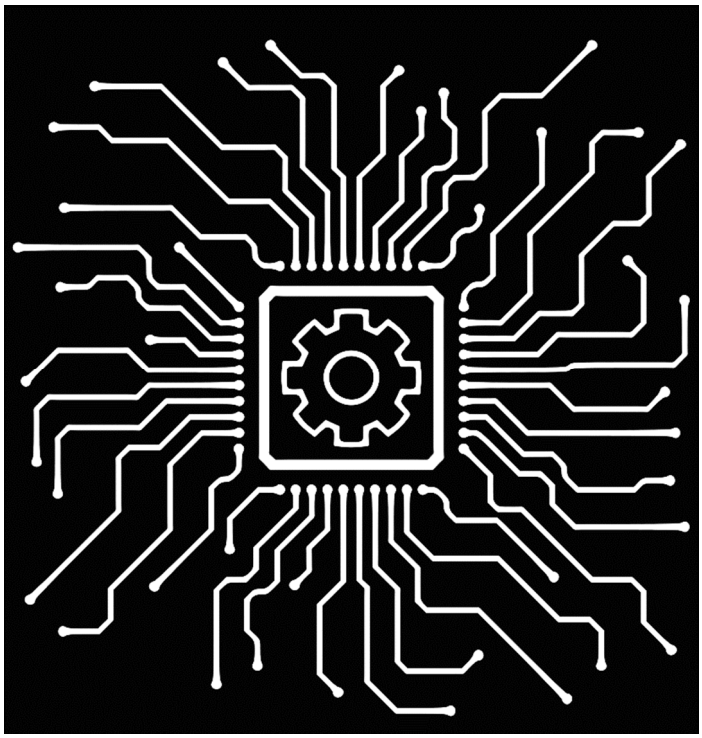
الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Aboubakr BELKAÏD – Tlemcen –
جامعة أبي بكر بلقايد – تلمسان –
Faculté de Technologie
كلية التكنولوجيا

Support de Cours et Travaux Dirigés

Microprocesseurs et Microcontrôleurs

(Programmation en mikroC. Application pour les microcontrôleurs de la famille PIC)

Filière de Génie Biomédical



Elaboré par :

Dr HAMZA CHERIF Lotfi

Année universitaire : 2018 – 2019

Avant-propos

Apparus dès la création des premiers circuits intégrés numériques, au début des années 1970, les microprocesseurs constituent le cœur de presque toutes les réalisations électroniques ; on en trouve dans tous les domaines, notamment : l'informatique (de la calculatrice à l'ordinateur), l'automobile (ABS, injection, ...), l'automatique (automates programmables, contrôle de processus, ...), l'électronique domestique (thermomètre, télécommande, carte à puce, ...)

Les performances des microprocesseurs sont liées aux possibilités offertes par la technologie, en terme de capacité (nombre de portes logiques intégrées) et de vitesse, et au choix d'architectures adaptées (ou imposées pour cause de compatibilité ascendante); à l'heure actuelle, on trouve sur le marché des microprocesseurs intégrant des millions de transistors, fonctionnant à plus de 3000 MHz et disposés dans des boîtiers de plusieurs centaines de broches, ils sont issus de différentes approches architecturales : CISC, RISC, DSP et VLIW.

En effet, la gestion des ressources internes d'un microcontrôleur impose de faire appel à de nombreuses fonctions nouvelles, spécifiques de ce type de circuit. En outre, si vous voulez pouvoir mettre au point votre application avec un maximum de chances de succès, le recours à un environnement de développement spécialisé, allant de l'éditeur de programme au simulateur en passant par le compilateur, est nécessaire.

Enfin, lorsque votre programme est prêt à être essayé en « vrai grandeur », il faut encore savoir comment le charger en mémoire du microcontrôleur.

Cet ouvrage se propose de répondre à toutes ces questions avec des exemples d'applications sur des microcontrôleurs de type PIC en langage C.

Ce document constitue le support du cours et ne prétend donc ni à l'originalité, ni à l'exhaustivité. Ces notes doivent beaucoup aux emprunts faits à de nombreux ouvrages et à différents travaux de collègues. Ce document s'adresse aux étudiants Master1 (Master en instrumentation Biomédicale, département de Génie Biomédical, Faculté de technologie, Université de Tlemcen.

Table de matière

Liste des figures

Liste des Tableaux

Chapitre 1 : Généralités sur les systèmes micro-programmés

1.1 Introduction

L'informatique, contraction d'information et automatique, est la science du traitement de l'information. Apparue au milieu du 20ème siècle, elle a connu une évolution extrêmement rapide. A sa motivation initiale qui était de faciliter et d'accélérer le calcul, se sont ajoutées de nombreuses fonctionnalités, comme l'automatisation, le contrôle et la commande de processus, la communication ou le partage de l'information [1].

-**Les circuits spécialisés ou ASIC** (Application Specific Integrated Circuit) : ce sont des circuits spécialisés dès leur conception pour une application donnée. Avec des avantages comme la rapidité, la Consommation moindre et l'Optimisé pour une application. Ils présentent aussi des inconvénients : Faible modularité, Possibilité d'évolution limité et le Coût.

Exemples :
DSP (Digital Signal Processing), co-processeur arithmétique, processeur 3-D, contrôleur de bus, ...

Figure 1.1 : Digital Signal Processor [1]

-**Les systèmes en logique programmée** et/ou en logique programmable sont connus sous la désignation de **PLD** (programmable logic device, circuit logique programmable). Un circuit logique programmable, ou réseau logique programmable, est un circuit intégré logique qui peut être reprogrammé après sa fabrication. Il est composé de nombreuses cellules logiques élémentaires pouvant être librement assemblé. Avantages : Forte modularité et Rapidité ; Inconvénients : Mise en œuvre plus complexe et Coûts de développement élevé.

Exemple :
- FPGA (field-programmable gate array, réseau de portes programmables in-situ),
- PAL (programmable array logic, réseau logique programmable),

Figure 1.2 : FPGA de Xilinx (modèle Spartan XC3S400) avec 400 000 portes.

-Les systèmes micro-programmés : Ce sont typiquement des systèmes micro-programmés. Un micro-contrôleur est un : Circuit intégré comprenant essentiellement un microprocesseur, ses mémoires, et des éléments personnalisés selon l'application. (Arrêté français du 14 septembre 1990 relatifs à la terminologie des composants électroniques). Avantages : Mise en œuvre simple et Coûts de développement réduits ; Inconvénients : Plus lent et Utilisation sous optimale.

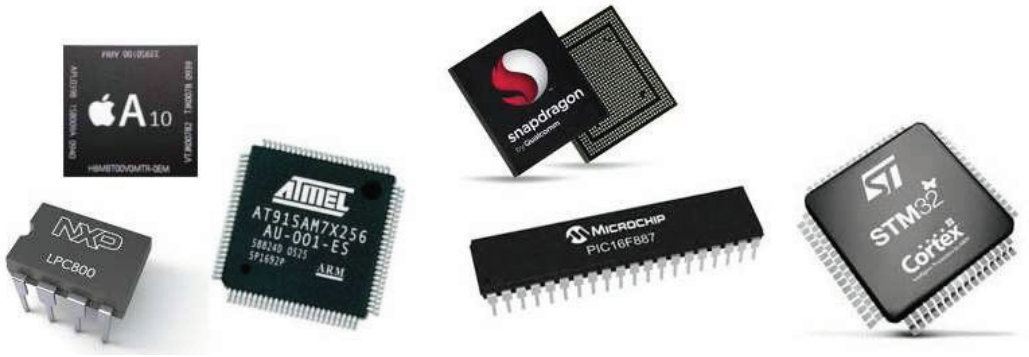


Figure 1.3 : Microcontrôleurs.

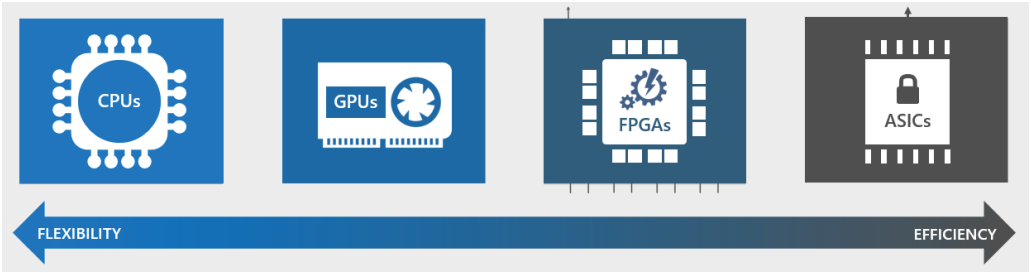


Figure 1.4 : La flexibilité par rapport à l'efficacité des différents systèmes programmables

L'usage des microprocesseurs était plutôt réservé à un public averti d'ingénieurs sachant les interfacer avec différents circuits périphériques (eprom, ram…) et programmer en assembleur.

Au fil du temps on a vu apparaître de nouveaux circuits regroupant dans une seule puce le microprocesseur et ses circuits périphériques : les microcontrôleurs. On en trouve maintenant partout : automobile, lave-vaisselle, rasoir, brosse à dent électrique… [1]

1.2 Intérêt de la programmation en langage C

Heureusement, avec la montée en puissance des microcontrôleurs, on voit apparaître actuellement des compilateurs en langage C (voire même C++) qui permettent de gagner un temps considérable pour le développement et le débogage des programmes [2].

1-Le C est un langage de « haut niveau », comparé à l'assembleur, qui permet d'écrire des programmes nettement plus intelligibles et donc plus faciles à relire et corriger ou modifier.

2-Le compilateur contrôle la cohérence du code au moment de la compilation et signale bon nombre d'erreurs, qui seront autant de bogues en moins à corriger.

3- Le compilateur prend en charge la gestion d'un certain nombre de mécanismes fastidieux : par exemple, pas besoin de spécifier la page mémoire dans laquelle on veut écrire, le compilateur s'en charge.

4- De plus, certains compilateurs permettent tout de même d'accéder à des ressources de bas niveau, voir même d'insérer dans le code des portions d'assembleur. A vrai dire, à moins d'être un « pro » de l'assembleur, vous pourrez certainement créer avec un bon compilateur C un code plus propre et plus robuste, en nettement moins de temps.

Pour bien fixer les idées sur la différence de niveau de langage entre assembleur et
C, je vais donner un exemple. Soit à décrire une action simple :
« Ouvrir la fenêtre de la pièce dans laquelle vous êtes actuellement ».

En assembleur ça donnerait :

{- soulever pied droit
- avancer pied droit
- poser pied droit
- soulever pied gauche
- avancer pied gauche
- poser pied gauche
- soulever pied droit
- avancer pied droit
- ….
- Sélectionner bras gauche
- Soulever bras
- Avancer bras
- Prendre poignée fenêtre dans main gauche
- Tourner poignée de -90°
- Tirer poignée
- Etc.…}

En C ce serait plutôt : {- **Ouvrir la fenêtre**} Et c'est le compilateur qui se chargerait de traduire la fonction « ouvrir la fenêtre » en instructions élémentaires compréhensibles par le microcontrôleur.

1.3 Choix du compilateur

La programmation des microcontrôleurs se fait naturellement en langage assembleur. Microchip propose MPLAB IDE, un environnement de développement performant, téléchargeable gratuitement. Il permet d'éditer le code source (sous la forme d'un fichier texte avec extension .asm), de le simuler et de le débugger. Le compilateur fournit le code objet (fichier avec extension .HEX). Notez que l'on peut aussi programmer les microcontrôleurs en :

Langage C (mikroC) ; Pascal (Pic Micro Pascal ; mikroPascal) ; Basic (Proton DS ; mikroBasic).
-Il existe aussi des langages de programmation graphique : Flowcode ; Niple.

Les figures suivantes illustrent des exemples de compilateurs.

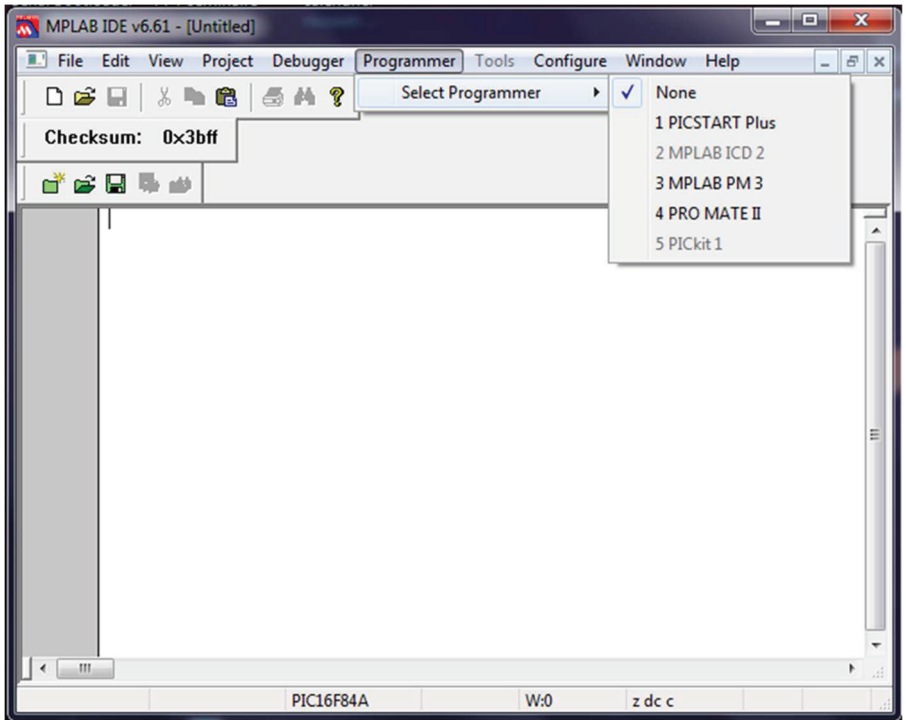


Figure 1.5 : Le compilateur MPLAB IDE de Microchip.

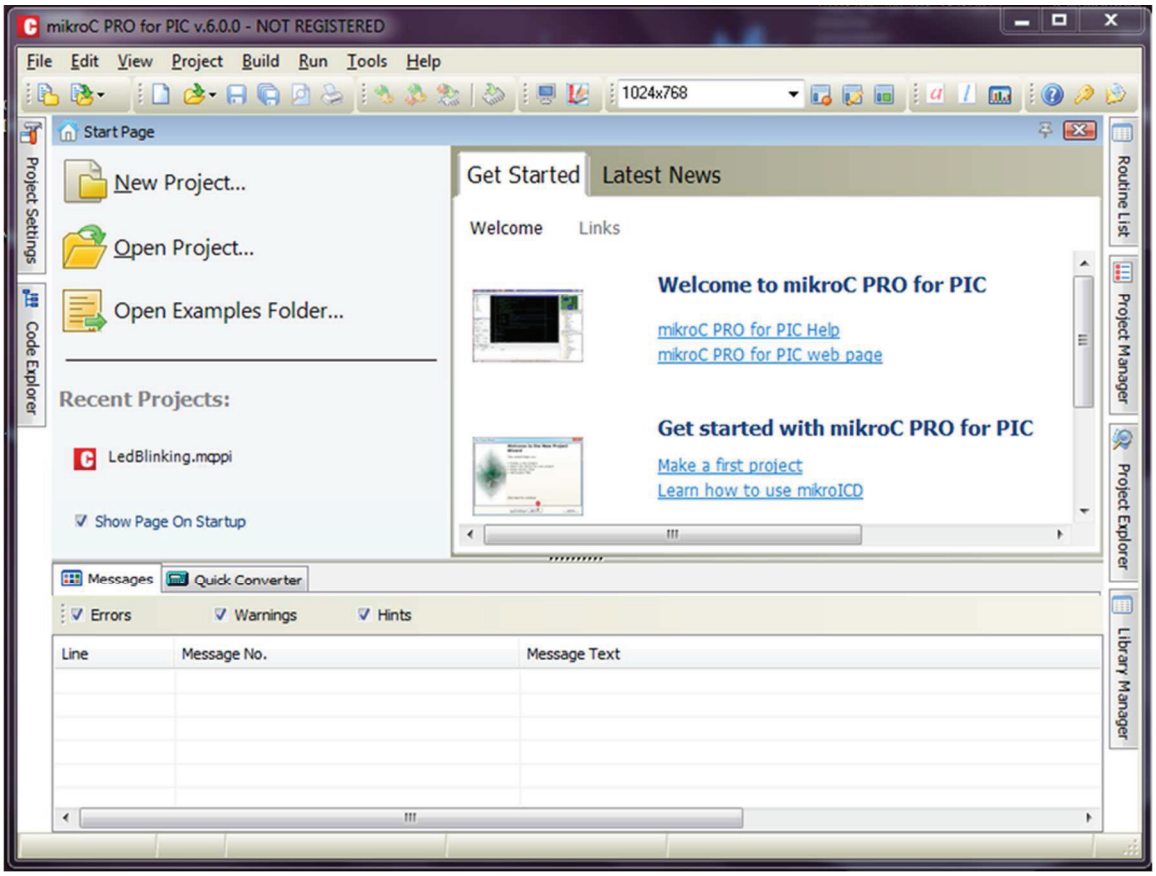


Figure 1.6 : Le compilateur MikroC Pro for PIC de Mikroeloktronika.

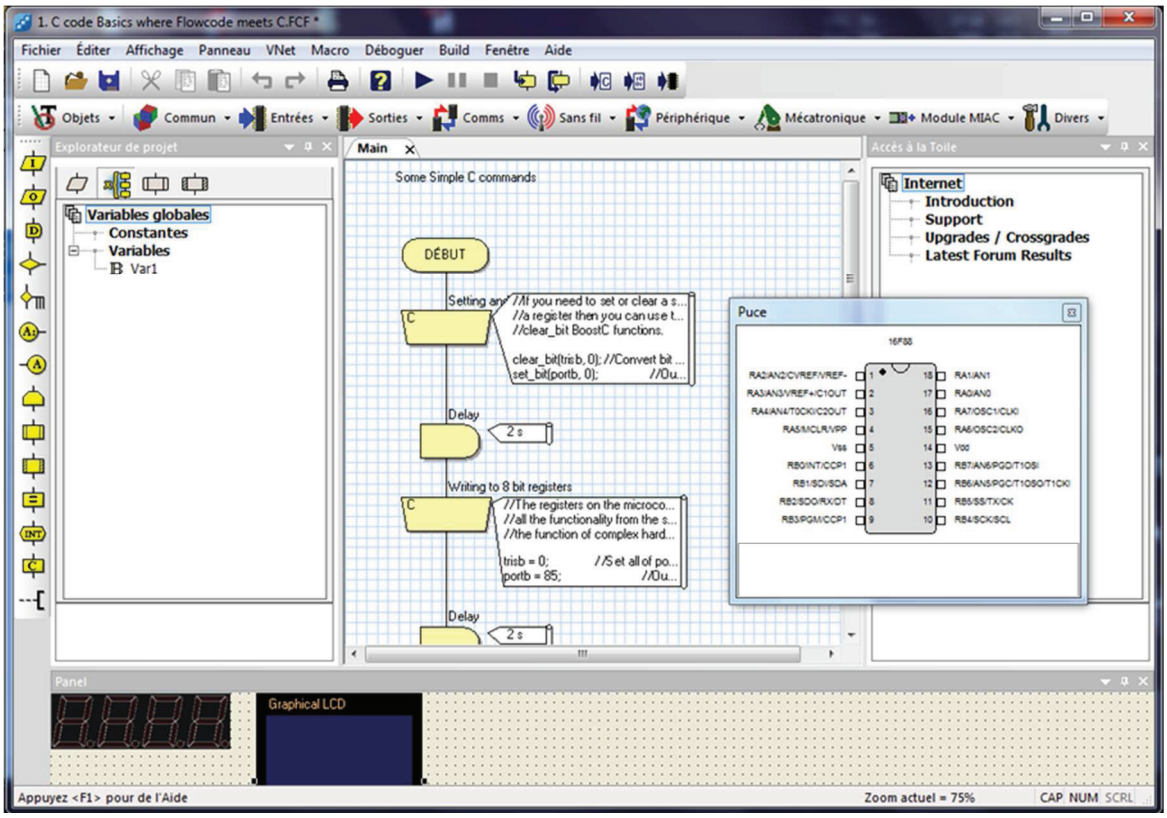


Figure 1.7 : Le compilateur Flowcode de Matrixmultimédia.

