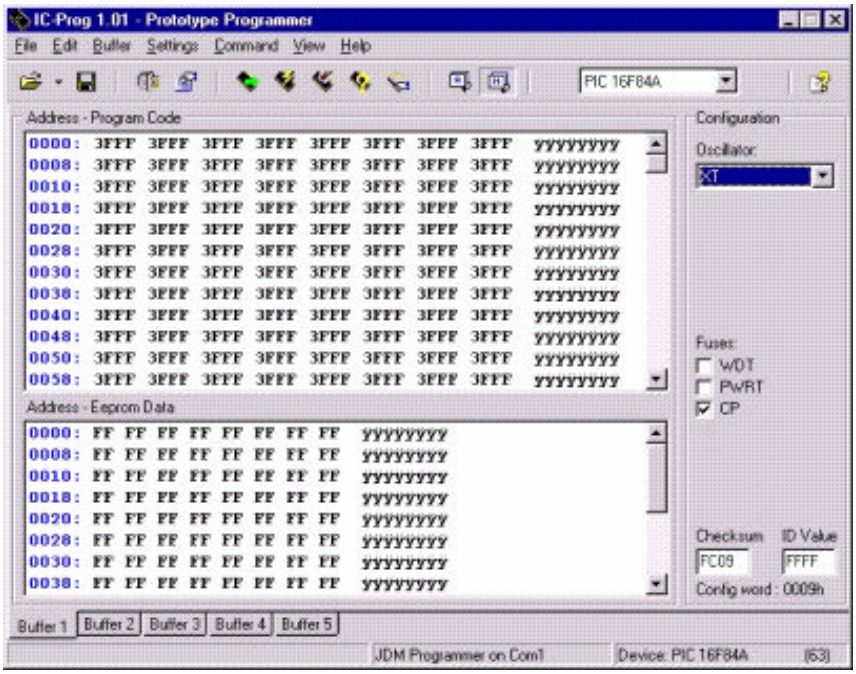


Figure 11 Vue du logiciel ICPROG

Utilisation du logiciel ICPROG

Le logiciel ICPROG possède une interface très conviviale, en quelques clicks de souris nous arrivons rapidement à transférer le fichier compilé vers le PIC.

Pour procéder à une programmation il faut dans un premier temps:

- définir la partie "Hardware", si vous possédez un programmateur de type "série" allez dans le menu "Settings" puis "Hardware" et sélectionnez "JDM PROGRAMMER" dans la liste déroulante puis "DIRECT I/O" si vous travaillez sous WIN 9x (**Figure 12**)ou bien "Windows API" pour les versions de Windows XP. Enfin il faut bien sûr sélectionner le port de communication "COM1" par exemple.

  Si vous possédez un programmateur sur port parallèle sélectionnez "Propic2 programmer" puis "Lpt1" pour le port. Attention certains signaux sont peut être inversés selon le type de programmateur, dans ce cas il faudra cocher les cases correspondantes pour les cinq signaux (MCLR, DATA in, DATA out, Clock , VCC), le mieux consiste bien sûr à s'appuyer sur la documentation du programmateur, en général la configuration de base est indiquée.

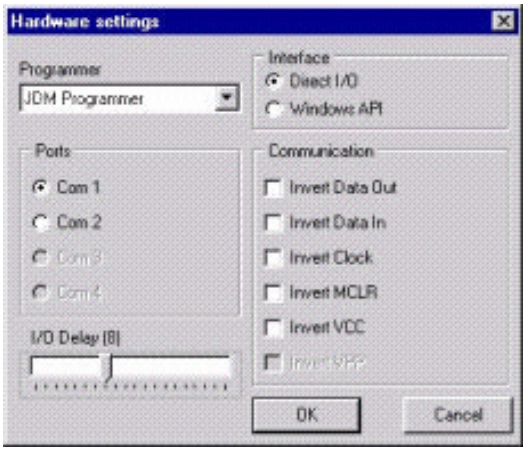


Figure 12 – Configuration du "HARWARE"

- sélectionner le type de composant à programmer, pour ce faire allez dans le menu "Settings" puis "Device" puis "Microchip PIC" et enfin le type de PIC à programmer. Vous noterez au passage les possibilités du logiciel à programmer de nombreux composants tel que les mémoires I2C.

- depuis le menu "File" puis "Open file" sélectionnez votre fichier .hex à transférer vers la mémoire du PIC. En cliquant dans le menu "View" puis "Assembler" vous pouvez visualiser le code en assembleur.

- Vous pouvez si vous n'avez pas déclarer la variable _CONFIG dans votre programme source redéfinir les "fusibles de configuration" dont le rôle nous l'avons déjà vu dans une précédente leçon consiste à choisir pour le PIC 16F84 (il existe d'autre fusibles pour certains PIC, vous pouvez les visualiser en sélectionnant un autre PIC dans la liste) un mode pour l'horloge, la protection pour la relecture, la temporisation au démarrage ainsi que l'utilisation du watchdog. La configuration consiste à cocher ou non les cases situées sur la partie gauche de l'écran (frame "configuration") et de sélectionner un type d'horloge dans la liste déroulante.

- Cliquez ensuite dans le menu "Command" puis "Programm all" pour lancer le transfert du fichier .hex vers la mémoire du pic. Pour la vérification vous pouvez soit lancer en manuel une fois la programmation effectuée ce mode en cliquant dans le menu "Command" puis "Verify" ou bien sélectionner une vérification automatique en fin ou en cours de programmation depuis le menu "Settings" puis "Options" et l'onglet "Programming" (**Figure 13**).

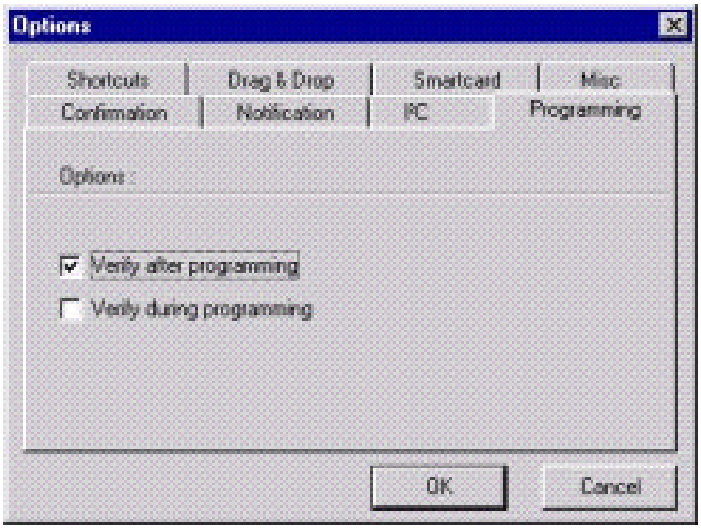
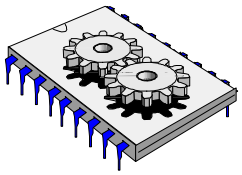


Figure 13 – Modes de vérification

Pour conclure ce chapitre

Avec cette septième partie nous avons enfin téléchargé notre programme vers la mémoire du PIC. Sans oublier la partie simulation qui peut être bien sûr approfondie afin de bénéficier des possibilités d'apprentissage du fonctionnement interne des regsitres .
Vous trouverez sur le site de l'auteur une platine à réaliser pour tester vos programmes ainsi q'un jeu de lumière pour PIC 16F84 qui fera peut être qui sait sensation pour les fêtes de fin d'année.



A la découverte des microcontrôleurs PIC
Huitième partie

Nous allons dans ce numéro aborder l'étude des interruptions sur le microcontrôleur PIC. Comme nous le verrons tout au long de cet article les interruptions vont nous offrir des possibilités de gestion d'un programme par rapport à un événement extérieur.

Imaginez vous tranquillement dans votre fauteuil en train de regarder votre film préféré que vous venez de louer dans un vidéo club et soudain le téléphone retentit. Le premier réflexe qui vous viendra à l'esprit sera sans doute de mettre le magnétoscope en mode pause afin de ne rien manquer du film merveilleux que vous suivez et d'aller répondre à l'appel téléphonique. Une fois votre conversation terminée vous revenez devant votre petit écran et vous remettez le magnétoscope en mode lecture. Nous pourrions dire que vous avez été interrompu pendant votre occupation, vous avez traité le plus urgent.