- **Le Compilateur graphique (Flowcode) [2] :**

Flowcode est un langage de haut niveau pour la programmation système PICmicro microcontrôleurs basé sur les organigrammes. Flowcode vous permet de concevoir complexe PIC fondé la robotique et les systèmes de contrôle par le dessin simplement un diagramme de votre émission souhaité en quelques minutes même sans connaissances en programmation préalable.

- Ne nécessite pas d'expérience de la programmation.
- Permet à un programme ou un code complexe en C à être conçu rapidement.
- Norme internationale utilisations diagramme de symboles (ISO5807).
- Plein écran sur la simulation permet le débogage et accélère le processus de développement.
- Facilite l'apprentissage par suite complète de démonstration des didacticiels et des systèmes virtuels (alarmes anti-vol etc).
- Produit ASM code pour une série de 18, 28 et 40 broches dispositifs.
- Permet un codage en C ou un assemblage de plusieurs codes C pour être intégré comme une macro.
- Prend en charge les interruptions et les convertisseurs A / D.

- **Fonctionnement du Flowcode :**

Flowcode est construit sur un compilateur C - C2C. Il s'agit d'un objectif général 8 / 16 bits compilateur conçu spécialement pour les appareils PICmicro. Flowcode génère un fichier de code C de l'organigramme que vous créez. Ce code est automatiquement compilés en code assembleur par le compilateur C2C et ensuite traduits en un code machine Hex fichier à l'aide de Microchip MPASM assembleur. Toute tierce partie PIC programmeur peut ensuite être utilisé pour télécharger le fichier résultant Hex dans l'objectif du PIC program mémoire.

1.4 La carte de programmation (hardware)

Il existe plusieurs marques sur le marché ; des cartes de programmation qui se branchent sur le port parallèle, le port série ou bien le port USB d'un ordinateur. Une fois le microcontrôleur brancher sur cette carte, le programme écrit sur le compilateur peut être envoyer à l'aide du logiciel pilote spécifique à cette carte (le programmateur).

Figure 1.8 : Exemples de cartes de programmation des Microcontrôleurs.

1.5 Le programmateur (software)

Le programmateur ou le logiciel pilote de la carte de programmation commande cette carte, il existe plusieurs marque sur le marché ; Exemple : IC-Prog (logiciel freeware développé par Bonny Gijzen), Serial Bootloader AN1310 (Microchip) et mikroProg Suite For PIC (Mikroelectronika) permettent le transfert du code objet (fichier .HEX) dans la mémoire Flash du microcontrôleur

.

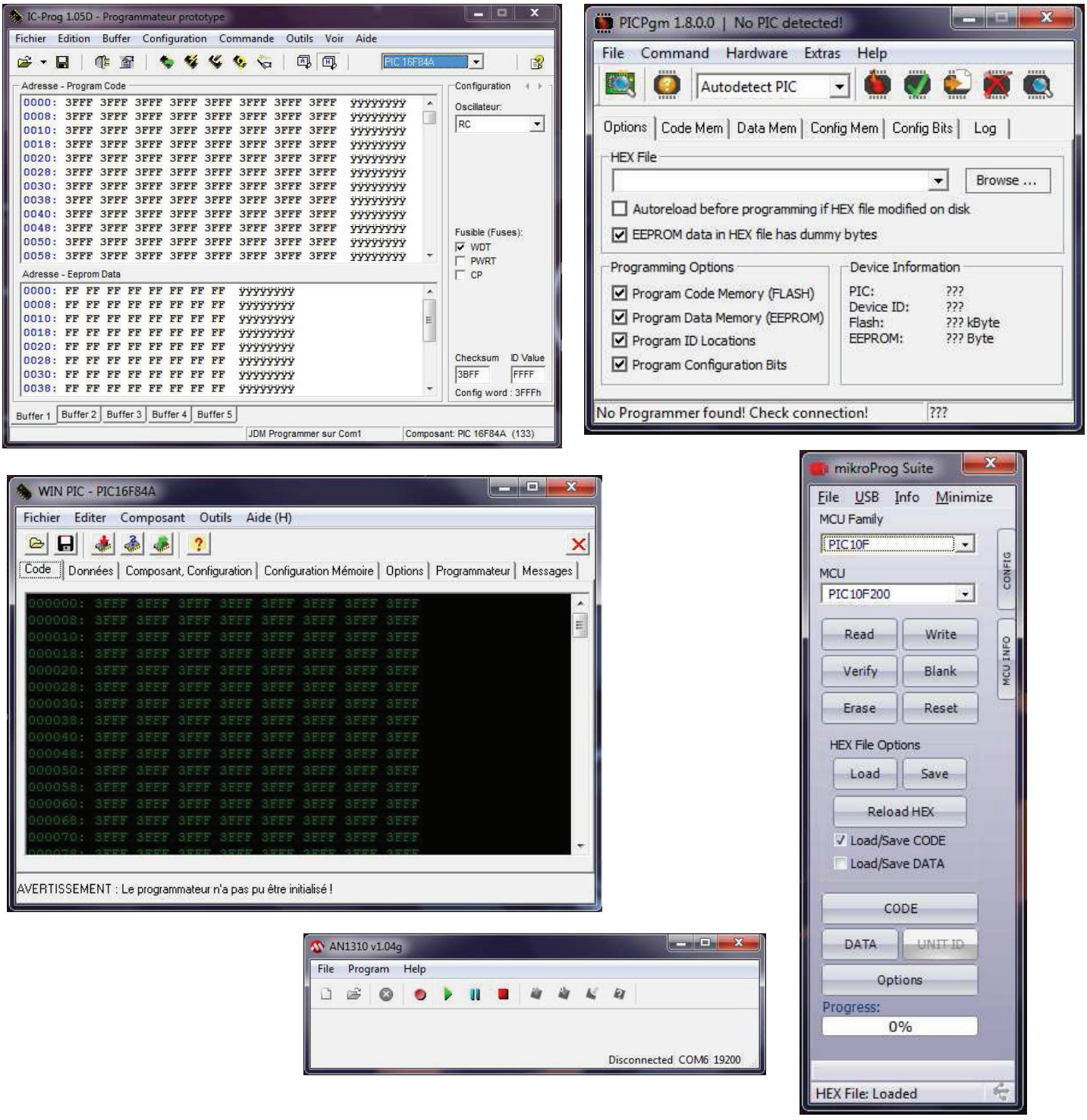


Figure 1.9 : Exemples de logiciels programmateurs.

1.5 Conclusion

La mise en œuvre de ces systèmes micro-programmés s'appuie sur deux modes de réalisation distincts : le matériel et le logiciel. Le matériel (hardware) correspond à l'aspect concret du système : unité centrale, mémoire, organes d'entrées-sorties, etc… Le logiciel (software) correspond à un ensemble d'instructions, appelé programme, qui sont contenues dans les différentes mémoires du système et qui définissent les actions effectuées par le matériel. Le chapitre suivant sera consacré à la description des microcontrôleurs.

Chapitre 2 : Du microprocesseur au microcontrôleur

2.1 Introduction

L'environnement minimal d'un système microprogrammé est constitué des éléments suivants :
-Unité Centrale de Traitement (microprocesseur).
-Mémoire principale.
-Interfaces d'entrées/sorties.

-Unité Centrale de Traitement :
L'UCT (CPU : Central Processing Unit) ou **microprocesseur** est chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Toutes les activités du processeur sont cadencées par une horloge. On caractérise le processeur par :
-Sa fréquence d'horloge : en MHz ou GHz.
-Le nombre d'instructions par secondes qu'il est capable d'exécuter.
-La taille des données qu'il est capable de traiter (en bits).

-Mémoire principale :
Elle contient les instructions du programme en cours d'exécution et les données associées à ce programme. Physiquement, elle se décompose souvent en :
-Mémoire morte (ROM : Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.
-Mémoire vive (RAM : Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.
Remarque : Les disques durs, disquettes, CDROM, etc… sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

-Interfaces d'entrées/sorties :
Les interfaces d'entrées/sorties permettent d'assurer la communication entre le µP et les périphériques (capteur, clavier, moniteur ou afficheur, imprimante, modem, etc…)

Ces éléments sont reliés par 3 bus : ils permettent de faire transiter (liaison série/parallèle) des informations codées en binaire entre deux points. Typiquement les informations sont regroupées en mots : octet (8 bits), word (16 bits) ou double word (32 bits). Autrement dit « Un bus est un jeu de lignes partagées pour l'échange de mots numériques. » [3].

-**Le bus d'adresse** qui permet au microprocesseur de sélectionner la case mémoire ou le périphérique auquel il veut accéder pour lire ou écrire une information (instruction ou donnée) ;
-**Le bus de données** qui permet le transfert des informations entre les différents éléments ; ces informations seront soit des instructions, soit des données en provenance ou à destination de la mémoire ou des périphériques ;
-**Le bus de contrôle** qui indique si l'opération en cours est une lecture ou une écriture, si un périphérique demande une interruption pour faire remonter une information au processeur, etc.

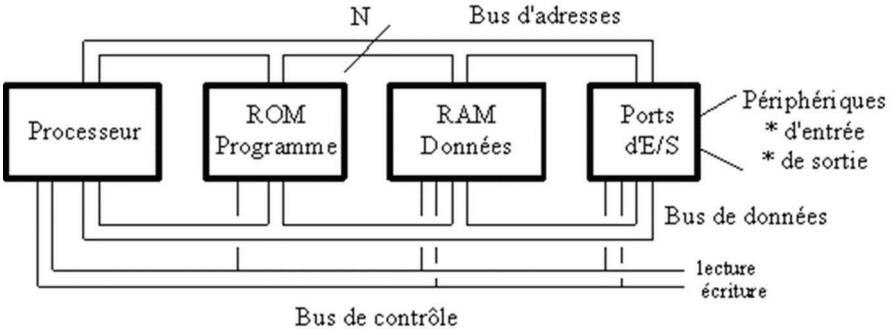


Figure 2.1 : Schéma de principe d'un système microprogrammé [4].

2.2 Le microprocesseur

Un microprocesseur est **un circuit intégré complexe**. Il résulte de l'intégration sur une puce de fonctions logiques combinatoires (logiques et/ou arithmétique) et séquentielles (registres, compteur, etc…). Il est capable d'interpréter et **d'exécuter les instructions d'un programme**. Il doit aussi prendre en compte les informations extérieures au système et assurer leur traitement. C'est le cerveau du système.

Son domaine d'utilisation est donc presque illimité. Le concept de µP a été créé par la Société **INTEL**. Créée en 1968, elle était spécialisée dans la conception et la fabrication de puces mémoire. À la demande de deux de ses clients, fabricants de calculatrices et de terminaux, INTEL étudia une unité de calcul implémentée sur une seule puce. Ceci donna naissance, en 1971, au premier µP, le 4004, qui était une unité de calcul 4 bits fonctionnant à 108 kHz. Il résultait de l'intégration d'environ 2300 transistors. A l'heure actuelle, un microprocesseur regroupe sur quelques mm^2 des fonctionnalités toujours plus complexes (Core I7 : 1,17 milliard de transistors). Leur puissance continue de s'accroître et leur encombrement diminue régulièrement respectant toujours, pour le moment, la fameuse loi de Moore (co-fondateur de la société Intel) dont l'hypothèse est le progrès technologique permettrait de multiplier par 2 tous les 18 mois le nombre de transistors intégrés sur les circuits).

La figure suivante montre l'évolution des nombres de transistors dans un microprocesseurs Intel [4].

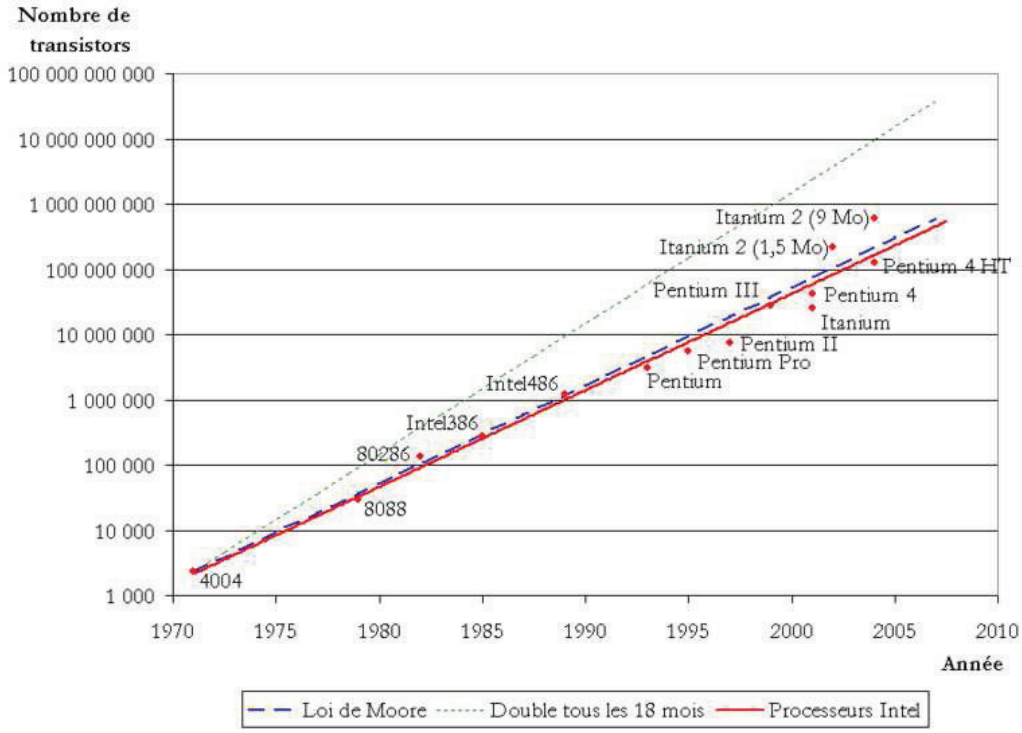
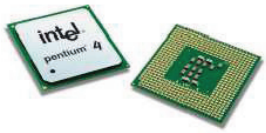


Figure 2.2 : Graphique (en ordonnées logarithmiques) illustrant la loi de Moore par rapport à l'évolution réelle du nombre de transistors dans les microprocesseurs Intel; en pointillés verts au-dessus, représentation de l'hypothèse selon laquelle ce nombre doublerait tous les 18 mois

Intel 8086 (1978)
architecture interne 16 bits
bus 16 bits
fréquence d'horloge 4,77/10 Mhz
39 000 transistors, gravés en 3µm
0,33/0,75 MIPS



Intel Pentium 4 Northwood C (2002)
architecture interne 32 bits
fréquence d'horloge 2,4/3,4 Ghz
(bus processeur : 200Mhz)
plus de 42 millions de transistors, gravés en 0,13 µm
450 MIPS

Intel Core i7 Gulftown (2011)
architecture interne 64 bits
4/6 coeurs
fréquence d'horloge 3,46 Ghz
Fréquence de bus: 3,2 GHz
Fréquence de transfert des données 25.6 Gb/sec.
1,17 Milliards de transistors, gravés en 32nm
6000 MIPS

Figure 2.3 : Evolution des Microprocesseurs

2.3 Structure interne d'un microprocesseur

La composition interne d'un microprocesseur peut être représentée par le schéma suivant :

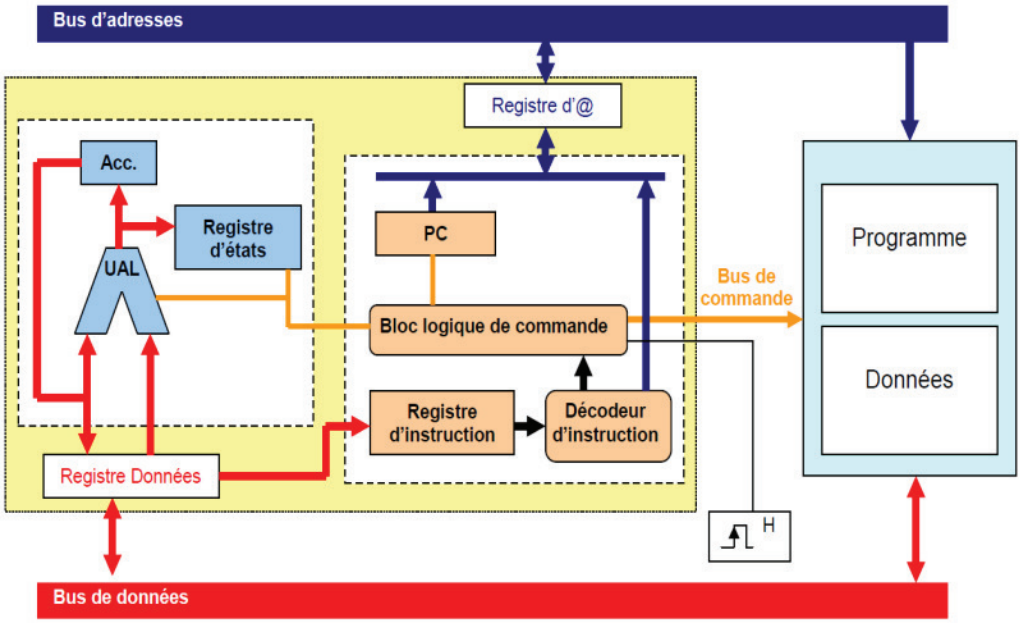


Figure 2.4 : Structure interne d'un microprocesseur [5].

Le microprocesseur est composé des deux grands blocs : **L'unité de contrôle** qui lit en mémoire un programme et donne à **l'unité de calcul** (Unité Arithmétique et Logique) la séquence des instructions à effectuer. Le microprocesseur est connecté aux **bus de communication** (données, adresses et contrôle) permettant d'accéder aux autres parties aux périphériques.

Le microprocesseur est également constitué de **registres**, qui sont des petites mémoires internes très rapides d'accès utilisées pour stocker temporairement une donnée, une instruction ou une adresse.
Chaque registre stocke 8, 16 ou 32 bits. Le nombre exact de registres dépend du type de processeur et varie typiquement entre une dizaine et une centaine. Les principaux registres sont : le registre d'adresse, le **registre de donnée**, le **registre d'état**, le **registre d'instruction** et le(s) **accumulateur**(s).

Les accumulateurs, sont utilisés pour stocker les résultats des opérations arithmétiques et logiques. Ils interviennent dans une proportion importante des instructions.
Les instructions d'un microprocesseur sont souvent nombreuses mais élémentaires. Il existe différents types d'instructions :

-**Opérations arithmétiques ou logiques** : addition, complémentation...
-**Opérations d'échanges ou de transfert** : sauver l'accumulateur, charger l'accumulateur avec une donnée....
-**Opérations liées au fonctionnement du processeur** : arrêt logiciel, masquage d'interruption...

2.4 Le microcontrôleur

C'est un ordinateur monté dans un circuit intégré. Les avancées technologiques en matière d'intégration, ont permis d'implanter sur une puce de silicium de quelques millimètres carrés la totalité des composants qui forment la structure de base d'un ordinateur. Leur prix varie de quelques Euros à une dizaine d'Euros pour les plus complexes [6].

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

2.5 Microcontrôleurs comme Composants intégrés

Un microcontrôleur est un circuit constitué, sur une seule puce, de l'environnement minimal d'un système microprogrammé. Il contient :

-Un processeur (CPU), avec une largeur du chemin de données allant de 4 bits pour les modèles les plus basiques à 32 ou 64 bits pour les modèles les plus évolués ;

-De la mémoire vive (RAM) pour stocker les données et variables ;

-De la mémoire morte (ROM) pour stocker le programme.

-Différentes technologies peuvent être employées :
EPROM, EEPROM, mémoire **flash** (la plus récente) ;

Souvent **un oscillateur** pour le cadencement. Il peut être réalisé avec un **quartz**, un **circuit RC**, des périphériques, capables d'effectuer des tâches spécifiques.

On peut mentionner entre autres :

-les **convertisseurs analogiques-numériques (CAN)** (donnent un nombre binaire à partir d'une tension électrique),
-les **convertisseurs numériques-analogiques (CNA)** (effectuent l'opération inverse),
-**les générateurs de signaux à modulation de largeur d'impulsion** (MLI, ou en anglais, PWM pour Pulse Width Modulation),
-**les timers/compteurs** (compteurs d'impulsions d'horloge interne ou d'événements externes),
-**les chiens de garde** (watchdog),
-**les comparateurs** (comparent deux tensions électriques),
-**les contrôleurs** de bus de communication (UART, I²C, SSP, CAN, FlexRay, USB, Ethernet, etc.).

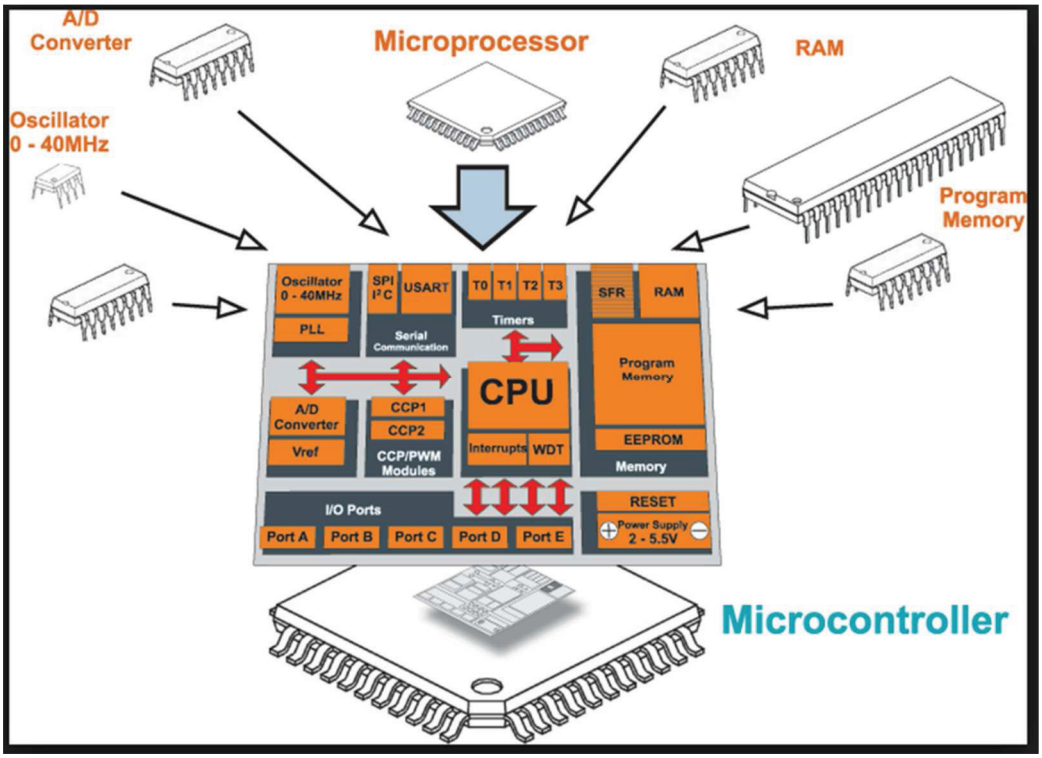


Figure 2.5 : Les composants interne d'un microcontrôleur.

2.6 Famille de microcontrôleur

Il existe un très grand nombre de modèles de microcontrôleurs de nombreux fabricants proposent chacun plusieurs familles de microcontrôleurs, comptant chacun parfois des centaines de modèles. On trouve des microcontrôleurs allant du petit circuit à 6 pattes jusqu'à des circuits comptant des centaines de pattes. Citons par exemple quelques familles :

- la famille Atmel AT91 ;
- la famille Atmel AVR (utilisée par des cartes Arduino) ;
- le C167 de Siemens/Infineon ;
- la famille Hitachi H8 ;
- la famille Intel 8051, qui ne cesse de grandir ; de plus, certains processeurs récents utilisent un cœur 8051, qui est complété par divers périphériques (ports d'E/S, compteurs/temporisateurs, convertisseurs A/N et N/A, chien de garde, superviseur de tension, etc.) ;
- l'Intel 8085, à l'origine conçu pour être un microprocesseur, a en pratique souvent été utilisé en tant que microcontrôleur ;
- le Freescale 68HC11 ;
- la famille Freescale 68HC08 ;
- la famille Freescale 68HC12 ;
- **la famille des PIC de Microchip** ;
- la famille des dsPIC de Microchip ;

- la famille des ST6, ST7, STM8, ST10, STR7, STR9, STM32 de STMicroelectronics ;
- la famille ADuC d'Analog Devices ;
- la famille PICBASIC de Comfile Technology;
- la famille MSP430 de Texas Instruments ;
- la famille 8080, dont les héritiers sont le microprocesseur Zilog Z80 (désormais utilisé en tant que contrôleur dans l'embarqué) et le microcontrôleur Rabbit ;
- la famille PSoC de Cypress ;
- la famille LPC21xx ARM7-TDMI de Philips ;
- la famille V800 de NEC ;
- la famille K0 de NEC.

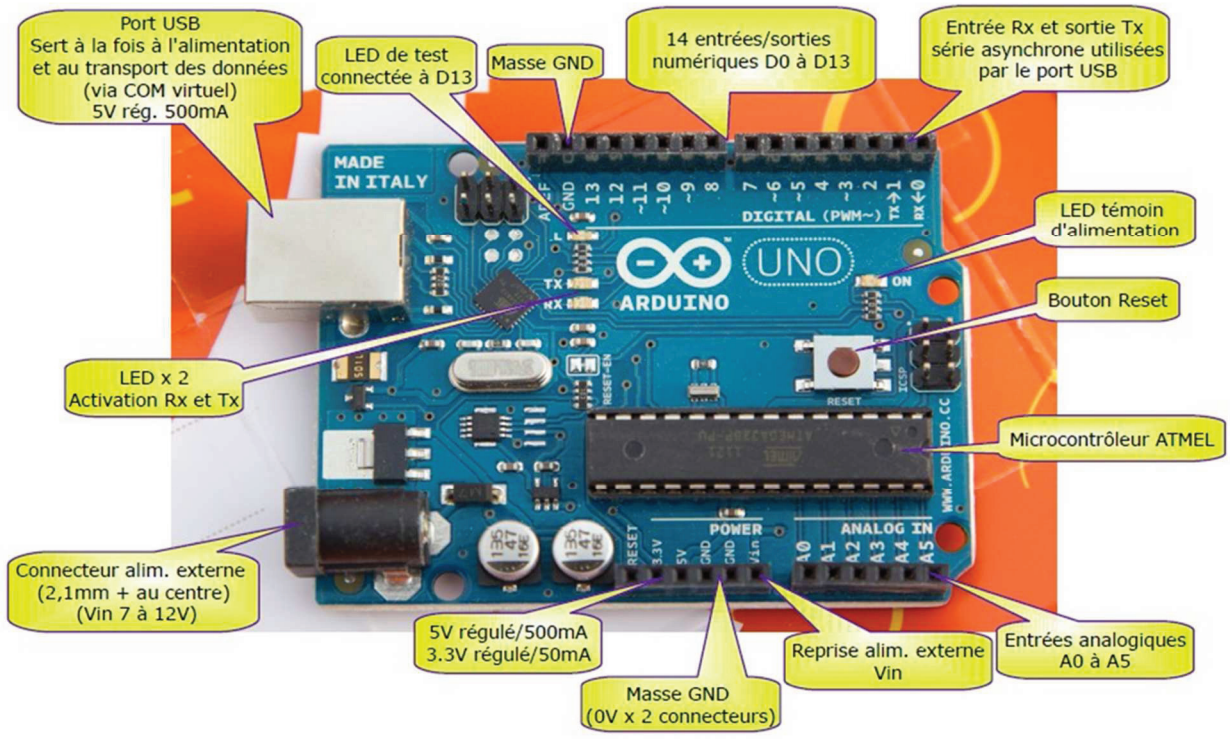


Figure 2.6 : Exemple de Microcontrôleur de la famille Atmel AVR (utilisée par des cartes Arduino).

2.7 Choix du microcontrôleur

Il y'un certain nombre de critère de choix d'un microcontrôleur par exemple [6] :

-Le nombre de pattes d'entrées-sorties.
-La Taille de la mémoire de programme (ROM)
-La Taille de la mémoire vive (RAM)
-La Consommation électrique (tension de fonctionnement, courant consommé)
-La Puissance du processeur (difficile de comparer).
-La Taille du bus d'adresses qui dépend du nombre de cellules mémoires.
-La Taille du bus de données qui influe sur la capacité de traitement du système.
-Le prix, expérience et l'environnement de développement (matériel et logiciel, cout et disponibilité).

2.8 Classifications des microcontrôleurs à deux niveaux

On peut classifier les microcontrôleurs sur deux niveaux :

►Au niveau du processeur :

– RISC (Reduced Instruction Set Computer). Nombre d'instructions réduit (sélection des instructions pour une exécution plus rapide), Décodage des instructions plus rapide.
– CISC (Complex Instruction Set Computer). Grand nombre d'instructions, Type de processeur le plus répandu

►Au niveau de l'organisation de la mémoire

– Architecture Von Neumann : une mémoire unique et pour le programme et pour les données
– Architecture Harvard : le programme et les données sont stockées dans des mémoires physiquement séparées.
La différence se situe au niveau de la séparation ou non des mémoires programmes et données.
La structure de Harvard permet de transférer données et instruction simultanément, ce qui permet un gain de performances.

Explication :
En effet, ils ont été conçus sur une architecture dite **HARVARD (RISC)** et non sur un modèle **VON NEUMANN (COMPLEX)**.
- L'architecture **VON NEUMANN** employée par la plupart des microcontrôleurs actuels (**INTEL80XX, motorola HC05, HC08 et HC11, ou ZILOG Z80**) est basée sur un bus de données unique. Celui-ci véhicule les **instructions** et les **données**.

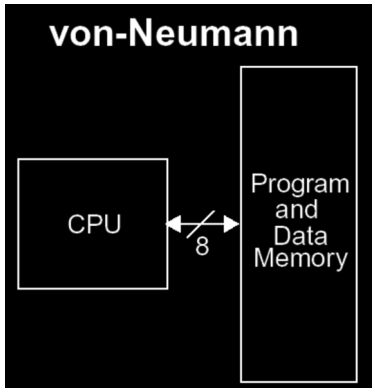


Figure 2.7 : Architecture Von Neumann.

-L'architecture **HARVARD** utilisée par les microcontrôleurs **PICS** est basée sur deux bus de données. Un bus est utilisé pour les **données** et un autre pour les **instructions**.

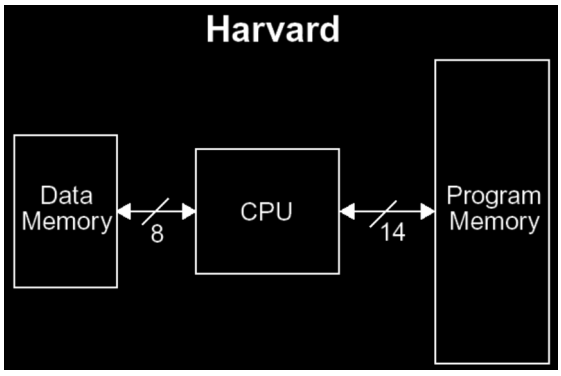


Figure 2.7 : Architecture Harvard

2.9 Avantages et inconvénients.

Chaque architecture présente des avantages et des inconvénients le tableau suivant résume la différence entre ces deux architectures :

	Architecture VON NEUMANN (MOTOROLA, INTEL, ZILOG, ..)	Architecture HARVARD (RISC) (MICROCHIP PICs)
Avantages	- Jeu d'instructions riches. - Accès à la mémoire facile.	- Jeu d'instructions pauvre, mais facile à mémoriser. - Le codage des instructions est facile, chaque instruction est codée sur un mot et dure un cycle machine. - Le code est plus compact.
Inconvénients	- Le temps pour exécuter une instruction est variable. - Le codage des instructions se fait sur plusieurs octets.	- Le jeu d'instruction est très pauvre, par exemple pour effectuer une comparaison il faut faire une soustraction. - Les accès aux registres internes et la mémoire sont très délicats.

Tableau 2.1 : Avantages et inconvénients des architectures des mémoire des microcontrôleurs.

2.10 Conclusion

Un microcontrôleur embarque à son intérieur un microprocesseur qui exécute un programme logé dans la mémoire intégré au circuit et commande son environnement à travers des ports. Il peut contenir des convertisseurs Analogique/Numérique ce qui permet de contrôler de mieux les systèmes analogiques ainsi que des modules de commande des moteurs (PWM,..).

Dans le chapitre suivant on étudiera le cas du microcontrôleur de famille PIC 16F877A. Le besoin de la conversion Analogique/Numérique et la gestion de la communication série rendent l'utilisation du fameux PIC 16F84 seul insuffisante. Ce qui nous motivera d'étudier aussi un PIC plus puissant le cas du PIC 16F877A de la maison MICROCHIP.

Chapitre 3 : Les microcontrôleurs de la famille PIC

3.1 Introduction

Notre choix pour les microcontrôleurs de la famille PIC peut se justifier par les raisons suivantes :
-Les performances sont identiques voir supérieurs à ses concurrents.
-Les prix sont les plus bas du marché.
-Très utilisé donc très disponible.
- Les outils de développement sont gratuits et téléchargeables sur le WEB.
-Le jeu d'instruction réduit est souple, puissant et facile à maîtriser.
-Les versions avec mémoire flash présentent une souplesse d'utilisation et des avantages pratiques indéniables.

3.2 Les PICs de Microchip [7]

Les PICs **(Peripheral Interface Controller)** sont des microcontrôleurs à architecture RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide ce qui augmente la vitesse de fonctionnement du microcontrôleur. La famille des PICs est subdivisée en 3 grandes familles :

-La famille Base-Line, qui utilise des mots d'instructions de 12 bits,
-La famille Mid-Range, qui utilise des mots de 14 bits (et dont font partie la 16F84 et 16F877),
-**La famille High-End**, qui utilise des mots de 16 bits.

Un PIC est identifié par un numéro de la forme suivante : **XX F YY-zz**

XX: famille du composant(16: Mid-Range).
F : La lettre F indique que la mémoire programme de ce PIC est de type "Flash".
C: EPROM ou EEPROM.
CR: PROM.
YY: Identification.
Zz : Vitesse maximale du quartz.

Une référence de type **16F84$_{zz}$** par exemple peut avoir un suffixe du type "**-zz**" dans lequel **zz** représente la fréquence d'horloge maximal que le PIC peut recevoir

Les PICs sont des composants STATIQUES, Ils peuvent fonctionner avec des fréquences d'horloge allant du continu jusqu'à une fréquence max spécifique à chaque circuit. Un PIC16F876- 04 peut fonctionner avec une horloge allant du continu jusqu'à 4 MHz.

Exemple : 16F877 = Mid-Range, mémoire FLASH (F).

PIC	FLASH	RAM	EEPROM	I/O	A/D	Port //	Port Série
16F870	2K	128	64	22	5	NON	USART
16F871	2K	128	64	33	8	PSP	USART
16F872	2K	128	64	22	5	NON	MSSP
16F873	4K	192	128	22	5	NON	USART/MSSP
16F874	4K	192	128	33	8	PSP	USART/MSSP
16F876	8K	368	256	22	5	NON	USART/MSSP
16F877	8K	368	256	33	8	PSP	USART/MSSP

Tableau 3.1 : Différents circuit de la famille 16F87X.

3.3 Architecture

Les PICs les plus répandus, dits "**8 bits**", sont bâtis autour d'un processeur 8 bits, c'est à-dire que les registres et le bus de données internes ont un format de 8 bits. Ils ont une **architecture Harvard** et possèdent donc un **bus "programme"** distinct du **bus de données** ce qui leur permet d'accéder et manipuler une donnée et simultanément aller lire l'instruction suivante. Ce parallélisme est possible grâce à un pipeline à 2 niveaux (voir les chapitres suivants).

Le **processeur** est de type **RISC** (Reduced Instruction Set Computer) et ne possède donc que peu d'instructions. Cette caractéristique permet de les coder dans un seul emplacement mémoire pour chaque instruction, ce qui permet un accès et un décodage plus rapide. Une instruction s'exécute ainsi en 1 cycle d'horloge soit 4 périodes de celle-ci. Les PICs permettent ainsi une cadence de plusieurs MIPS (Million Instructions Per Second), jusqu'à 5 MIPS pour la série PIC16 que nous utiliserons.

La contrepartie de ce gain de rapidité se situe au niveau de la simplicité des instructions qui nécessite d'en associer souvent plusieurs pour réaliser une opération même basique. Un autre inconvénient concerne l'accès aux registres plus complexe puisqu'on ne code qu'une partie de leur adresse dans le champ de l'instruction afin de limiter la taille des emplacements de la mémoire programme, l'autre partie définissant une page mémoire ("banque") sélectionnée au moyen d'un registre spécifique [7].