Figure 1 – L'appel téléphonique est plus prioritaire

Pour le programme d'un microcontrôleur qui se déroule "tranquillement" le principe en est le même.

Une tâche de fond s'exécute, celle-ci peut être assimilée à votre programme principal, par exemple vous faites clignoter une led à une fréquence de 1 Hz .

Soudain l ' utilisateur appui sur le bouton "Fréquence rapide" (c'est notre appel téléphonique) , que va faire votre programme ?

en fait il fait comme vous avez fait précédemment, c 'est à dire que quelque soit l 'endroit ou l 'on se situe dans le programme (cela correspond au film que vous visualisez dans l'exemple imagé) , il exécute le plus urgent, en fait il détecte l 'appui sur le bouton et enregistre cette information .

Ensuite l'appui étant mémorisé le programme "revient" à l'endroit ou il s'était arrêté auparavant et la led clignote plus vite (une fois que vous avez répondu au téléphone vous remettez le magnétoscope en lecture). Pour terminer cette introduction on peut dire qu'une interruption est une action extérieure obligeant un programme à traiter un sous programme prioritaire. Une fois ce sous programme terminé le microcontrôleur revient au programme principal à l'endroit ou il l'a quitté avant l'action extérieure.

On dit également que le programme est "dérouté" vers un sous programme d'interruption.

Il existe sur le PIC de la famille 16F84 plusieurs types d'événements pouvant provoquer une interruption ils sont au nombre de quatre, sur d'autres PIC tel que le PIC16F877 on trouvera 15 sources d'interruption, ces sources d' IT sont en rapport bien sûr avec les différentes fonctionnalités ainsi que le nombre de registres internes propres à chaque PIC:

Pour le PIC 16F84 on aura une interruption sur :

- Un changement d'état sur la broche RB0
- Un changement d'état sur une des broches RB4 à RB7
- La fin de l'écriture en E²PROM
- Le débordement du Timer interne

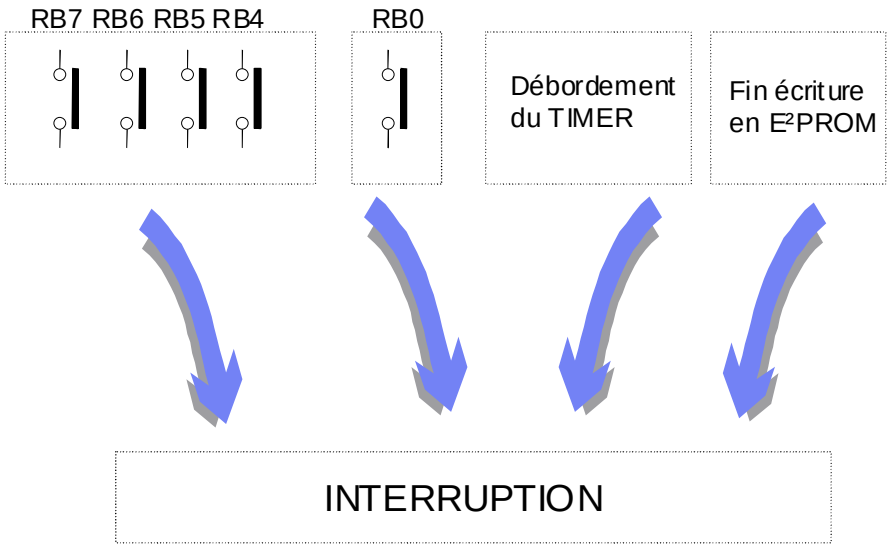


Figure 2 – Quatre types d'interruptions possibles sur le PIC 16F84

Comme nous venons de le voir quatre sources différentes d'événement peuvent déclencher une interruption, le programmeur (c'est à dire vous) aura la possibilité

de valider telle ou telle source d'interruption. Cette possibilité se définie dans un registre du PIC, le registre INTCON (INTerrupt CONtrol). Dans ce registre interne de huit bits il existe quatre bits permettant d'autoriser une ou plusieurs sources d'Interruption, trois bits permettant d'indiquer d'ou vient l'interruption, ainsi qu'un bit permettant de valider globalement les interruptions.

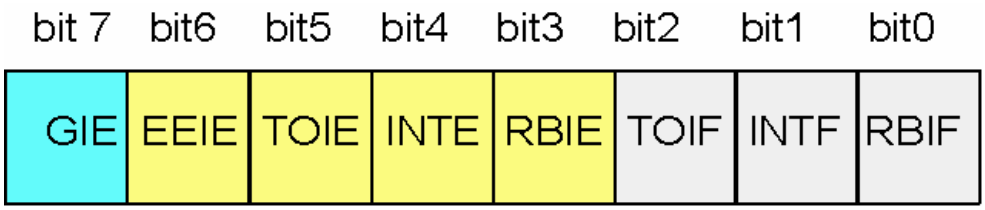


Figure 3 – Vue du registre INTCON

Vous l'avez peut être remarquez, le nom des bits est significatif, les noms se terminant par F (comme flag) sont les indicateurs, les noms se terminant par E (enabled) sont les bits permettant de valider la source d'interruption.

Rôle des bits du registre INTCON

Bit GIE (Global Interrupt Enabled bit)

Ce huitième bit du registre permet de valider les interruptions, c'est une validation générale, cela signifie que si ce bit est positionné à "1" alors le programme principal pourra être dérouté par une interruption, dans le cas contraire ("0") alors aucun événement ne pourra provoquer une IT (interruption), toutes les sources susceptibles de dérouter le programme sont donc inhibées .

Bit EEIE (Eeprom write complète Interrupt Enabled bit)

Ce bit permet de valider une interruption issue de la fin d'écriture en E²PROM. Si ce bit est positionné à "1" alors à chaque fois que vous allez écrire en E²PROM et à condition que le bit GIE vu précédemment soit également positionné à "1" une interruption sera générée et le programme principal sera dérouté, cela peut par exemple permettre de savoir si l'écriture E²PROM s'est bien déroulée.

Bit TOIE (Timer 0 Interrupt Enabled bit)

Si ce bit est positionné à "1" alors un débordement du TIMER interne provoquera une interruption. Le TIMER interne est un registre de huit bits qui s'incrémente au rythme de l'horloge. Dès que la valeur de ce registre atteint 255 et passe à 0 et si le bit GIE vu précédemment est également positionné à "1" alors une IT est générée et le programme principal est dérouté vers le sous programme d'interruption.

Bit INTE (INTerrupt pin Enabled bit)

Ce cinquième bit valide une interruption suite au changement d'état sur la broche RB0. La broche RB0 peut changer d'état soit sur un front montant, soit sur un front descendant, cet option se définie dans le registre OPTION à l'aide du 6eme bit (INTEDG). Comme pour les autres sources d'interruption il faut bien sûr que le bit GIE soit également à "1".

Bit RBIE (RB port change INTerrupt Enabled bit)

Ce quatrième bit permet de valider une interruption suite au changement d'état sur une des broches RB4,RB5,RB6 ou RB7.. Comme pour les autres sources d'interruption il faut bien sûr que le bit GIE soit également positionné à "1".

Avant de poursuivre le rôle des bits du registre INTCON, nous allons parler très brièvement de l'adresse de notre sous programme d'interruption. En effet depuis le début du cours nous avons appris que le programme principal se déroutait mais sans savoir exactement à quelle adresse mémoire. En fait pour toutes les sources D'IT il n'y à qu'une seule adresse, c'est l'adresse 04 cela signifie que pour n'importe quelle source d'Interruption (débordement TIMER, changement d'état sur la broche RB0, ou bien RB4-RB7 ou bien encore une fin d'écriture en E²PROM) le programme se "branche" à l'adresse 04 en mémoire.
Voilà vous l'avez compris les flag (les 3 derniers bits du registre INTCON) vont renseigner l'utilisateur sur l'origine de la source d'IT responsable du déroutement du programme, sans eux nous ne pourrions savoir d'ou provient l'interruption. Dans le sous programme nous pourrons alors tester ces différents bits et provoquer tel ou tel traitement selon la source d'IT.

Bit TOIF (Timer 0 Interrupt Flag bit)

Ce bit du registre INTCON sera à "1" lorsqu'une interruption sera issue du débordement du Timer interne.

Attention ce bit n'est pas remis à zéro automatiquement, c'est au programmeur (à vous…) de réaliser cette action (dans le sous programme d'IT bien sûr).

Bit INTF (INTerrupt pin Flag bit)

Ce bit du registre INTCON sera à "1" lorsqu'une interruption sera issue du changement d'état de la broche RB0.

Attention comme cité précédemment ce bit n'est pas remis à zéro automatiquement, c'est au programmeur (à vous…) de réaliser cette action (dans le sous programme d'IT bien sûr).

Bit RBIF (poRt B Interrupt Flag bit)

Ce bit du registre INTCON sera à "1" lorsqu'une interruption sera issue du changement d'état d'une des broches RB4 à RB7.

Comme cité précédemment il est possible de tester ce bit et celui-ci n'est pas remis à zéro automatiquement, c'est au programmeur (à vous…) de réaliser cette action (dans le sous programme d'IT bien sûr).

Vous l'avez peut être remarqué il n'y a pas d'indicateur (flag) sur la fin d'écriture en E²PROM, mais sachant que nous avons les trois autres indicateurs, il est facilement concevable de faire la déduction. En fait si nous utilisons les quatre sources d'interruption il faudra faire trois tests de flags (RBIF,INTF,TOIF) et si aucun d' entres eux n'est à "1" alors l'interruption proviendra de la fin d'écriture en EPROM. Un exemple de programme illustrant ces tests sera fournit en fin d'article.

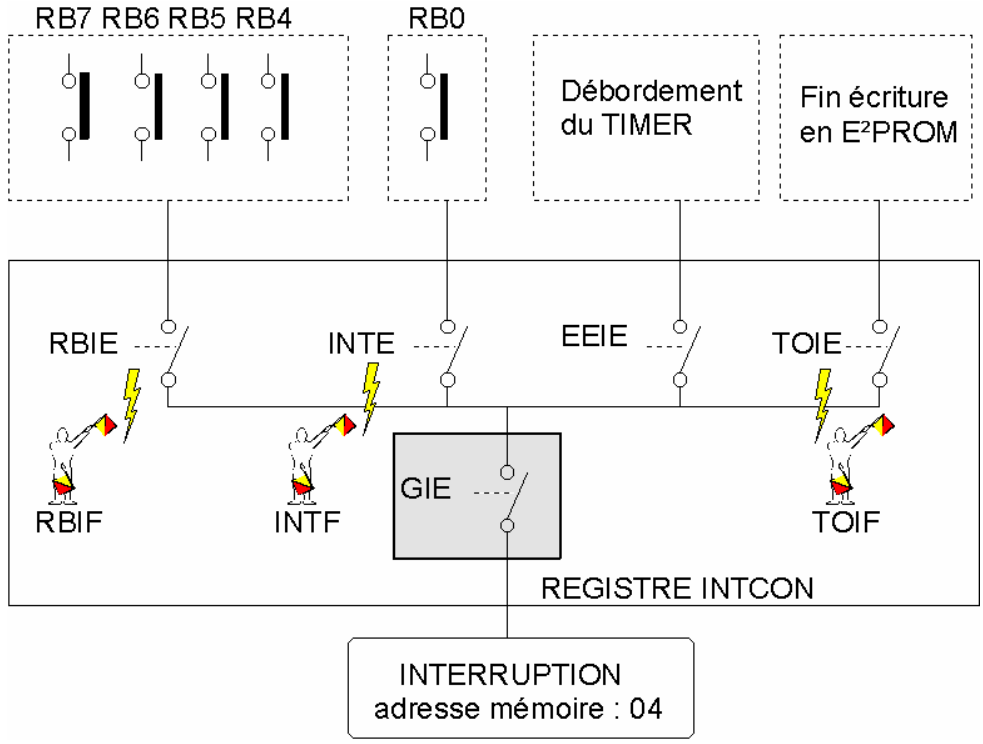


Figure 4 – Vue générale du cheminement d'une Interruption

Intégration dans un programme de la gestion d'une interruption

Le programme commencera toujours par les éternelles directives d'assemblage tel que nous l'avons étudié dans les précédentes leçons, celles-ci ne seront pas détaillées.
Comme vous le savez le programme d'IT se trouvera toujours à l'adresse 4 en mémoire, donc il faudra commencer par déclarer l'adresse de début du programme principal (voir fichier source **Figure 5**) avec une instruction du type :

ORG 0	'adresse 0 (dès alimentation du PIC ou RESET)
Goto init	'on saute au programme principal qui commencera à l'étiquette "init"

Permettant de faire un saut au programme principal grâce à l'instruction "GOTO" dès la mise sous tension du PIC (vecteur Reset).
Dans notre programme principal nous allons déclarer si besoin est la direction des registres TRISA et TRISB et également le positionnement des bits du registre INTCON qui comme nous l'avons vu va permettre de configurer les sources d'IT.

Dans le fichier source fournit (**Figure 5**) nous utilisons le basculement de la broche RB0 sur un front montant pour provoquer l'IT, nous allons donc déclarer le type de déclenchement sur la broche RB0. Le déclenchement sur front montant se déclare dans le registre interne OPTION le bit à positionner se nomme "INTEDG", il faut le mettre à "1" pour avoir une détection sur front montant, c'est ce que réalise l'instruction suivante :

bsf OPTION_REG,INTEDG ; Détection d'un front montant sur RB0

Passons maintenant à la description du registre de gestion des interruptions, le registre INTCON. Nous devons positionner à "1" le bit GIE pour autoriser les interruptions en général puis nous devons également mettre à "1" le bit "INTE" pour valider la source d'IT en provenance de la broche RB0. Voici les deux instructions qui permettent de configurer le registre INTCON :

bsf INTCON,INTE ; on autorise l ' IT sur RB0
bsf INTCON,GIE ; on autorise les Interruptions
Voilà désormais tout front montant en provenance de RB0 sera prit en compte.

Passons maintenant au sous programme d'interruption. Il faut dans un premier temps passer l'adresse du sous programme, la directive ORG 4 se chargera de réaliser cette opération, puis vient ensuite les instructions qui seront exécutées dans cette routine d'interruption.

ORG 4

```
bsf PORTB,7          ; on allume la led connectée sur rb7

bcf INTCON,INTF      ; on remet à 0 le bit du registre d' IT qui est passé à 1

RETFIE               ; retour d 'interruption
```

Comme déjà précisé il ne faut pas oublier de remettre le flag concernant l'a provenance de l'interruption (ici le bit INTF puisque nous avons choisit de travailler que sur une seule source d'IT avec la broche RB0) à zéro, puis quelque chose de fondamental, le retour d'interruption avec l'instruction "RETFIE" afin que le programme se repositionne à l'endroit du programme principal ou à eu lieu l'événement (l'interruption). Ce retour d'IT repositionnera dans notre exemple le PIC en mode SLEEP (voir Figure 5).

Dans les précédentes leçons nous avions vu que l'adresse courante contenu dans le compteur de programme (PC) au moment de l'interruption est sauvegardé dans la pile puis est restitué grâce à l'instruction RETFIE (Figure 6). Notez bien que seul le compteur de programme est sauvegardé automatiquement, pour sauvegarder par exemple le registre de travail W et le registre STATUS ce qui peut être optionnel dans certain cas, il faudra insérer les instructions correspondantes au début du programme d 'IT, en restaurant ces informations en fin de sous programme d'IT. Voici un exemple de sauvegarde de ces deux registres :

Nous avons déclaré auparavant deux variables nommées w_temp et status_temp

L'utilisation de l'instruction SWAPF permet de n'affecter aucun flag du registre d'état, ainsi on peut le sauvegarder ce registre sans le modifier auparavant ce qui parait évident.