3.4 Différentes familles

Les PICs sont aujourd'hui découpés en 4 séries (PIC10, PIC12, PIC16 et PIC18) classées par "puissance" en termes de ressources internes, d'entrées/sorties et, ce qui est étroitement lié, en capacité de mémoire programme d'autant plus étendue qu'il y aura de ressources à contrôler. Certains PICs ont aujourd'hui disparu (PIC14, PIC16C5 en voie d'obsolescence…) et il faut être conscient que les PICs sont en constante évolution avec une pérennité donc réduite, mais les références supprimées du catalogue sont en général remplacées par de nouvelles généralement compatibles et plus performantes. Il est bien clair que les PICs sont avant tout destinés aux produits grands publics à faible durée de vie plutôt qu'aux applications militaires… [8]

3.5 Présentation du PIC 16F84

La structure générale du PIC 16F84 comporte 4 blocs :
-Mémoire de programme ; Mémoire de données ; Processeur ; Ressources auxiliaires (périphériques).

3.6 Caractéristiques du PIC 16F84

►La mémoire de programme contient les instructions pilotant l'application à laquelle le microcontrôleur est dédié. Il s'agit d'une mémoire non volatile (elle garde son contenu, même en l'absence de tension), elle est de type FLASH c'est à dire qu'elle peut être programmée et effacée par l'utilisateur via un programmateur et un PC. La technologie utilisée permet plus de 1000 cycles d'effacement et de programmation. Pour le PIC 16F84 cette mémoire est d'une taille de 1024*14 bits, c'est à dire qu'elle dispose de 1024 emplacements (de 000h à 3FFh) contenant chacun 14 cases car dans le cas du PIC, les instructions sont codées sur 14 bits. On peut donc stocker 1024 instructions.

►La mémoire de donnée est séparée en deux parties :
-une mémoire RAM de 68 octets puisque le bus de donnée est de huit bits. Cette RAM est volatile (les données sont perdues à chaque coupure de courant). On peut y lire et écrire des données.
-une mémoire EEPROM de 64 octets dans laquelle on peut lire et écrire des données (de huit bits soit un octet) et qui possède l'avantage d'être non volatile (les données sont conservées même en l'absence de tension). La lecture et l'écriture dans cette mémoire de données sont beaucoup plus lentes que dans la mémoire de données RAM.

►Le processeur est formé de deux parties :
-une unité arithmétique et logique (UAL) chargée de faire des calculs.
-un registre de travail noté W sur lequel travail l'UAL.

►les ressources auxiliaires qui sont dans le cas du PIC16F84
-ports d'entrées et de sorties.
-temporisateur.
-interruptions
-chien de garde
-mode sommeil

3.7 Structure interne du PIC 16F84

-Brochage et caractéristiques principales :

Le PIC16F84 est un circuit intégré de 18 broches (figure 3.1) :

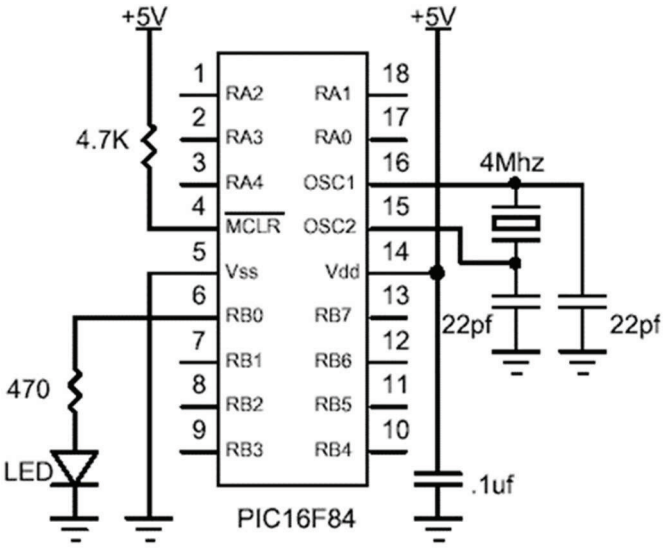


Figure 3.1 : Brochage du PIC 16F84.

►L'alimentation du circuit est assurée par les pattes VDD et VSS. Elles permettent à l'ensemble des composants électroniques du PIC de fonctionner. Pour cela on relie VSS (patte 5) à la masse (0 Volt) et VDD (patte 14) à la borne positive de l'alimentation qui doit délivrer une tension continue comprise entre 3 et 6 Volts.

►Le microcontrôleur est un système qui exécute des instructions les unes après les autres à une vitesse (fréquence) qui est fixée par une horloge interne au circuit. Cette horloge doit être stabilisée de manière externe au moyen d'un cristal de quartz connecté aux pattes OSC1/CLKIN (patte 16) et OSC2/CLKOUT (patte 15) .

►La patte 4 est appelée MCLR. Elle permet lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur. C'est à dire que si un niveau bas (0 Volt) est appliqué sur MCLR le microcontrôleur s'arrête, place tout ses registres dans un état connu et se redirige vers le début de la mémoire de programme pour recommencer le programme au début (adresse dans la mémoire de programme :0000). A la mise sous tension, la patte MCLR étant à zéro, le programme démarre donc à l'adresse 0000,(MCLR=Master Clear Reset).

►Les broches RB0 à RB7 et RA0 à RA4 sont les lignes d'entrées/sorties numériques. Elles sont au nombre de 13 et peuvent être configurées en entrée ou en sortie. Ce sont elles qui permettent au microcontrôleur de dialoguer avec le monde extérieur (périphériques). L'ensemble des lignes RB0 à RB7 forme le port B et les lignes RA0 à RA4 forment le port A. Certaines de ces broches ont aussi d'autres fonctions (interruption, timer).

Remarque : Cas particulier de la broche RA4 configurée en sortie possède une sortie de type drain ouvert. Cela veut dire qu'elle ne peut pas fournir de courant. Par contre, elle peut en consommer.

Exemple de programme (Structure d'un programme en mikroC voir partie TD) :

Source code 3.1 :

```
void main()
{
TRISB=0x00;  // registre de configuration (1 seul fois (0 : sortie, 1 : entrée))  TRISB=0b00000000; TRISB=0 ;
PORTB=0x00; // registre de travail.
delay_ms(1000);
PORTB=0b00111111;
PORTB=0;
}
```

3.8 Présentation générale du PIC 16F877A

Le PIC 16F877A est un circuit intégré de type CMOS de la maison MICROCHIP. Ce microcontrôleur est commercialisé sous un boîtier un DIL (Dual In Line) de 2x20 pattes. Le PIC 16F877A est caractérisé par :

Principales caractéristiques du PIC 16F877 :

Le PIC 16F877 est caractérisé par :
- Une fréquence de fonctionnement élevée, jusqu'à 20 MHz.
- Une mémoire vive de 368 octets.
- Une mémoire morte EEPROM de 256 octets pour la sauvegarde des données. - Une mémoire de type FLASH de 8 Kmots (1mot = 14 bits) .
- Chien de garde WDT.
- 33 lignes d'entrées /sorties. Chaque sortie peut sortir un courant maximum de 25 mA. - 3 Temporisateurs :
· TIMER0 (compteur 8 bits avec prédiviseur).
· TIMER 1 (compteur 16 bits avec prédiviseur et possibilité d'utiliser une horloge externe réseau RC ou QUARTZ).
· TIMER2 (compteur 8 bits avec prédiviseur et postdiviseur).
- 2 entrées de captures et de comparaison avec PWM (Modulation de largeur d'impulsions).
- Un convertisseur Analogique Numérique 10 bits avec 8 entrées multiplexées.
- Une interface de communication série asynchrone et synchrone (USART/SCI). - Une interface de communication série synchrone (SSP/SPI et I2C).
- Une tension d'alimentation entre 2 et 5.5 V.

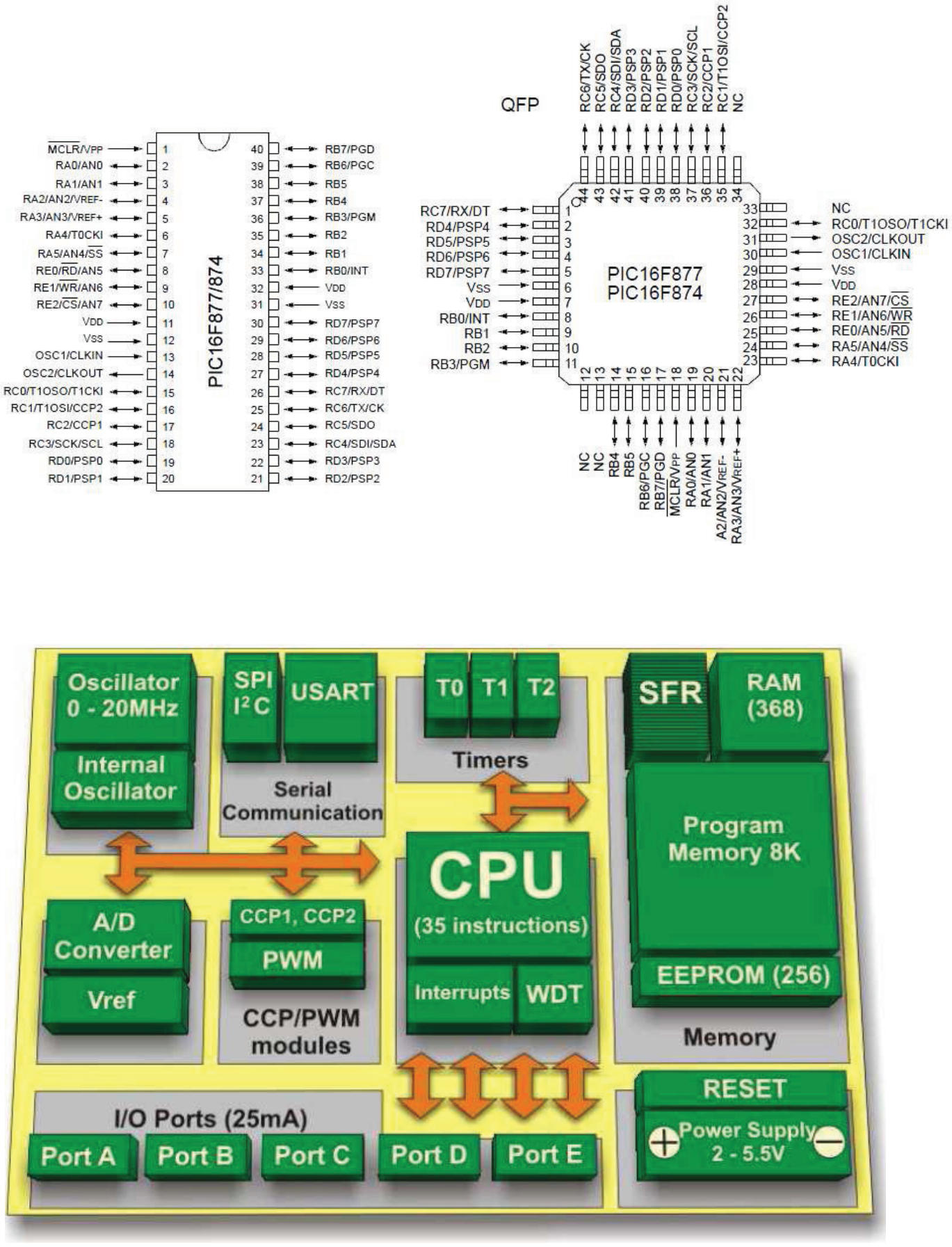


Figure 3.2 : Brochage et Architecture interne du PIC16F877A.

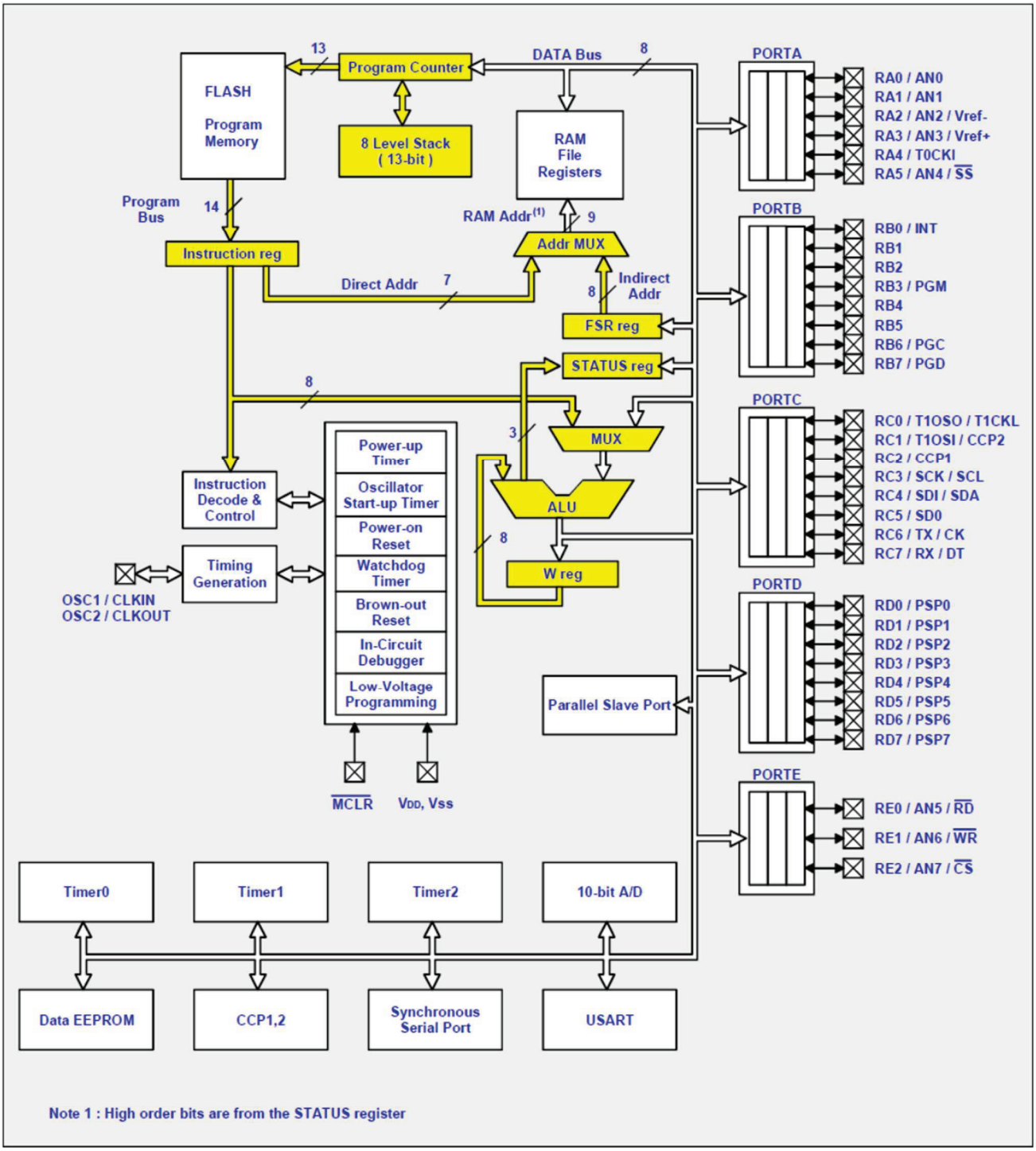


Figure 3.3 : Synoptique complet du PIC 16F877A.

Un microcontrôleur, c'est avant tout un microprocesseur, une unité de traitement logique qui effectue l'une après l'autre les opérations contenues dans un microprogramme stocké en mémoire (**la mémoire FLASH**). On peut le voir sur le schéma (en jaune ci-dessous), il est essentiellement composé de :
-l' « **ALU** » (Unité Arithmétique et Logique) qui effectue les opérations sur les données,

-le registre de travail « **W reg**. », le multiplexeur « **MUX** », le registre de statut « **status reg** », le registre « **FSR reg** » utilisé pour l'adressage indirect (en assembleur…), le multiplexeur d'adresse « **Addr mux** », le compteur programme « **Program Counter** » qui pointe les instructions à exécuter, la pile à 8 niveaux « **8 level Stack** », le registre d'instruction « **Instruction reg** », ainsi que les différents bus qui relient tous ces éléments entre eux [8].

On n'entrera pas ici dans les détails du fonctionnement d'un microprocesseur ; c'est très intéressant à connaître, indispensable même si on programme en assembleur.
Mais en ce qui nous concerne, on veut programmer en C, et c'est donc le compilateur qui se chargera de traduire notre code source en instructions de bas niveau pour le microprocesseur contenu dans le PIC. C'est là le principal avantage de la programmation en C.
On se concentre d'avantage sur « ce que fait le programme » que sur « comment fonctionne le programme ».

On va tout de même jeter un petit coup d'œil sur le schéma ci-dessus, histoire de comprendre quelques particularités du PIC, déroutantes au premier abord.

On s'aperçoit que les bus autour de l'ALU ont un format de 8 bits : le PIC 16F877 travaille sur des données de 8 bits, c'est donc bien un microcontrôleur 8 bits.
-Pourquoi donc ce cas le « Program Bus » est-il, lui, large de 14 bits ?

C'est simple, certaines instructions peuvent être codées sur plus de 8 bits. Si ce bus n'était large que de 8 bits, il faudrait plus d'un cycle d'horloge pour transmettre ces instructions, alors qu'avec un bus plus large, ça passe en une fois. De plus, la mémoire programme, indépendante du bus de données, est elle-même adressée avec un bus large : le « **Program Counter** », qui pointe sur l'instruction en cours, à une largeur de 13 bits. Et avec 13 bits on peut coder environ 8000 adresses. Autrement dit, on à 8000 cases de mémoire programme pouvant contenir chacune UNE instruction complète. Cette architecture avec bus de données et de programme séparés (architecture « Harvard ») permet donc d'optimiser le fonctionnement du PIC.

3.9 Conclusion

La plupart du temps, le PIC exécute une instruction et charge la suivante simultanément en un seul cycle d'horloge. En comparaison, un microcontrôleur 8 bits construit selon l'architecture concurrente « Von Neumann » (mémoire programme et données reliés au microprocesseur par un unique bus 8 bits) devra faire plusieurs cycles pour chercher les instructions en mémoire (en plusieurs fois si c'est une instruction longue) et les exécuter ensuite. En conséquence, à fréquence d'horloge égales, un microprocesseur « Harvard » sera plus rapide qu'un « Von Neumann ». Dans le chapitre suivant le principe de fonctionnement du PIC sera détaillé.

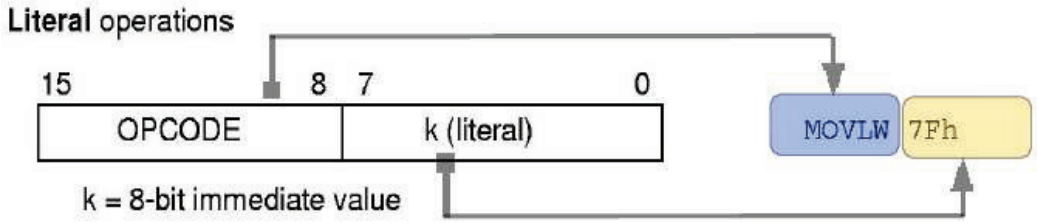
Chapitre 4 : Le principe de fonctionnement du PIC 16F877

4.1 Introduction

Pour une meilleure compréhension de la suite du cours il est nécessaire de donner quelques définitions : Une instruction est composée au minimum de deux parties [5] :

Instruction = OPCODE + opérande(s)

OPCODE (Operation CODE) : partie d'une instruction qui précise quelle opération doit être réalisée.



Extrait du datasheet (documentation technique) du PIC18F4520.