En début de sous-programme d'IT

```
movwf  w_temp          ; sauvegarde du registre W dans w_temp
swapf  STATUS,w        ; sauvegarde du registre status dans w
movwf status_temp ; sauvegarde du registre status dans status_temp
```

instructions du sous-programme d'IT

en fin de sous-programme d'IT avant RETFIE :

```
swapf  status_temp,w   ; on remet l' ancien status dans w
movwf STATUS           ; restauration du registre status
```

```
swapf  w_temp,f          ; Inversion L et H de l'ancien W
swapf   w_temp,w   ; Restauration du registre de travail W
```

RETFIE

Test de la source d'Interruption

Dans le cas ou vous voulez utiliser les quatre sources d'IT (en positionnant à "1" les bits RBIE,INTE,EEIE et TOIE correspondants du registre INTCON) et que vous souhaitez savoir d'ou provient l'interruption afin de réaliser des traitements différents, il faudra faire un test des flags d'interruption ((RBIF,INTF,TOIF).

Voici un exemple de test de ces indicateurs :

```
        btfsc   INTCON,T0IE     ; tester si interrupt timer autorisée
        btfss   INTCON,T0IF     ; oui, tester si interrupt timer en cours
        goto    intsw1          ; non test suivant
        call    inttimer        ; oui, traiter interrupt timer
        bcf     INTCON,T0IF     ; effacer flag interrupt timer
        goto    restorereg      ; et fin d'interruption
                                ; SUPPRIMER CETTE LIGNE POUR
                                ; TRAITER PLUSIEURS INTERRUPT
                                ; EN 1 SEULE FOIS

intsw1
        btfsc   INTCON,INTE     ; tester si interrupt RB0 autorisée
        btfss   INTCON,INTF     ; oui, tester si interrupt RB0 en cours
        goto    intsw2          ; non sauter au test suivant
        call    intrb0          ; oui, traiter interrupt RB0
        bcf     INTCON,INTF     ; effacer flag interupt RB0
        goto    restorereg      ; et fin d'interruption
                                ; SUPPRIMER CETTE LIGNE POUR
                                ; TRAITER PLUSIEURS INTERRUPT
                                ; EN 1 SEULE FOIS

intsw2
```

```
        btfsc   INTCON,RBIE     ; tester si interrupt RB4/7 autorisée
        btfss   INTCON,RBIF     ; oui, tester si interrupt RB4/7 en cours
        goto    intsw3          ; non sauter
        call    intrb4          ; oui, traiter interrupt RB4/7
        bcf     INTCON,RBIF     ; effacer flag interupt RB4/7
        goto    restorereg      ; et fin d'interrupt
                                ; SUPPRIMER CETTE LIGNE POUR
                                ; TRAITER PLUSIEURS INTERRUPT
                                ; EN 1 SEULE FOIS

intsw3
        BANK1                   ; passer banque1
        btfsc   INTCON,EEIE     ; tester si interrupt EEPROM autorisée
        btfss   EECON1,EEIF     ; oui,tester si interrupt EEPROM
        goto    restorereg      ; non sauter
        call    inteep          ; traiter interruption eeprom


                        ;restaurer registres
                        ;-------------------
restorereg
        swapf status_temp,w     ; swap ancien status, résultat dans w
        movwf   STATUS          ; restaurer status
        swapf   w_temp,f        ; Inversion L et H de l'ancien W
                                ; sans modifier Z
        swapf   w_temp,w        ; Réinversion de L et H dans W
                                ; W restauré sans modifier status
        retfie          ; return from interrupt

;*****************************************************************
;               INTERRUPTION TIMER 0                    *
;*****************************************************************
inttimer
        return                  ; fin d'interruption timer
                                ; peut être remplacé par
```

```
                        ; retlw pour retour code d'erreur

;*********************************************************************
;               INTERRUPTION RB0/INT                    *
;*********************************************************************
intrb0
        return          ; fin d'interruption RB0/INT
                        ; peut être remplacé par
                        ; retlw pour retour code d'erreur

;*********************************************************************
;               INTERRUPTION RB0/RB4                    *
;*********************************************************************
intrb4
        return          ; fin d'interruption RB0/RB4
                        ; peut être remplacé par
                        ; retlw pour retour code d'erreur

;*********************************************************************
;               INTERRUPTION EEPROM                     *
;*********************************************************************
inteep
        return          ; fin d'interruption eeprom
                        ; peut être remplacé par
                        ; retlw pour retour code d'erreur
```

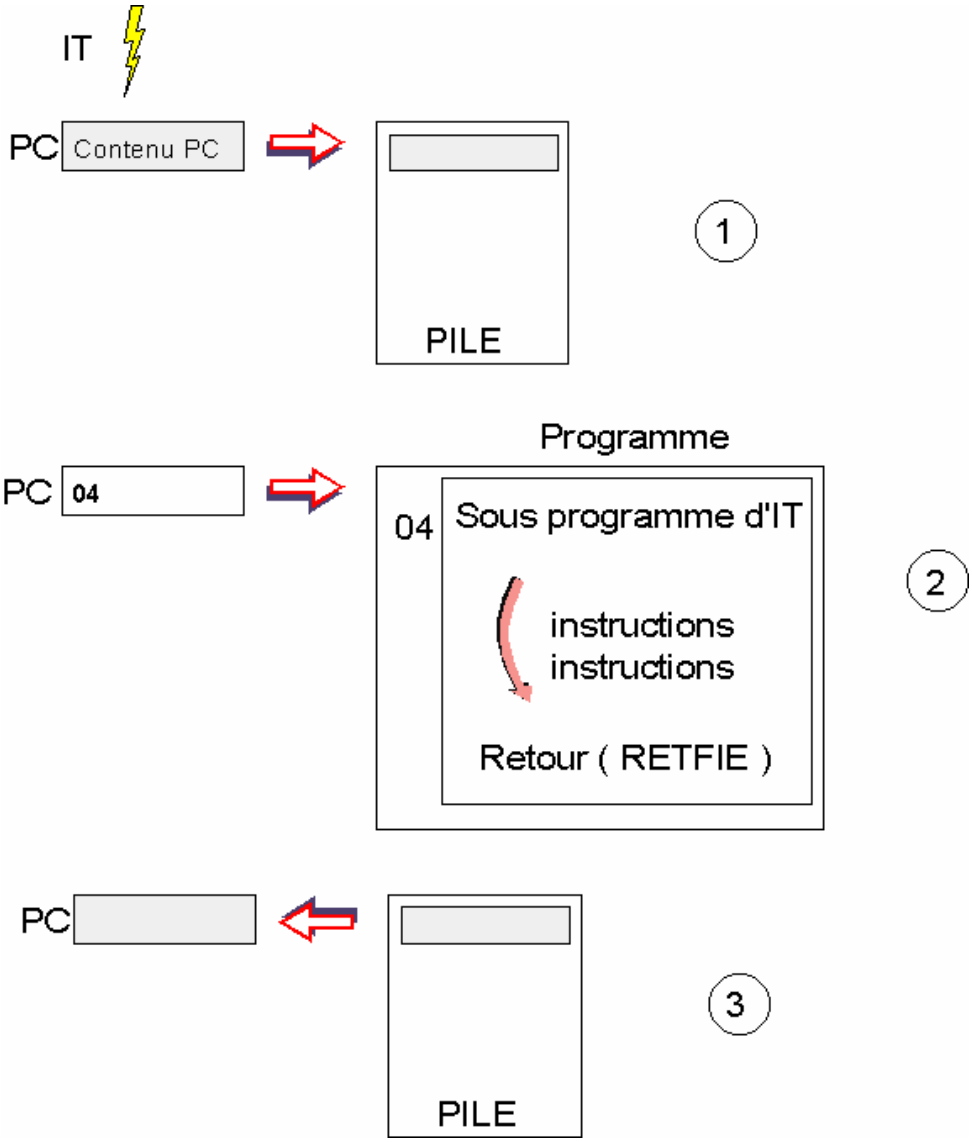
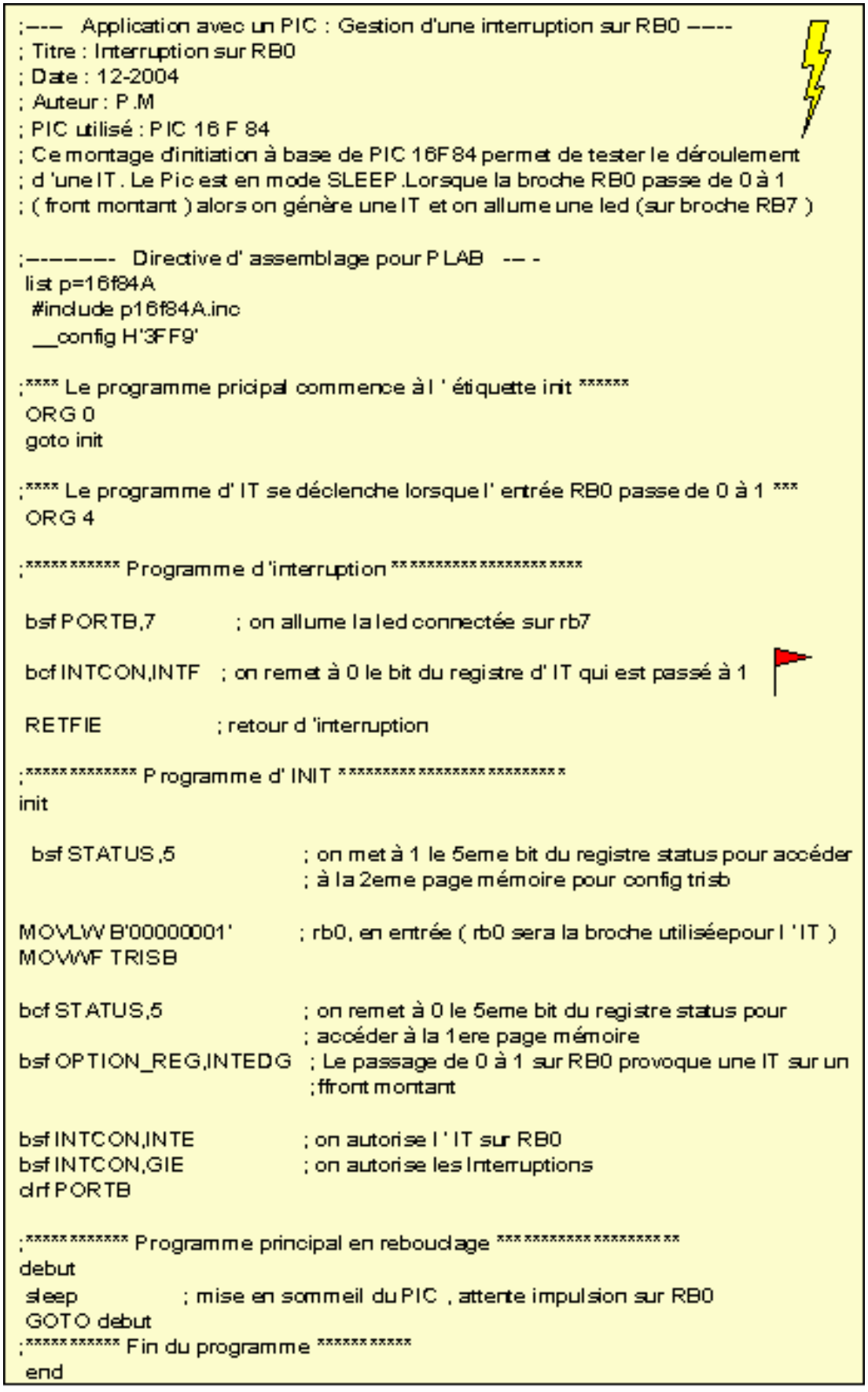


Figure 6 – Sauvegarde de l'adresse courante

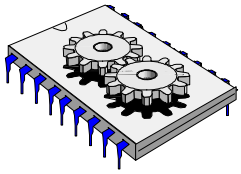


```
;---  Application avec un PIC : Gestion d'une interruption sur RB0 ----
; Titre : Interruption sur RB0
; Date : 12-2004
; Auteur : P.M
; PIC utilisé : PIC 16 F 84
; Ce montage d'initiation à base de PIC 16F84 permet de tester le déroulement
; d 'une IT. Le Pic est en mode SLEEP.Lorsque la broche RB0 passe de 0 à 1
; ( front montant ) alors on génère une IT et on allume une led (sur broche RB7 )

;--------  Directive d' assemblage pour P LAB  -- -
 list p=16f84A
  #include p16f84A.inc
  __config H'3FF9'

;**** Le programme pricipal commence à l ' étiquette init ******
 ORG 0
 goto init

;**** Le programme d' IT se déclenche lorsque l' entrée RB0 passe de 0 à 1 ***
 ORG 4

;************ Programme d 'interruption **********************

 bsf PORTB,7          ; on allume la led connectée sur rb7

 bcf INTCON,INTF   ; on remet à 0 le bit du registre d' IT qui est passé à 1

 RETFIE              ; retour d 'interruption

;************** Programme d' INIT ************************
init

  bsf STATUS,5              ; on met à 1 le 5eme bit du registre status pour accéder
                            ; à la 2eme page mémoire pour config trisb

MOVLW B'00000001'       ; rb0, en entrée ( rb0 sera la broche utiliséepour l 'IT )
MOVWF TRISB

bcf STATUS,5              ; on remet à 0 le 5eme bit du registre status pour
                          ; accéder à la 1ere page mémoire
bsf OPTION_REG,INTEDG  ; Le passage de 0 à 1 sur RB0 provoque une IT sur un
                          ;ffront montant

bsf INTCON,INTE           ; on autorise l ' IT sur RB0
bsf INTCON,GIE            ; on autorise les Interruptions
clrf PORTB

;************ Programme principal en rebouclage ********************
debut
 sleep             ; mise en sommeil du PIC , attente impulsion sur RB0
 GOTO debut
;************ Fin du programme ***********
 end
```

Figure 5 – programme type de gestion d'une IT déclenchée par RB0

Pour conclure ce chapitre

Avec cette huitième partie nous avons abordés les interruptions et leur rôle primordial dans un programme, si vous avez bien compris cette introduction aux IT il vous sera facile d'appréhender d'autres microcontrôleurs PIC dans lesquels le principe reste identique.
Cette utilisation des interruptions tel que nous venons de voir est présente dans notre vie courante, ne serait ce que lorsque l'on frappe sur les touches d'un clavier d'ordinateur…



A la découverte des microcontrôleurs PIC
Neuvième et dernière partie

Nous allons dans ce numéro aborder l'étude du TIMER et du WATCHDOG ces deux blocs fonctionnels faisant partie intégrante du microcontrôleur PIC .

Le Timer du PIC 16F84 est en fait un compteur 8 bits cadencé au rythme de l'horloge interne (issue bien sûr de la fréquence quartz divisée par quatre). En effectuant une lecture de ce compteur nous pouvons ainsi déterminer le "temps qui s'écoule" et faire des actions à des moments précis. Une fois ce compteur configuré (ce que nous allons voir par la suite), il est possible d' 'initialiser celui-ci avec une valeur comprise entre 0 et 255.

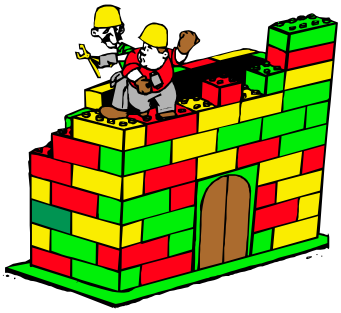


Figure 1 – Le TIMER et le WATCHDOG pour finir…

Synoptique interne du TIMER

Le Timer est composé principalement d'un Prédiviseur , d'une entrée de comptage externe, d'un compteur ainsi que d'un système d'aiguillage piloté par des bits de configuration (**Figure 2**).