Il existe trois étapes pour l'exécution d'une instruction :

✓ Lecture de l'instruction (1)
✓ Décodage de l'instruction (2)
✓ Exécution de l'instruction (3)

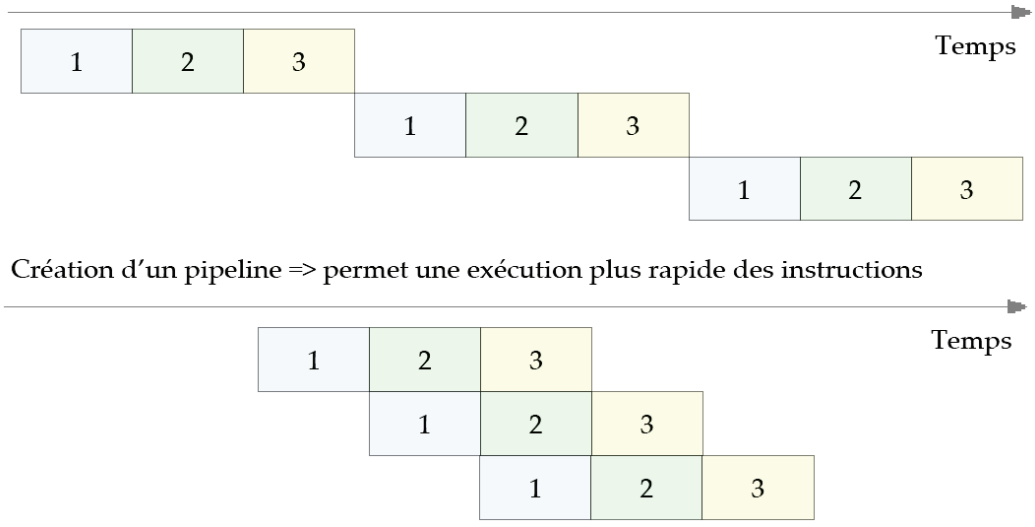


Figure 4.1 : Les Etapes d'exécution d'une instruction.

4.2 Principe de fonctionnement du PIC

Un microcontrôleur exécute des instructions. On définit « le cycle instruction » comme le temps nécessaire à l'exécution d'une instruction. Attention de ne pas confondre cette notion avec le cycle d'horloge qui correspond au temps nécessaire à l'exécution d'une opération élémentaire (soit un coup d'horloge). Une instruction est exécutée en deux phases [4]:

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

Chaque phase dure 4 cycles d'horloge comme le montre la figure 4.2 :

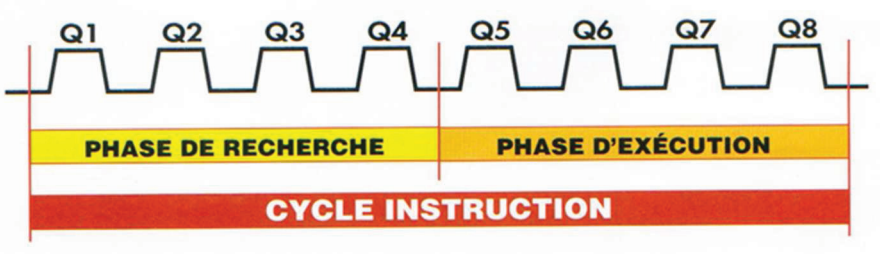


Figure 4.2 : Un cycle d'instruction.

On pourrait donc croire qu'un cycle instruction dure 8 cycles d'horloge mais l'architecture particulière du PIC lui permet de réduire ce temps par deux. En effet, comme les instructions issues de la mémoire de programme circulent sur un bus différent de celui sur lequel circulent les données, ainsi le processeur peut effectuer la phase de recherche d'une instruction pendant qu'il exécute l'instruction précédente (Voir figure 4.3 et 4.4).

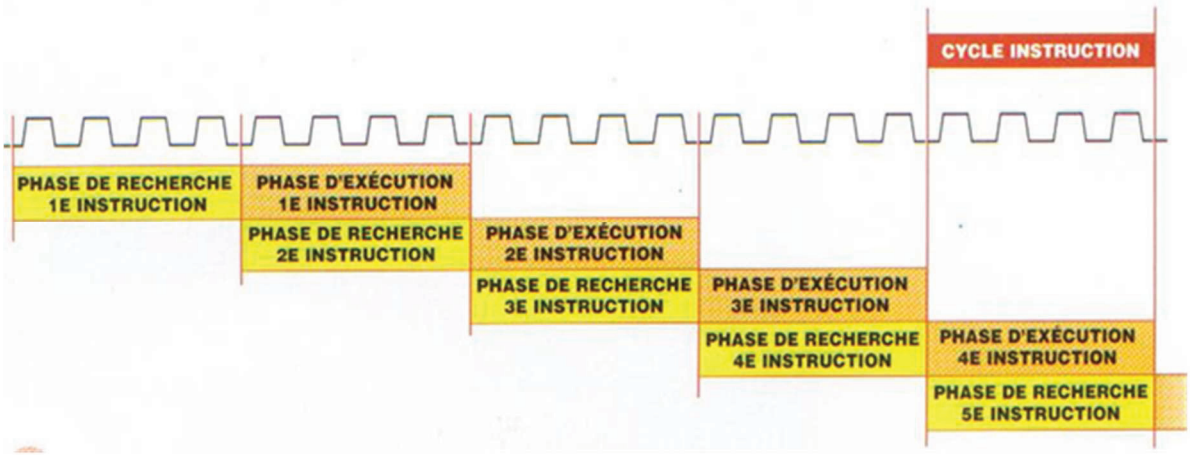


Figure 4.3 : Un cycle d'instruction dans un PIC 16F877.

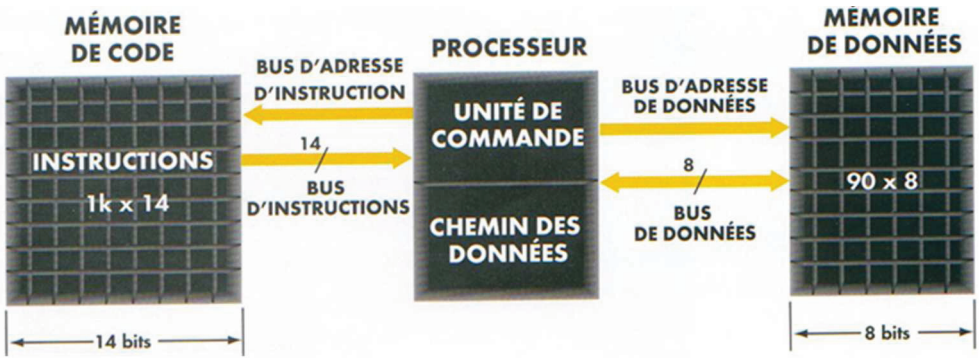


Figure 4.4 : L'Architecture Harvard dans un PIC.

4.3 Déroulement d'un programme

Le déroulement d'un programme s'effectue de façon très simple. A la mise sous tension, le processeur va chercher la première instruction qui se trouve à l'adresse 0000 de la mémoire de programme, l'exécute puis va chercher la deuxième instruction à l'adresse 0001 et ainsi de suite (sauf cas de saut ou d'appel de sous-programme que nous allons voir plus loin). On parle de fonctionnement séquentiel. La figure 4.5 va nous permettre de mieux comprendre le fonctionnement [4]:

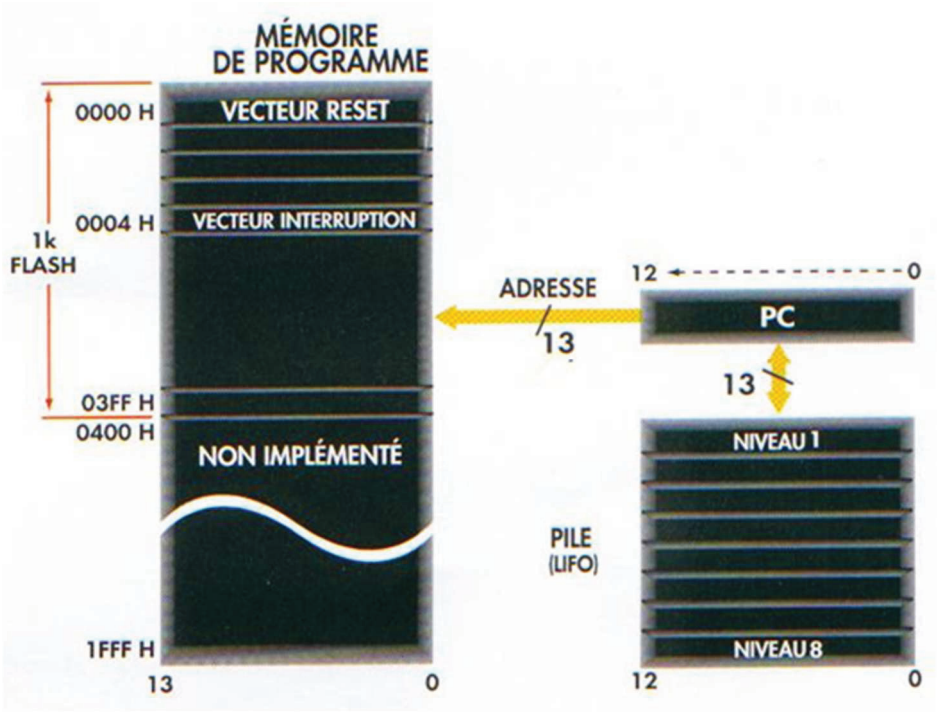


Figure 4.5 : Schéma du fonctionnement séquentiel et recherche d'instruction dans le PIC 16F877.

-Le déroulement d'un programme est basé essentiellement sur :

4.3.1 Mémoire de programme (ROM)

La Mémoire Programme, de type FLASH sur le 16F877. Capacité : 8K. C'est dans celle-ci qu'est stocké le programme du PIC. Après compilation de votre code, le compilateur génère un fichier « .hex », une suite de codes hexadécimaux. Celui-ci est transféré ensuite dans la mémoire programme du PIC à l'aide du programmateur. Cette mémoire n'est pas reliée au bus de données (DATA Bus), sa vocation est de stocker le programme du PIC, mais PAS les variables de votre programme. Le gros avantage de la mémoire FLASH c'est que vous pouvez la réécrire, donc implanter un nouveau programme dans le PIC. Les PIC existent également avec d'autres versions de mémoire programme (non-FLASH), certaines ne pouvant être programmée qu'une seule fois. En résumer :

Elle contient le programme à exécuter.
Elle contient 1k "mots" de 14 bits.
Le PIC 16F84 possède un compteur ordinal qui permet d'adresser 8K * 14 bits.
L'adresse 0000h contient le vecteur du reset.
L'adresse 0004h l'unique vecteur d'interruption du PIC.
La pile contient 8 valeurs : Ce sont des zones réservées
Par le système (pas d'adresse).

4.3.2 Compteur Ordinal (PC : Program Counter)

Le Compteur ordinal pointe l'instruction en cours, il a une largeur de 13 bits.
Avec 13 bits on peut coder environ 8000 adresses. Autrement dit, on à 8000 cases de mémoire programme pouvant contenir chacune UNE instruction complète. Il est lié à la pile système. Une pile 8 niveaux d'appels de sous-programme ou fonctions.

Explication du déroulement du programme [9] :

On constate sur cette la figure 4.5 que la mémoire de programme (ROM) contient 1024 emplacements (3FF en hexadécimale) contenant 14 bits (de 0 à 13). Une instruction occupe un emplacement qui est défini par une adresse. Le processeur peut alors sélectionner l'emplacement souhaité grâce au bus d'adresse et il peut lire son contenu (ici l'instruction) grâce à son bus d'instruction (voir figure 4.4). Cet adressage s'effectue à l'aide du compteur ordinal appelé PC qui lors de la mise sous tension démarre à zéro puis s'incrémente de 1 tous les quatre coups d'horloge, on exécute bien ainsi les instructions les unes à la suite des autres.

Mais il arrive que dans un programme on fasse appel à un sous-programme dont l'adresse de l'instruction ne se trouve pas juste après celle qui est en train d'être exécutée. C'est le rôle de la pile qui sert à emmagasiner de manière temporaire l'adresse d'une instruction. Elle est automatiquement utilisée chaque fois que l'on appelle un sous-programme et elle permet une fois que l'exécution du sous-programme est terminée de retourner dans le programme principal juste après l'endroit où l'on a appelé le sous-programme. On constate que cette pile possède huit niveaux, cela signifie qu'il n'est pas possible d'imbriquer plus de huit sous programmes, car au-delà de huit, le processeur ne sera plus capable de retourner à l'adresse de base du programme principal.

La programmation en C apporte là un peu de souplesse : un bon compilateur veillera pour vous à ce que la limite de 8 niveaux ne soit pas dépassée, quitte à recopier localement une fonction pour éviter un saut à l'adresse mémoire où cette fonction est déjà présente. Au détriment donc de l'occupation en mémoire. Ainsi, mieux vaut-il éviter de créer trop de niveaux d'imbrication d'appels de fonctions dans nos programmes.

L'adresse 0000 est réservée au vecteur RESET, cela signifie que c'est à cette position que l'on accède chaque fois qu'il se produit une réinitialisation (0 volts sur la patte MCLR). C'est pour cette raison que le programme de fonctionnement du microcontrôleur doit toujours démarrer à cette adresse.

L'adresse 0004 est assignée au vecteur d'interruption et fonctionne de manière similaire à celle du vecteur de Reset. Quand une interruption est produite et validée, le compteur ordinal PC se charge avec 0004 et l'instruction stockée à cet emplacement est exécutée.

4.4 La mémoire de données RAM

Cette mémoire fait partie de la zone d'adressage des données. Elle comprend tous les registres spéciaux permettant de contrôler le cœur du PIC ainsi que ses périphériques. Elle contient également des cases mémoires à usage générique dans lesquelles pourront être stockées les variables de nos futurs programmes.

Si l'on regarde cette mémoire, on s'aperçoit que celle-ci est un peu particulière comme le montre la figure 4.6. On constate en effet que cette mémoire est séparée en deux pages (page 0 et page 1). De plus, on remarque que tant pour la page 0 que pour la page 1, les premiers octets sont réservés (SFR pour Special File Register). Ces emplacements sont en effet utilisés par le microcontrôleur pour configurer l'ensemble de son fonctionnement. On les appelle registres spécifiques et nous verrons aux chapitres suivant leurs rôles. Le bus d'adresse qui permet d'adresser la RAM est composé de 7 fils ce qui veut dire qu'il est capable d'adresser 128 emplacements différents. Or, chaque page de la RAM est composée de 128 octets, le bus d'adresse ne peut donc pas accéder aux deux pages, c'est pourquoi on utilise une astuce de programmation qui permet de diriger le bus d'adresse soit sur la page 0, soit sur la page 1. Cela est réalisé grâce à un bit d'un registre spécifique (le bit RP0 du registre STATUS) dont nous verrons le fonctionnement plus loin.

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

Adr.	Banque0	Banque1	Adr.
00h	Indirect addr.	Indirect addr.	80h
01h	TMR0	OPTION_REG	
02h	PCL	PCL	82h
03h	STATUS	STATUS	
04h	FSR	FSR	84h
05h	PORTA	TRISA	
06h	PORTB	TRISB	86h
07h	--	--	
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	
0Ch - 4Fh	68 cases mémoires	idem banque 0	8Ch - CFH
50h - 7Fh	inutilisé	inutilisé	D0H - FFH

Figure 4.6 : Mémoire de programme RAM.

4.5 La mémoire EEPROM

La Mémoire EEPROM est plutôt une mémoire de stockage de données à long terme, alors que la RAM est utilisée pour les variables du programme. Sur le PIC 16F877, on a 256 octets d'EPROM que l'on peut lire et écrire depuis un programme. Ces octets sont conservés même après une coupure de l'alimentation et sont très utiles pour conserver des paramètres semi permanents : code d'accès, version du programme, message d'accueil, valeur invariable, etc.

Les mémoires de type EEPROM sont limitées en nombre de cycles d'effacement / écriture. Ce nombre de cycle est tout de même de l'ordre du million pour le PIC, mais si on l'utilisait pour stocker des variables modifiées plusieurs milliers de fois par secondes, cette limite pourrais être atteinte plus vite qu'on ne le croit. Cependant, pour stocker toute les heures une mesure de température, c'est tout bon (Programme de fonctionnement du PIC).

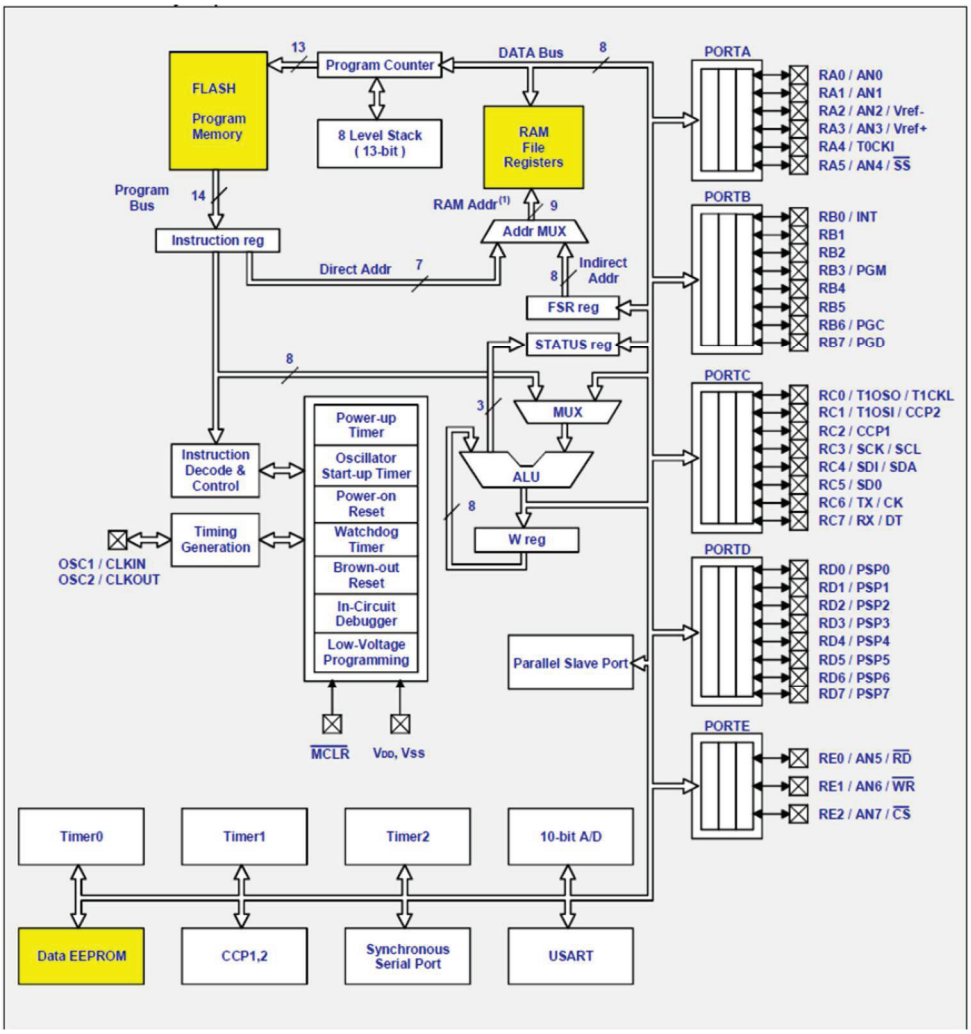


Figure 4.7 : Schéma synoptique de l'emplacement des mémoires du PIC 16f877.