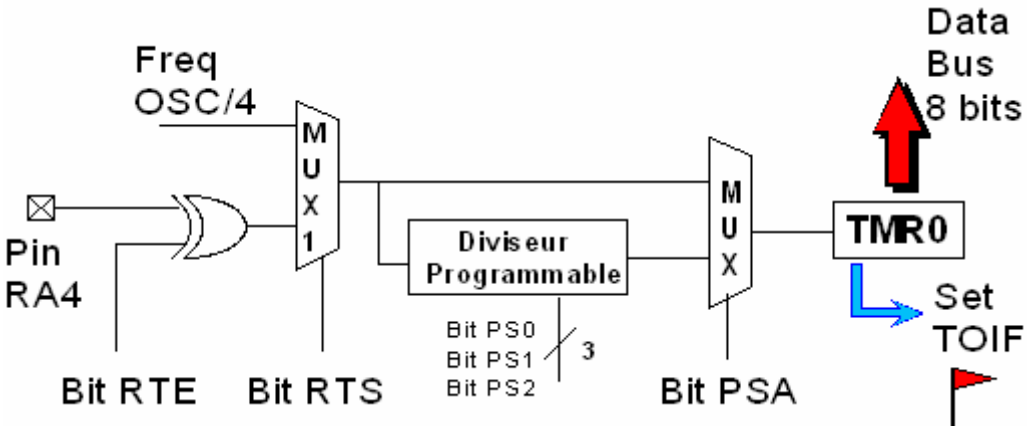


Figure 2 – Synoptique interne simplifié du TIMER

Mise en œuvre et principe de fonctionnement

Comme vous le visualisez sur la **Figure 2** et **Figure 2a** l'entrée de comptage peut être issue :

- soit d'une fréquence appliquée sur la broche RA4 on parlera alors de mode "COUNTER".
- soit de la fréquence du quartz divisée par 4 (le PIC divise lui même par 4 la fréquence du quartz, dans le cas d'un quartz de 4 Mhz nous aurons une période de 1 µs), se sera alors le mode "TIMER".

Le Bit RTS permet de sélectionner la source de comptage, si ce bit est à 0 alors en sortie de multiplexeur (MUX1) nous aurons le signal Freq OSC/4, si ce bit est à "1" alors on retrouvera le signal provenant de la broche RA4 (**Figure 3**). Il est également possible d'inverser le signal issu de la broche RA4, c'est le bit RTE qui est chargé d'effectuer cette opération, si celui-ci est à "0" alors en sortie du ou exclusif nous aurons le signal provenant de la broche RA4 , dans le cas contraire nous aurons le signal RA4 inversé (**Figure 3**).

Enfin le bit PSA permet de sélectionné ou non le prédiviseur, si ce bit est à "0" alors on retrouvera en sortie du multiplexeur (MUX) le signal provenant de la sortie de MUX1 prédivisé par un nombre programmable dans le prédiviseur (par les bits PS0,PS1,PS2), si le bit PSA est positionné à "1" alors le signal en sortie de MUX sera celui issu de la sortie de MUX1 (on ne prédivise pas dans ce cas).

Tous ces bits que nous venons de voir sont dans le registre OPTION du PIC, nous verrons bien sûr comme d'habitude un exemple pour illustrer l'utilisation du TIMER.

	1	0
RTE	Sortie Xor = $\overline{RA4}$	Sortie Xor = RA4
RTS	Sortie MUX1 = sortie Xor	Sortie MUX1 = Fosc/4
PSA	Sortie MUX = sortie MUX1	Sortie MUX = Prédiviseur

MUX : Multiplexeur
Xor : Ou exclusif

Figure 3– Bits de Configuration

Dans le cas d'utilisation du prédiviseur les trois bits nommé PS0, PS1, PS2 permettent de définir une division du signal en entrée de celui-ci. Le tableau en **Figure 4** résume les différentes possibilités de prédivision. Comme vous le voyez le taux maximum de division sera lorsque les bits PS0,PS1 et PS2 seront positionnés à "1", ce qui signifie que le signal appliqué en entrée de prédiviseur sera dans ce cas divisé par 256.

PS2	PS1	PS0	Division par :
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

Figure 4 – Détermination du taux de prédivision

La programmation des différents bits de configuration du fonctionnement du TIMER sera effectuée dans le registre OPTION dont la figure 5 rappel les bits à positionner.

7	6	5	4	3	2	1	0
		RTS	RTE	PSA	PS2	PS1	PS0

Figure 5 – Les bits à positionner dans le registre OPTION du PIC

Il est à noter qu'a chaque fois que l'on remettra à jour le TIMER ou bien que l'on modifiera un bit de configuration, alors deux cycles d'instructions seront nécessaires avant que le comptage ne commence. Si dans votre programme vous désirez lire la valeur du compteur, cela ne perturbera pas le comptage.

Génération d'interruption par le TIMER

Comme nous l'avons vu dans une précédente leçon, le TIMER va pouvoir générer une interruption en passant le flag TOIF du registre INTCON à 1 et ceci dès que le compteur passera de 255 à 0. Si vous avez autorisé les interruptions générales (bit GIE) et si vous avez autorisé les interruptions sur le Timer (bit TOIE) alors le programme se déroute à l'adresse 04 tel que nous l'avons précédemment détaillé (**Figure 6**).

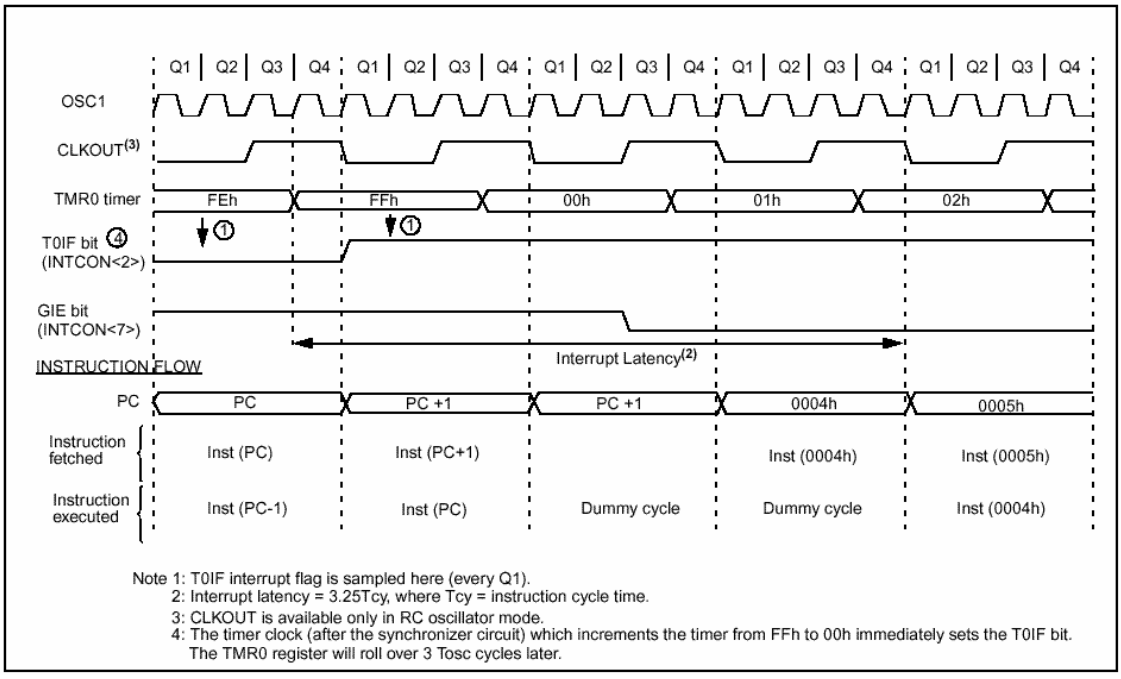


Figure 6 – Le mode interruption avec le TIMER

Pour illustrer le fonctionnement et l'utilisation du TIMER nous vous proposons de réaliser un programme qui aura pour but de faire clignoter une led à une fréquence issue de l'utilisation du TIMER (**Figure 7**).

Détail du programme :

Nous passons rapidement sur les déclarations ainsi que les directives que nous avons déjà commentées lors de précédentes leçons. Pour réaliser une temporisation de huit secondes par exemple (tel que notre exemple de programme) nous allons commencer par définir les bits du registre OPTION.
Les lignes suivantes : MOVLW 0x07 et MOVWF OPTION_REG ont pour effet de positionner le registre OPTION à la valeur 7 soit en binaire 0000 0111. Si nous reprenons les explications sur les bits du registre OPTION on s'aperçoit que :
le bit RTS = 0 donc on utilise la fréquence du quartz / 4 ,
le bit PSA = 0 donc on utilise le prédiviseur
les bits PS2,PS1,PS0 = 1 donc on prédivisera par 256 (tableau Figure 4)
Si on prédivise par 256 la fréquence du quartz / 4 on aura donc si le quartz est à 4 MHz :

4 MHz / 4 = 1MHz en entrée de prédiviseur et 1MHz / 256 = 3906,25 Hz en sortie du prédiviseur.

Les prochaines instructions à expliquer sont les suivantes : MOVLW 0x06 et MOVWF TMR0 , ces instructions ont pour effet d'initialiser le compteur du TIMER avec la valeur 6.
Dans la suite de notre programme on vérifiera que le compteur du TIMER n'est pas à "0", ce qui signifie que celui –ci comptera jusqu'à 256 – 6 soit 250, donc notre signal de 3906,25 Hz sera encore divisé par 250 soit une fréquence de 15,625 Hz.

Nous utilisons ensuite un registre temporaire nommé "retard1" celui-ci est initialisé à la valeur 7D soit 125 en décimal, cette valeur de registre nous permet d'effectuer une boucle. Pour conclure nous avons un signal de 15,625 Hz que nous allons encore divisé par 125 (puisque nous allons effectuer la boucle 125 fois) dons en sortie de la temporisation nous auront compter :
15,625 Hz / 125 soit 0.125 Hz ce qui représente une période de 1 / 0.125 = 8 secondes (Figure 8). Les leds connectées sur le port B s'allumeront pendant 8 secondes et resteront éteintes également 8 secondes. Si vous avez compris le principe il est très facile de modifier les temps obtenus pour avoir d'autres temporisations.

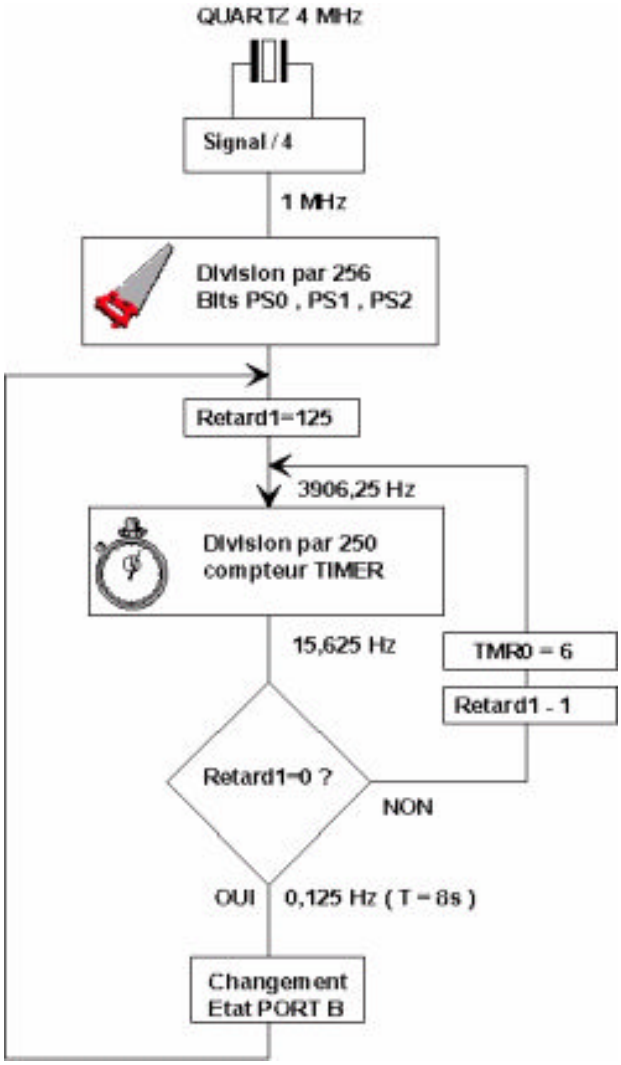


Figure 8 – Synoptique simplifié du programme

Le WATCHDOG du PIC

Le WATCHDOG ou "chien de garde" peut être assimilé à un mécanisme qui va surveiller si votre programme s'exécute toujours dans de bonnes conditions et dans le cas contraire ce processus va faire un reset du microcontrôleur.

Le fonctionnement du WATCHDOG est lié comme le Timer à un compteur interne. Ce compteur est cadencé depuis un circuit RC dont la fréquence peut évoluer :
Selon la tension d'alimentation, le PIC et bien sûr la température (**Figure 10**).
La période typique de ce circuit RC pour une alimentation de 5V est de 18mS, celle-ci peut varier entre 7 et 28 mS.
Si vous utilisez le WATCHDOG (nous verrons plus loin qu'il faut configurer un bit) et que vous lancer votre programme, alors 18 mS (période typique du WATCHDOG) plus tard le PIC se RESET. Pour éviter cela il faudra dans votre programme insérer la ligne "**clrwdt"** (clear WATCHDOG) qui aura pour effet de remettre à zéro le compteur du WATCHDOG avant expiration du délai programmable (nous le verrons

par la suite). Si cette instruction n'est pas exécutée , par exemple si votre programme est "planté" ou si celui-ci est dans une boucle infinie… alors le PIC sera redémarré à l'adresse de RESET (adresse 0) et votre programme redémarre donc.
Il est à noter qu'un bit du registre STATUS (le bit TO) est remis à zéro dès que le délai du WATCHDOG à expiré, cela permet au démarrage du PIC de tester d'ou vient le RESET (mise en route de l'utilisateur ou bien redémarrage sur activation WATCHDOG).
Pour conclure le WATCHDOG est comme un compte à rebours, si celui-ci n'est pas arrêté à temps (instruction clrwt) alors le PIC redémarre .

Sur le synoptique de la **Figure 9** on peut voir que la période du WATCHDOG peut être allongée en utilisant le diviseur que nous avons déjà détaillé pour le TIMER. Sachant que ce diviseur est commun une utilisation pour le WATCHDOG interdira l'utilisation pour le TIMER et vice - versa. Le bit PSA fera l'aiguillage pour l'utilisation du diviseur, si ce bit est à "0" le diviseur sera utilisé pour le TIMER et si ce bit est à "1" alors ce sera pour le WATCHDOG.
Attention dans le cas de l'utilisation du WATCHDOG le taux de division du prédiviseur est différent par rapport au TIMER. Le tableau de la **Figure 11** reprend ce taux. Comme vous le voyez la période max (typique pour une alimentation de 5V) sera de 18mS x 128 = 2304 mS soit 2,3 seconde.
Un programme utilisant le WATCHDOG est proposé en **Figure 12,** ce programme allume une led au démarrage puis environ 2,3 secondes après le PIC se reset . Pour vérifier le bon fonctionnement de la RAZ du watchdog il suffira d'insérer dans la boucle "Goto debut" l'instruction …
Clrwdt (cette instruction est en commentaire dans le programme **Figure 12**).
Dans le programme proposé vous remarquerez que pour utiliser la WATCHDOG il faut déclarer le fusible correspondant par la directive :
_CONFIG _WDT_ON
Pour utiliser le prédiviseur il faut mettre le bit PSA du registre OPTION à 1 ainsi que les bits PS0-PS1-PS2 pour avoir la période la plus longue offerte par le WATCHDOG (division par 128).