4.6 Les registres

Il faut bien différencier ici deux concepts : **les registres internes** du processeur et **les registres externes** en mémoire. On parlera dans les deux cas de registres, mais on se réfèrera la plupart du temps a des registres externes.

Nous avons vu dans la partie précédente que la mémoire de données RAM contenait des registres spécifiques qui permettent de configurer le PIC, nous allons détailler les registres les plus utilisés et voir comment on peut accéder à la page 0 ou la page 1. Afin de faciliter la compréhension, les registres utilisés serons donner on exemples. Les suivants Tableaux regroupes l'ensemble des registres.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
01h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	43, 150
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	47, 150
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	48, 150
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	49, 150
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
0Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	26, 150
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	28, 150
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	57, 150
11h	TMR2	Timer2 Module Register								0000 0000	62, 150
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	61, 150
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	79, 150
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	82, 82, 150
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	63, 150
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	63, 150
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	64, 150
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	112, 150
19h	TXREG	USART Transmit Data Register								0000 0000	118, 150
1Ah	RCREG	USART Receive Data Register								0000 0000	118, 150
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	63, 150
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	63, 150
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	64, 150
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	133, 150
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	127, 150

Tableau 4.1 : Tableau de la page 0 du registres spécifiques SFR.

►adresse 00, INDF. Cette adresse ne contient pas de registre physique, elle sert pour l'adressage indirect.
►adresse 01, TMR0 . Contenu du Timer (8 bits). Il peut être incrémenté par l'horloge (fosc/4) c'est à dire tous les 4 coups d'horloge ou par la broche RA4.
►adresse 02, PCL .8 bits de poids faibles du compteur ordinal PC. Les 5 (13-8) bits de poids forts sont dans PCLATH.
►adresse 04, FSR . Registre de sélection de registre : contient l'adresse d'un autre registre
►adresse 05 , PORTA . Ce registre contient l'état des lignes du port A
►adresse 06 , PORTB . Ce registre contient l'état des lignes du port B

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Tableau 4.2 : Tableau de la page 1 du registres spécifiques SFR.

►adresse 85, TRISA. Direction des données pour le port A : 0 pour sortir et 1 pour entrer
►adresse 86, TRISB. Direction des données pour le port B : 0 pour sortir et 1 pour entrer

4.6.1 Le registre d'état STATUS [10]

L'unité arithmétique et logique est composée de :

-D'un accumulateur **8** bits **W : WORKING** (travail)**,** c'est lui qui effectue toutes les opérations arithmétiques et logiques.
- Un registre d'état **8** bits **STATUS**.

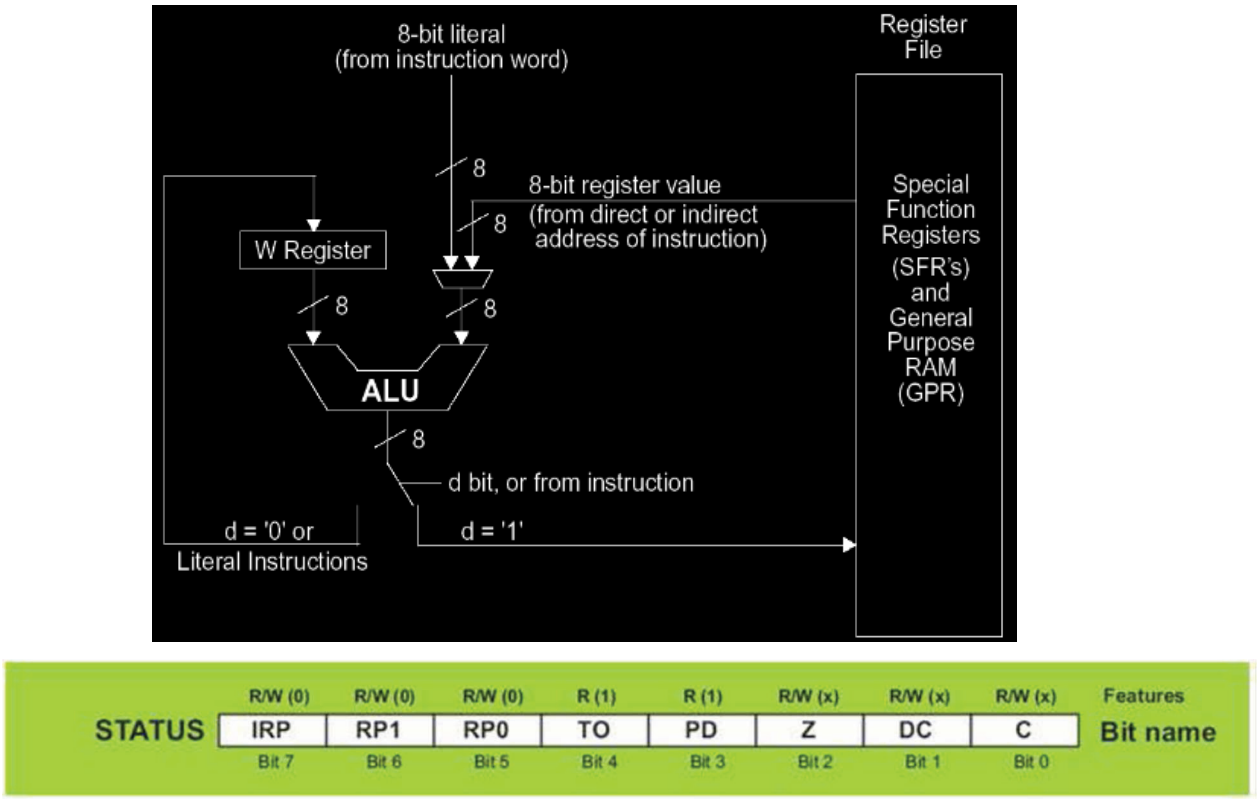


Figure 4.8 : L'Unité arithmétique et logique, le registre de travail et le registre STATUT.

Le Registre d'état STATUS à l'adresse 03. Les cinq bits de poids faible de ce registre sont en lecture seule, ce sont des témoins (drapeaux ou flag en anglais) caractérisant le résultat de l'opération réalisée par l'UAL. Le bit RP0 est lui en lecture /écriture et c'est lui qui permet de sélectionner la page dans la mémoire RAM . Si RP0=0 on accède à la page 0 et si RP0=1 on accède à la page 1.

Au reset, seul le bit RP0 de sélection de page est fixé (RP0=0 : page 0)
TO/ (Time Out) : débordement du timer WDT
PD/ : (Power Down) caractérise l'activité du chien de garde WDT
Z (zéro) résultat nul pour une opération arithmétique et logique.
DC (digit carry) retenue sur un quartet (4 bits)
C (carry) retenue sur un octet (8 bits).

4.6.2 Le registre de travail W

Contrairement au registre d'état STATUS ; Le registre W, il n'a pas d'adresse, est un registre de travail de 8 bits. L'ALU de 8 bits qui réalise les opérations entre W et n'importe quel autre registre FSR (page 0 ou 1) ou constante k (68 cases mémoire). Le résultat de l'opération peut être placé soit dans W soit dans les registres adressé. L'ALU est associé au registre d'état STATUS par les bits Z, C et DC :

☑ **C** : Ce bit flag passe à 1 lorsqu'il y a une retenue sur un octet.
☑ **DC** : Ce bit flag passe à 1 lorsqu'il y a une retenue sur un quartet.
☑ **Z** : Ce bit flag passe à 1 quand le résultat d'une opération est nul.

4.6.3 Les registres pour l'écriture et la lecture sur la mémoire EEPROM

On y accède à la mémoire EEPROM l'aide des registres EEADR et EEDATA avec les deux registres de contrôle EECON1 et EECON2.

►adresse 08 , EEDATA .Contient un octet lu ou à écrire dans l'EEPROM de données.
►adresse 09 , EEADR . Contient l'adresse de la donnée lue ou écrite dans l'EEPROM de données.
►adresse 88 , EECON1 Contrôle le comportement de l'EEPROM de données.

EEIF (EEProm Interrupt Flag) passe à 1 quand l'écriture est terminée.
WRERR (Write Error) 1 si erreur d'écriture.
WREN (Write Enable) : 0 pour interdire l'écriture en EEPROM de données.
WR (Write) 1 pour écrire une donnée. Bit remis automatiquement à 0
RD (Read) : 1 pour lire une donnée. Bit remis automatiquement à 0

►adresse 89 , EECON2 . Registre de sécurité d'écriture en EEPROM de données.
Une donnée ne peut être écrite qu'après avoir écrit successivement 0x55 et 0xAA dans ce registre.

4.6.3 Le registre INTCON

Le registre INTCON Contrôle 4 interruptions ; L'interruption fait appel à un sous-programme pour arrêter le programme principal est exécuté ce dernier.

Masques :
GIE : (Global Interrupt Enable) : masque global d'inter.
EEIE : (EEProm Interrupt Enable) autorise l'interruption venant de l'EEPROM.
T0IE : (Timer 0 Interrupt Enable) autorise l'interruption provoquée par le débordement du TIMER0
INTE: (Interrupt Enable) autorise l'interruption provoquée par un changement d'état sur broche RB0/INT
RBIE: (RB Interrupt Enable) autorise les interruptions provoquées par un changement d'états sur l'une des broches RB4 à RB7. Si ces bits sont mis à 1 , ils autorisent les interruptions pour lesquels ils sont dédiés .

Drapeaux :

T0IF : (Timer 0 Interrupt Flag) débordement du timer

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

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

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Une interruption provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption. A la fin de cette procédure, le microcontrôleur reprend le programme principal à l'endroit où il l'a laissé. A chaque interruption sont associés deux bits, un bit de validation et un drapeau. Le premier permet d'autoriser ou non l'interruption, le second permet au programmeur de savoir de quelle interruption il s'agit. Sur le 16F876/877, l'es interruptions sont classées en deux catégories, les interruptions primaires et les interruptions périphériques. Elles sont gérées par les registres [5] :

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIE1 (bk1)	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIR1 (bk0)	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE2 (bk0)	-	-	-	EEIE	BCLIE	-	-	CCP2IE
PIR2 (bk1)	-	-	-	EEIF	BCLIF	-	-	CCP2IF
OPTION_REG(bk1)		INTEDG						

Tableau 4.3 : Registres d'interruptions Sur le 16F876/877.

►Toutes les interruptions peuvent être validées/interdites par le bit INTCON.GIE
►Toutes les interruptions périphériques peuvent être validées/interdites par le bit INTCON.PEIE
►Chaque interruption peut être validée/interdite par son bit de validation individuel

En résumé, pour valider une interruption périphérique (par exemple), il faut positionner 3 bits, GIE, PEIE et le bit individuel de l'interruption.

-Les sources d'interruptions

Interruption : Source d'interruption	Validation	Flag	PEIE
T0I : Débordement Timer 0	INTCON,T0IE	INTCON,T0IF	non
INT : Front sur RB0/INT	INTCON,INTE	INTCON,INTF	non
RBI : Front sur RB4-RB7	INTCON,RBIE	INTCON,RBIF	non
ADI : Fin de conversion A/N	PIE1,ADIE	PIR1,ADIF	oui
RCI : Un Octet est reçu sur l'USART	PIE1,RCIE	PIR1,RCIF	oui
TXI : Fin transmission d'un octet sur l'USART	PIE1,TXIE	PIR1,TXIF	oui
SSPI : Caractère émis/reçu sur port série synchrone	PIE1,SSPIE	PIR1,SSPIF	oui
TMR1I : Débordement de Timer 1	PIE1,TMR1IE	PIR1,TMR1IF	oui
TMR2I :Timer 2 a atteint la valeur programmée	PIE1,TMR2IE	PIR1,TMR2IF	oui
PSPI : Lecture/écriture terminée sur Port parallèle (16F877)	PIE1,PSPIE	PIR1,PSPIF	oui
CCP1I : Capture/comparaison de TMR1 avec module CCP1	PIE1,CCP1IE	PIR1,CCP1IF	oui
CCP2I : Capture/comparaison de TMR1 avec module CCP2	PIE2,CCP2IE	PIR2,CCP2IF	oui
EEI : Fin d'écriture en EEPROM	PIE2,EEIE	PIR2,EEIF	oui
BCLI : Collision sur bus SSP en mode I2C	PIE2,BCLIE	PIR2.BCLIF	oui

Tableau 4.4 : Sources d'interruptions Sur le 16F876/877.

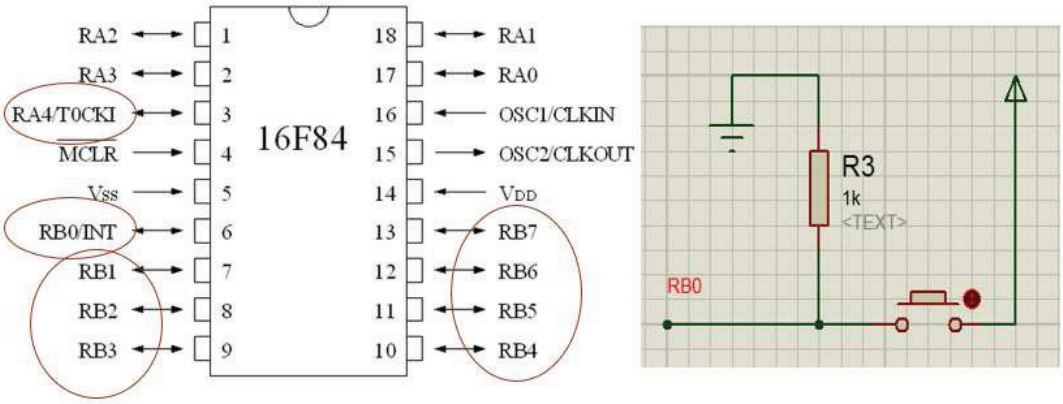


Figure 4.9 : Les 3 types d'interruptions physique utilisés par le PIC 16F84 (Timer0, RB0/INT et l'ensemble des RB qui restent).

Mise en œuvre :

Pour Configurer le Pic pour qu'il utilise l'interruption RB0 il faut fermer GIE et INTE (mètre à 1). D'où l'utilisation du registre INTCOM ; INTF représente le Flag par défaut elle est égale à 0 une on choisit RB0 elle devient 1

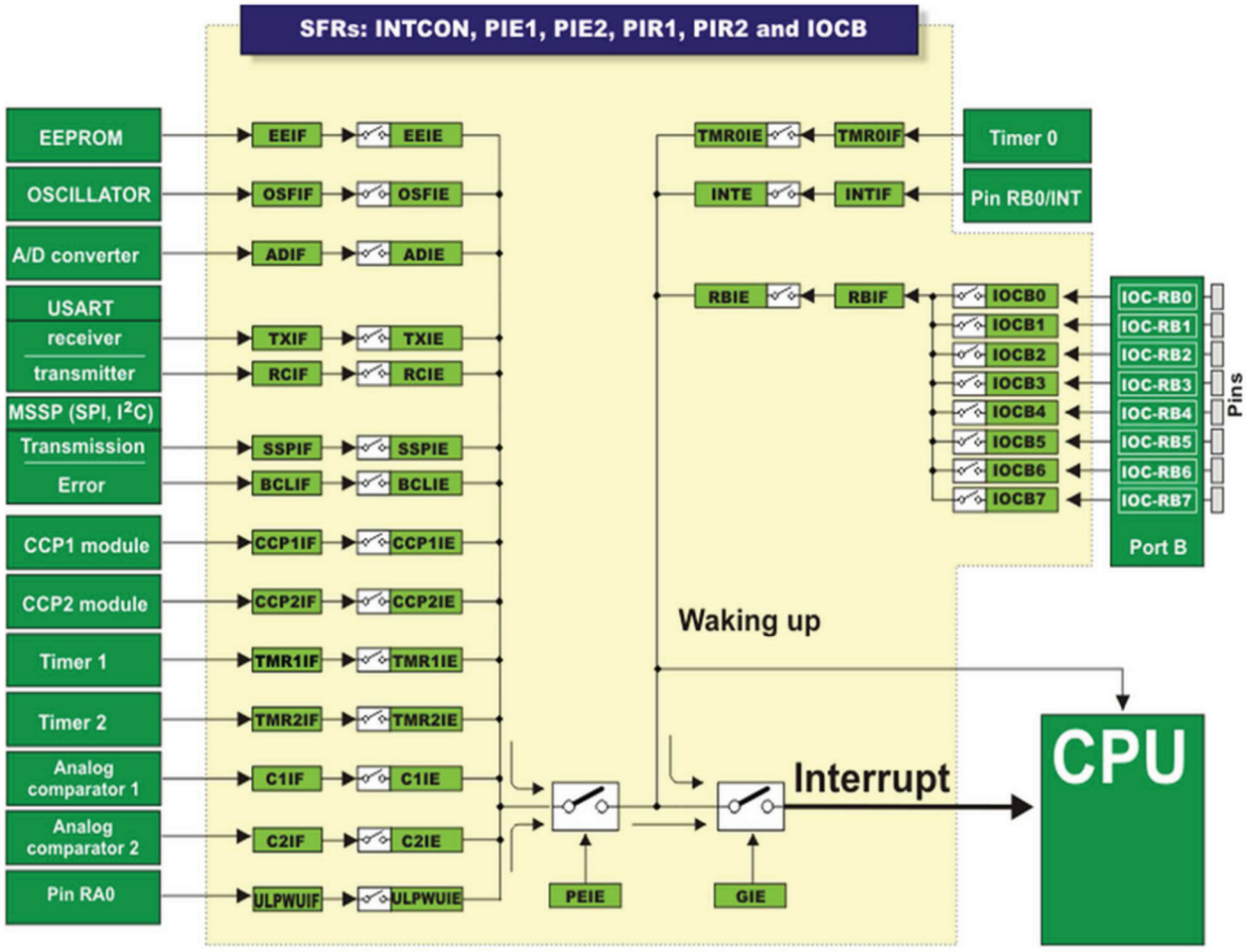


Figure 4.9 : Masques et drapeaux pour la configuration du PIC pour l'interruption.

Exemple de programme :

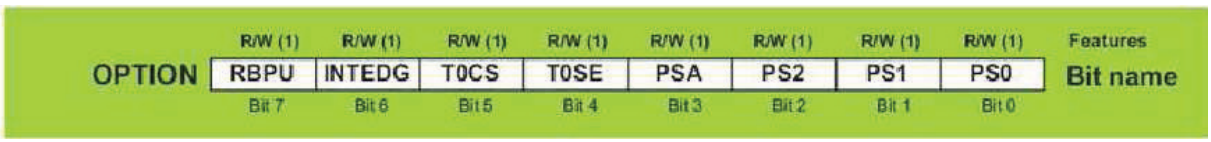
Le programme suivant incrémente la valeur S1S2 de 00 jusqu'à 59 (00 01 02 03 04 05 …59 00 01 …) et ainsi de suite. Le fête d'enfoncer l'interrupteur physique place au niveau de RB0, le programme principal s'arrête laissons la place à l'exécution du sous-programme de l'interruption.
00 ►(Enfoncer l'interrupteur)► 02 ; La valeur 01 sera dépasser automatiquement.