PS2	PS1	PS0	Division par :
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Figure 9 – Synoptique du TIMER et du WATCHDOG

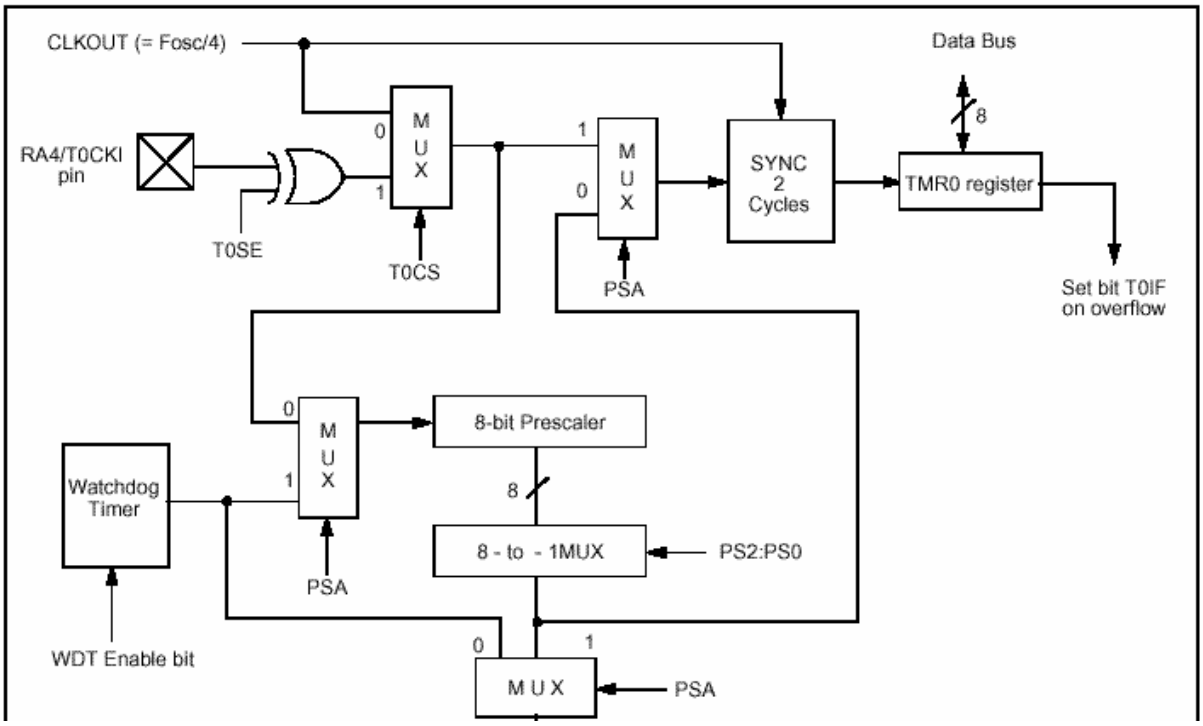


Figure 11 – Le taux de pré division selon PS0-PS1-PS2

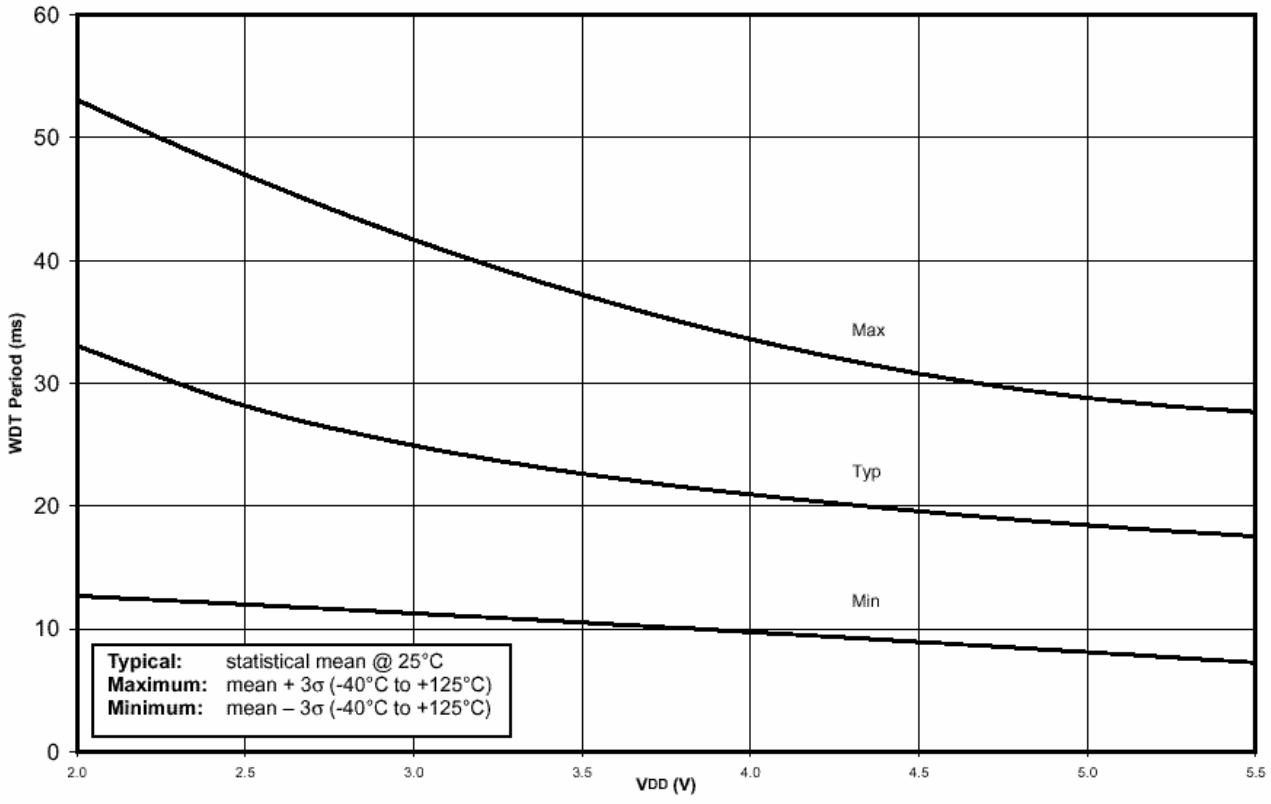

60
50
40
30
20
10
0
WDT Period (ms)
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)
Max
Typ
Min
Typical: statistical mean @ 25°C
Maximum: mean + 3σ (-40°C to +125°C)
Minimum: mean – 3σ (-40°C to +125°C)

Figure 10

```
;-----  Exemple d'application avec un PIC : Essai sur le WATCHDOG  -----
; Titre : WATCHDOG
; Date : 01 - 2005
; Auteur : P.M
; PIC utilisé : PIC 16 F 84
; essai du WATCHDOG période typique = 2.2 seconde
; le montage fonctionne avec un Quartz de 4 MHz

;----------  Directive d' assemblage pour PLAB  ----------
list p=16f84A
#include p16f84A.inc
__config _CP_OFF & _WDT_ON & _PWRTE_ON & _HS_OSC

;----------  Init des ports A et B  ----------
 ORG 0
 bsf STATUS,5          ; on met à 1 le 5eme bit du registre status pour accéder
                       ; à la 2eme page mémoire ( pour trisa et trisb et OPTION )
 MOVLW 0x00            ; on met 00 dans le registre W
 MOVWF TRISB           ; on met 00 dans le port B il est programmé en sortie

 MOVLW 0x0F                ; on met 0F dans le registre W
 MOVWF OPTION_REG          ; on met 0F dans le registre OPTION
                           ; PSA= 1 PS2=1 PS1=1 PS0=1

 bcf STATUS,5          ; on remet à 0 le 5eme bit du registre status pour accéder
                       ; à la 1eme page mémoire
 COMF PORTB            ; on change d'état les leds du port B

;----------  Programme principal  ----------
debut
   clrwdt              ; pour mettre à "0" le compteur du watchdog
                       ; en commentaire pour essayer
   goto debut

END
```

Figure 12 – Le programme du WATCHDOG

Pour conclure ce dernier chapitre

Avec cette ultime partie nous terminons cette découverte des microcontrôleurs PIC, nous espérons que ces leçons vous auront donné entière satisfaction et surtout vous auront donné envie d'en savoir plus sur ces composants qui méritent que l'on s'attarde un peu pour en comprendre le fonctionnement . Vous pouvez télécharger les fichiers sources sur le site de l'auteur.

P.MAYEUX

http://p.may.chez-alice.fr