Source code 4.1 :

```
char s1,s2 ;
void interrupt ()
{
if ( INTCON.b1==1)  // pour ouvrir le FLAG.
    {
     s1++;
    }
INTCON.b1=0; // pour fermer le flag.
}
void main()
{
INTCON=0b10010000 ;
s1=s2=48; //code ascci de 0 initialisation
for(;;)
{
delay_ms (1000);
s1++;
if(s1==58)  {s1=48;  s2++;}
if(s2==54)  {s1=48; s2=48;}
}
}
```

4.6.4 Le registre OPTION

Le registre OPTION à 8 bits (tous à 1 au RESET) affectant le comportement des E/S et des timers.



RBPU/ (RB Pull Up) Résistances de tirage à Vdd des entrées du port B. Si RBPU/=0 les résistances de pull-up sont connectées en interne sur l'ensemble du port B.
INTEDG (Interrupt Edge) sélection du front actif de l'interruption sur RB0/INT (1 pour front montant et 0 pour front descendant).
RTS (Real Timer Source) sélection du signal alimentant le timer 0 : 0 pour horloge interne, 1 pour RA4/T0CLK
RTE (Real Timer Edge) sélection du front actif du signal timer (0 pour front montant).
PSA (Prescaler assignment) 0 pour Timer 0 et 1 pour chien de garde WDT.
PS2..0 (Prescaler 210) sélection de la valeur du diviseur de fréquence pour les timers.

4.6.5 Registre de configuration

On les trouve très souvent sous le nom de « fuses » (fusibles) dans les documentations, car, historiquement, il s'agissait de petits fusibles rassemblés au sein d'une grille. On appliquait une tension de 12V à ceux que l'on voulait faire « claquer ». Cela rendait leur écriture définitive. Ils ont depuis été remplacés par de simples cases en mémoire.

Pendant la phase de la programmation du µC, on programme aussi un registre de configuration logé dans la mémoire EEPROM. Ce registre est un mot de 14 bits qui permet de :

- ☑ Choisir le type de l'oscillateur pour l'horloge.
- ☑ Valider ou non le timer du watchdog WDT.
- ☑ Autoriser ou non une temporisation à la mise sous tension.
- ☑ Interdire ou non la lecture des mémoires de programme et de données.

Registre de configuration ⇒

CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	$FOSC_1$	$FOSC_0$

☑ Bits **$FOSC_0$** et **$FOSC_1$** : Sélection du type d'oscillateur pour l'horloge.
$FOSC_1FOSC_0$=11 ⇒ Oscillateur à circuit RC jusqu'à 4 MHz.
$FOSC_1FOSC_0$=10 ⇒ Oscillateur HS, quartz haute fréquence, jusqu'à 20 MHz.
$FOSC_1FOSC_0$=01 ⇒ Oscillateur XT, quartz standard jusqu'à 4 MHz.
$FOSC_1FOSC_0$=00 ⇒ Oscillateur LP, quartz basse fréquence, jusqu'à 200 KHz.

☑ Bit **WDTE** : Validation du timer du watchdog WDT.
WDTE=1 ⇒ WDT validé et WDTE=0 ⇒ WDT inhibé.

☑ Bit **PWRTE** : Validation d'une temporisation à la mise sous tension.
Le µC possède un timer permettant de retarder de 72 ms le lancement du programme après la mise sous tension. Ce délai maintient le µC à l'arrêt et permet ainsi à la tension d'alimentation de bien se stabiliser.
PWRTE=1 ⇒ le µC démarre tout de suite et PWRTE=0 ⇒ le µC attend 72 ms.

☑ Bits **CP** : Protection en lecture des mémoires de programme et de données.

Exemple : On désire configurer le registre pour répondre aux critères suivants :
oscillateur à quartz de 4 MHz, le timer du watchdog n'est pas autorisé, une attente de 72 ms est souhaitée et le µC n'est pas protégé en lecture.

Configuration requise ⇒

1	1	1	1	1	1	1	1	1	1	0	0	0	1

⇔ **3FF1** en hexadécimal

4.7 Conclusion

On à vue dans ce chapitre que les registres sont des espaces mémoires adressables indépendamment par le microcontrôleur. Selon les registres, ils sont soient à :

►Usage général : permettant ainsi de stocker des données (résultats de calculs)
►Usage plus spécifique : permettant la gestion de certains modules du microcontrôleur (ADC, PWM, Timers...) ou l'interaction avec l'environnement extérieur (port d'entrées-sorties)

Ces registres sont à lecture-écriture. Il est donc possible d'aller lire ou modifier leur valeur. La taille de ces registres est imposée par le format de données traitées par le microcontroleur (8, 16, 32 ou 64 bits). Dans le cas des microcontrôleurs PIC16F de la société Microchip, ces registres ont une taille de 8 bits : c'est à dire que le plus petit espace adressable est un mot de 8 bits (soit 1 octet).

Chapitre 5 : Les différents périphérique du PIC 16F877

5.1 Introduction

Un microcontrôleur possède généralement un cristal interne (quartz), qui jouera le rôle d'horloge par effet piézoélectrique. Il est souvent possible d'exploiter à la place un cristal externe, notamment pour utiliser des fréquences plus élevées que le cristal interne.

L'horloge va cadencer le processeur ; un tic correspondra à une instruction élémentaire. Une fréquence basse ralentira forcement l'exécution d'un processus, et empêchera d'utiliser des protocoles nécessitant une fréquence plus élevée. Elle permettra néanmoins de consommer beaucoup moins d'énergie. Une fréquence élevée permettra d'être plus efficace mais consommera plus d'énergie, ce qui peut également dégager de la chaleur.

Certains microcontrôleurs permettent de modifier leur fréquence pendant l'exécution ; c'est une fonctionnalité très intéressante qui permet d'économiser énormément d'énergie en mettant le montage en veille, en attendant par exemple une interruption (appui sur un bouton, détection de lumière, …). La fréquence passera alors à quelques kilohertz, remontant à plusieurs MHz lors d'une utilisation active.

La fréquence peut également être amplifiée par un mécanisme appelé « boucle à verrouillage de phase » (PLL ; Phase-locked loop en anglais). On peut attendre ainsi des fréquences très élevées (2 à 10 fois la fréquence maximale), au prix de la stabilité et de la fiabilité du microcontrôleur.

5.2 horloge

L'horloge peut être soit interne soit externe. L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC. Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 20 MHz selon le type de μC. Le filtre passe bas (Rs, C1, C2) limite les harmoniques dus à l'écrêtage et réduit l'amplitude de l'oscillation, il n'est pas obligatoire.

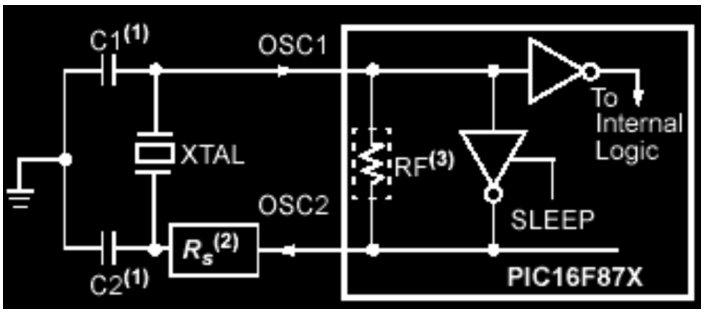


Figure 5.1 : Oscillateur à quartz du PIC 16F87x

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

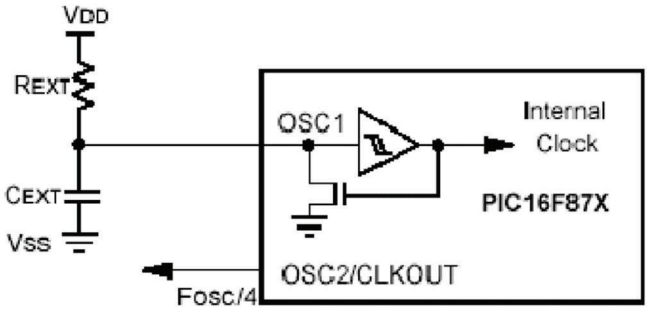


Figure 5.2 : Oscillateur RC du PIC 16F87x

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

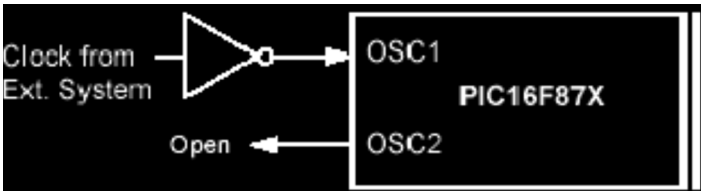


Figure 5.3 : horloge externe du PIC 16F87x

Quel que soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Dans la suite de ce document on utilisera le terme Fosc/4 pour désigner l'horloge système. Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs.

5.3 les Ports d'entrées / sorties

L'une des caractéristiques les plus importantes du microcontrôleur est un nombre de broches Entrée/Sortie utilisés pour la connexion avec les périphériques extérieurs.

Le PIC 16F877A dispose de 35 broches configurables individuellement en Entrées ou Sorties, ce qui est tout à fait suffisant pour la plupart des applications. Ces E/S sont regroupés en cinq ports dits désignées par A, B, C, D et E. Pour des raisons pratiques, de nombreuses broches E/S ont deux ou trois fonctions. Si une broche est utilisée comme une autre fonction, il ne peut être utilisé que dans un but d'entrée/sortie générale broche.

Le µC 16 F84 dispose de 2 PORTS (A,B), 3 PORTS (A,B et C) pour le 16F876 et 5 PORTS (A,B,C,D et E) pour le 16F877. Tous les ports d'entrées sorties Input/ Output sont bidirectionnels (figure 5.4).

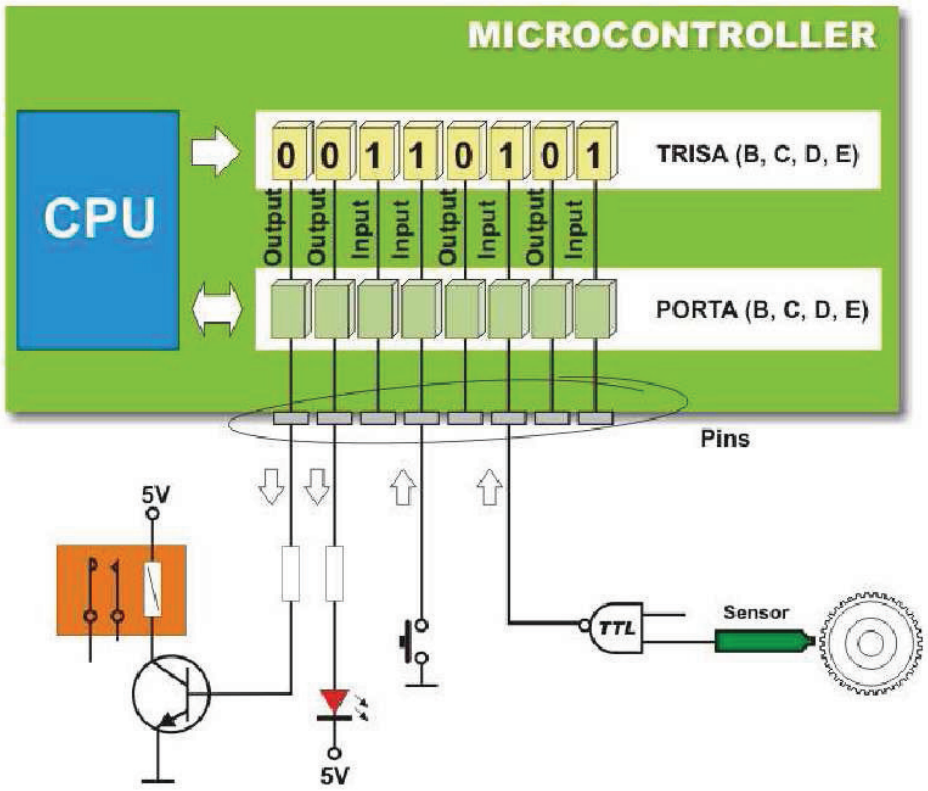


Figure 5.4 : les ports d'E/S.

La plupart des lignes de PORTs ont une double fonction (figure 5.5).

• **Le PORT A** (5 bits) I/O pure et/ou convertisseur analogique et/ou TIMER 0.
La broche RA4 du PORT A (Entrée du TIMER 0 T0CKI) est du type DRAIN OUVERT.
• **Le PORT B** (8 bits) I/O pure et/ou programmation in situ ICSP/ICD (Broche RB3/PGM, RB6/PGC et RB7/PGD) et l'entrée d'interruption externe RB0/INT.
Remarque : Si le PIC est utilisé en mode ICSP/ICD il faut laisser libre les broches RB3/PGM, RB6/PGC ainsi que RB7/PGD) et les configurer en entrée.
• **Le PORT C** (8 bits) I/O pure et/ou TIMER 1 et/ou SPI / I2C et/ou USART.
• **Le PORT D** (8 bits) I/O pure et/ou port parallèle 8 bits associé au PORT E.
• **Le PORT E** (3 bits) I/O pure et/ou pilotage du PORT E RE0/RD, RE1/WR et RE2/CS.
Toutes les lignes de PORTs peuvent fournir un courant de 25mA par ligne de PORT. Une limite de 40mA par PORT doit être respectée pour des questions de dissipation.

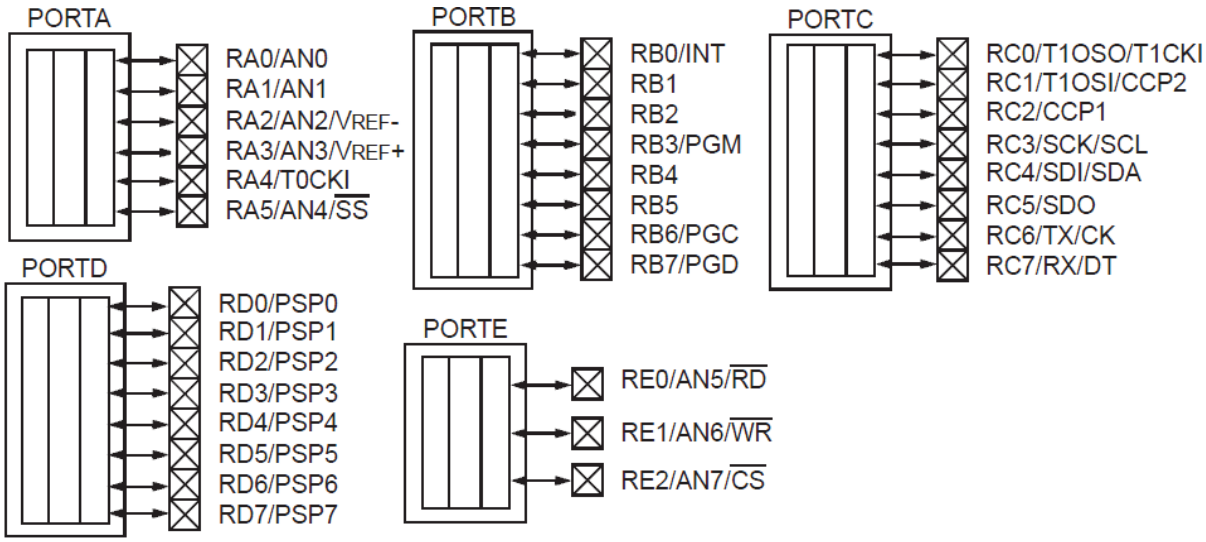


Figure 5.5 : La double fonction de la plupart des lignes des PORTs

Le port d'E/S PORTA

Le port A désigné par PORTA est un port de 6 bits (RA0 à RA5). RA6 et RA7 ne sont pas accessibles. La configuration de direction se fait à l'aide du registre TRISA :
Bit i de TRISA = 0 ►bit i de PORTA configuré en sortie
Bit i de TRISA = 1 ►bit i de PORTA configuré en entrée

En entrée, la broche RA4 peut être utilisée soit comme E/S numérique normale, soit comme entrée horloge pour le Timer TMR0. En sortie, RA4 est une E/S à drain ouvert, pour l'utiliser comme sortie logique, il faut ajouter une résistance de pull-up externe.
Les autres broches (RA0, RA1, RA2, RA3 et RA5) peuvent être utilisées soit comme E/S numériques soit comme entrées analogiques.

Au RESET, ces E/S sont configurées en entrées analogiques. Pour les utiliser en E/S numériques, il faut écrire '00000110' dans le registre ADCON1.
Pour utiliser PORTA en port Numérique (normal), il faut placer 06h dans le registre ADCON1 (bank1)
Quel que soit le mode (Analogique ou Numérique), il faut utiliser le registre TRISA pour configurer la direction des E/S.

Chaque broche du port A configurée en sortie peut fournir un courant de 20 mA au maximum, mais tout le port A configuré en sortie ne peut pas débiter un courant total supérieur à 50 mA. Chaque broche du port A configurée en entrée peut accepter un courant de 25 mA au maximum, mais tout le port A configuré en entrée ne peut pas accepter un courant total supérieur à 80 mA.

Le port d'E/S PORTB

Il comporte 8 bits. Le registre de direction correspondant est TRISB.
Si on écrit un "1" dans le registre TRISB, le driver de sortie correspondant passe en haute impédance.
Si on écrit un "0", le contenu du Latch de sortie correspondante est recopié sur la broche de sortie.

Chaque broche du PORT B est munie d'un tirage au +VDD que l'on peut mettre ou non en service en mode entrée uniquement. On active cette fonction par la mise à "0" du bit 7 dans le registre OPTION en h'81'. Au reset, le tirage est désactivé. Il est inactif quand le port est configuré en sortie.

Les 4 broches PB7 PB6 PB5 et PB4 provoquent une interruption sur un changement d'état si elles sont configurées en ENTREE. On doit remettre à zéro le Flag de cette interruption (bit 0 du registre INTCON en h'0B') dans le programme d'interruption.

Cette possibilité d'interruption sur un changement d'état associé à la fonction de tirage configurable sur ces 4 broches, permet l'interfaçage facile avec un clavier. Cela rend possible le réveil du PIC en mode SLEEP par un appui sur une touche du clavier. Le bit 0 du PORT B peut également être utilisé comme entrée d'interruption externe. Le choix du front de déclenchement se fait en configurant le bit 6 du registre OPTION.

Chaque broche du port B configurée en sortie peut fournir un courant de 20 mA au maximum, mais tout le port B configuré en sortie ne peut pas débiter un courant total supérieur à 100 mA. Chaque broche du port B configurée en entrée peut accepter un courant de 25 mA au maximum, mais tout le port B configuré en entrée ne peut pas accepter un courant total supérieur à 150 mA

Le port d'E/S PORTC

Il s'agit d'un PORT 8 bits bidirectionnel.

► Il est partagé avec le module de transmission synchrone I2C et l'USART.

► La configuration de direction se fait à l'aide du registre TRISC, positionner un bit de TRISC à 1 configure la broche correspondante de PORTC en entré et inversement. Au départ toutes les broches sont configurées en entrée.

Le port d'E/S PORTD

►Le port D désigné par PORTD est un port bidirectionnel de 8 bits (RD0 à RD7).
Toutes les broches sont compatibles TTL et ont la fonction trigger de Schmitt en entrée.

►Chaque broche et configurable en entrée ou en sortie à l'aide du registre TRISD. Pour configurer une broche en entrée, on positionne le bit correspondant dans TRISD à 1 et inversement.

►PORTD n'est pas implémenté sur 16F876, il est disponible sur le 16F877.

►PORTD peut être utilisé dans un mode particulier appelé parallel slave port, pour cela il faut placer le bit PSPMODE (bit 4) de TRISE à 1. Dans ce cas les 3 bits de PORTE deviennent les entrées de control de ce port (RE, WE et CS)

Pour utiliser PORTD en mode normal, il faut placer le bit PSPMODE de TRISE à 0

Le port d'E/S PORTE

PORTE contient seulement 3 bits RE0, RE1 et RE2. Les 3 sont configurables en entrée ou en sortie à l'aide des bits 0, 1 ou 2 du registre TRISE.

►PORTE n'est pas implémenté sur 16F876, il est disponible sur le 16F877.
►Les 3 bits de PORTE peuvent être utilisés soit comme E/S numérique soit comme entrées analogiques du CAN. La configuration se fait à l'aide du registre ADCON1.
►Si le bit PSPMODE de TRISE est placé à 1, Les trois bits de PORTE deviennent les entrées de control du PORTD qui (dans ce cas) fonctionne en mode *parallel Slave mode*
►A la mise sous tension (RESET), les 3 broches de PORTE sont configurées comme entrées analogiques.

Pour utiliser les broches de PORTE en E/S numériques normales :
- Placer 06h dans ADCON1
- Placer le bit PSPMODE de TRISE à 0

5.3.1 Configuration des PORTx , les registres PORTx et TRISx.

Chaque port a son directeur, c'est à dire le registre correspondant TRIS : TRISA, TRISB, etc TRISC qui détermine l'état (entrée ou sortie), mais pas le contenu des bits de port.

En désactivant certains bits du registre TRIS (bit = 0), la broche de port correspondant est configurée comme sortie. De même, en définissant un certain bit du registre TRIS (bit = 1), le pin du port correspondant est configuré en tant qu'entrée. Cette règle est facile à retenir 0 = Output, 1 = Input.

Le registre de PORTx, si le PORT x ou certaines lignes de PORT X sont configurées en sortie, ce registre détermine l'état logique des sorties.

- Le registre TRISx, c'est le registre de direction. Il détermine si le PORTx ou certaines lignes de port sont en entrée ou en sortie. L'écriture d'un 1 logique correspond à une entrée (1 comme Input) et l'écriture d'un 0 logique correspond à une sortie (0 comme Output).
Au RESET toutes les lignes de ports sont configurées en entrées.

Remarque : Les registres TRISx appartiennent à la BANQUE 1 des SFR. Lors de l'initialisation du µC il ne faut pas oublier de changer de page mémoire pour les configurer.
Au RESET du Mc, les ports PORTx sont configurés en entrée.

Exemple1 : On souhaite obtenir la configuration suivante des PORTA et PORTB.

SENS	NC	NC	S	E	S	S	S	E
PORTA	NC	NC	RA5	RA4	RA3	RA2	RA1	RA0
SENS	E	E	S	S	E	S	S	E
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RBO

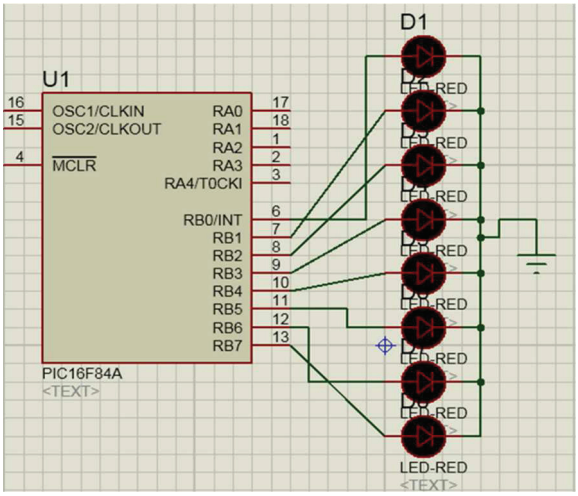
Source code 5.1 :

```
void main()
{
TRISA=0b11010001;  // registre de configuration, Configuration du PORTA X X S E S S S E
TRISB=0b11001001;  // registre de configuration
PORTA=0b11010001; // registre de travail.
PORTB=0b11001001; // registre de travail.
}
```

Exemple2 : On souhaite réaliser un jeu de lumière avec des LEDs.

Source code 5.2 :

```
void main()
{
TRISB=0x00;  // registre de configuration
                  // TRISB=0b00000000; TRISB=0 ;
PORTB=0x00; // registre de travail.
for(;;)
{
PORTB=0b00000001; delay_ms(1000);
PORTB=0b00000010; delay_ms(1000);
PORTB=0b00000100; delay_ms(1000);
PORTB=0b00001000; delay_ms(1000);
PORTB=0b00010000; delay_ms(1000);
}
}
```



5.4 Le Port Parallèle Esclave (PSP : Parallel Slave Port)

Le Port Parallèle Esclave est un port 8 bits permettant d'interfacer le PIC avec, par exemple, un autre microprocesseur. Les données transitent via les lignes PSP0 à PSP7, qui physiquement utilisent les mêmes broches que le PORTD.

Le flux de données est contrôlé par les lignes RD, WR et CS qui correspondent aux broches du PORTE. C'est le microprocesseur externe qui est le chef d'orchestre : il valide notre PIC par la ligne CS (Chip Select), et indique au PIC s'il lit ou écrit grâce aux lignes RD (Read) et WR (Write). D'où l'appellation de port parallèle ESCLAVE. Esclave, puisque c'est le microprocesseur externe qui donne les ordres, notre PIC ne fait qu'exécuter.

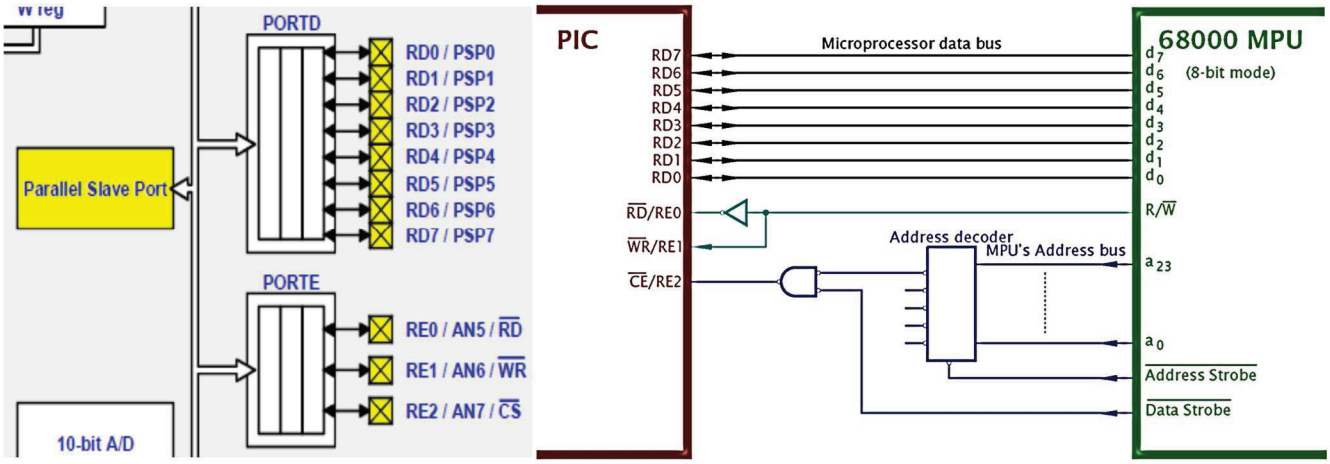


Figure 5.6 : Le Port Parallèle Esclave (PSP : Parallel Slave Port).

5.5 Le module USART

L'USART (Universal Synchronous Asynchronous Receiver Transmitter) est l'un des deux modules de communication série ou SCI en anglais (Serial Communication Interface) dont dispose le PIC 16F876/877.

Comme son nom l'indique, elle peut établir une liaison synchrone ou asynchrone, recevoir et transmettre des données, selon la manière dont elle est configurée. C'est son côté « je peux tout faire » qui lui vaut l'attribut « Universal ». Concrètement, l'USART permet de communiquer avec le reste du monde : un ordinateur ou tout autre matériel équipé d'une interface série RS232, des circuits intégrés convertisseurs Numérique/Analogique ou Analogique/Numérique, des EEPROMs série… La communication se fait sur les deux broches RC6/TX et RC7/RX qui doivent être configurés toutes les deux en ENTREE par TRISC.

L'USART peut être configurée selon 3 modes :

►Asynchrone (Full Duplex)

►Synchrone Maître (Half Duplex)

►Synchrone Esclave (Half Duplex)

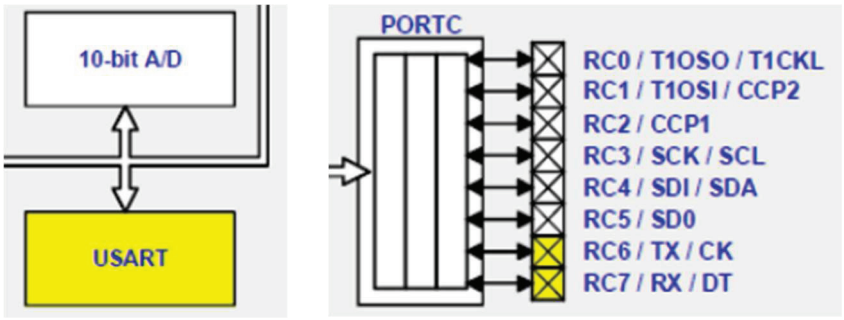


Figure 5.7 : Le Module USART.

5.6 Le module SSP (Synchronous Serial Port)

Le module SSP est la deuxième interface de communication série du PIC. En fait, il s'agit d'un port Maître, donc l'appellation exacte est plutôt MSSP pour « Master Synchronous Serial Port ». Il est utile pour communiquer avec d'autres modules ou microcontrôleurs, des EEPROMs séries, des registres à décalage, des afficheurs, des convertisseurs A/N ou N/A…

Il peut fonctionner dans deux modes de communication :
►Le mode SPI (Serial Peripheral Interface).
►Le mode I2C (Inter-Integrated Circuit).

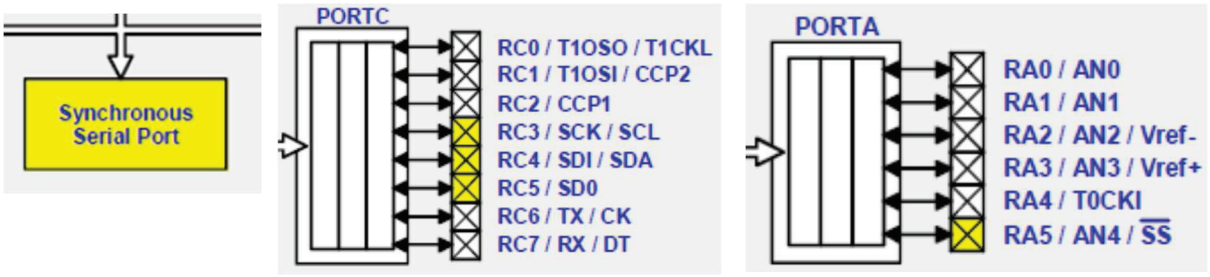
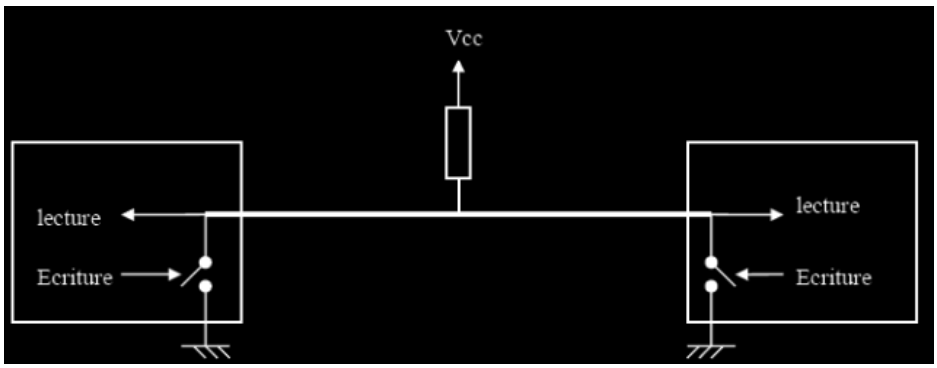


Figure 5.8 : Le Module SSP (Synchronous Serial Port).

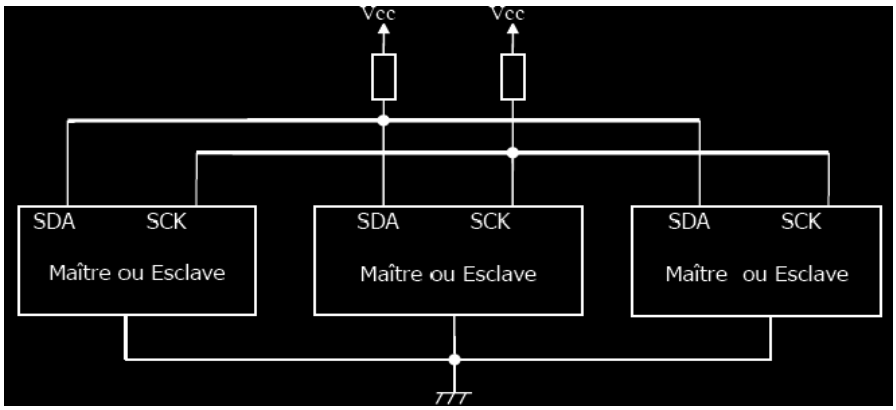
Le Mode I2C

Avant de parler du module MSSP en mode I2C du PIC, introduisons très brièvement Le standard I2C Le bus I²C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement 3 fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse). Comme les lignes SDA et SCK sont utilisées dans les deux sens par les deux circuits qui communiquent, on peut avoir un circuit qui place la ligne à 1 (Vcc) et l'autre qui la place à 0 (masse) ce qui correspond à un court-circuit qui peut détruire les deux composants. Pour éviter ce problème, les E/S SDA et SCK fonctionnent en mode collecteur ouvert (ou drain ouvert) de sorte qu'un circuit ne peut imposer que le niveau bas ou ouvrir la ligne, le niveau haut est obtenu par une résistance de tirage externe. Ainsi une ligne est à 0 quand un des deux circuits impose le 0. Elle passe à 1 quand les deux circuits imposent le 1 (circuit ouvert). Le protocole I2C jongle avec cette situation pour organiser l'échange des données entre les deux composants.



Un bus I2C peut être relié à plusieurs circuits, mais pendant une communication, un des circuits est le maître, c'est lui génère l'horloge et initie les séquences de transmission, l'autre est l'esclave, il subit l'horloge du maître sur la ligne SCK mais il peut tout de même recevoir et émettre des données sur la ligne SDA.

Chaque esclave a une adresse, au début d'une séquence de communication, le maître qui initie la séquence envoie l'adresse du slave avec lequel il désire communiquer, celui-ci reconnaît son adresse et répond, les autres slaves (s'il y en a) restent muets.



Certains circuits sont fabriqués pour être des masters, d'autres des slaves et d'autres peuvent être soit l'un soit l'autre. Pour prendre le contrôle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1'). Lorsqu'un circuit prend le contrôle du bus, il en devient le maître. C'est lui qui génère le signal d'horloge et c'est lui qui initie les séquences d'échange.

L'utilisation de l'un de ces deux modules de communication, USART ou MSSP dépend donc essentiellement du protocole de communication nécessaire. Schématiquement, l'USART est bien adaptée pour communiquer avec le reste de l'univers via la célèbre interface RS232 alors que le MSSP permet de communiquer aisément avec d'autres composants électroniques à interface série.

Une dernière remarque par rapport aux interfaces de communication : certains PIC (mais pas le 16F877) intègrent une interface USB ou un module de communication Ethernet.

Mise en œuvre

Exemple de communication entre deux PIC :

L'exemple suivant montre un exercice d'application d'une communication série entre deux PIC, écriture et lecture. Le programme est réalisé sur mikroC.

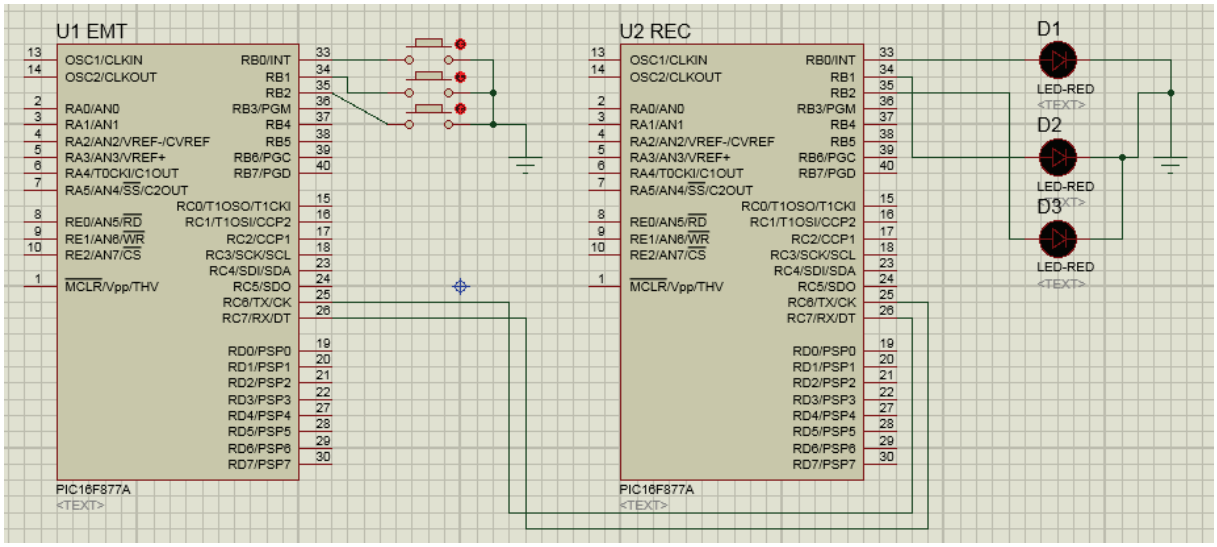


Figure 5.9 : Exemple de communication entre deux PIC 16F877A.

Source code 5.2 :

L'état initiale de RB0 RB1 RB2 est 111 sera reporté sur le deuxième PIC avec la communication série.se qui implique l'allumage des LEDs si l'un des interrupteurs est enfoncé la LED qui corresponde s'éteindra.

Programme U1

```
void main() {
      TRISB=0xFF ;
Option_reg.b7=0 ; // résistances internes active du PORTB
UART1_init(8929) ; // 9615 vitesse de transmission des données
 Delay_ms (100) ; //Pour la stabilisation
for( ;;)
{
UART1_write(PORTB) ;
Delay_ms(500);
}

}
```

Programme U2

```
void main() {
   TRISB=0x00;
UART1_init(8929) ;
for( ;;)
{
if (UART1_data_ready())
{
PORTB=UART1_read();
}
}

}
```

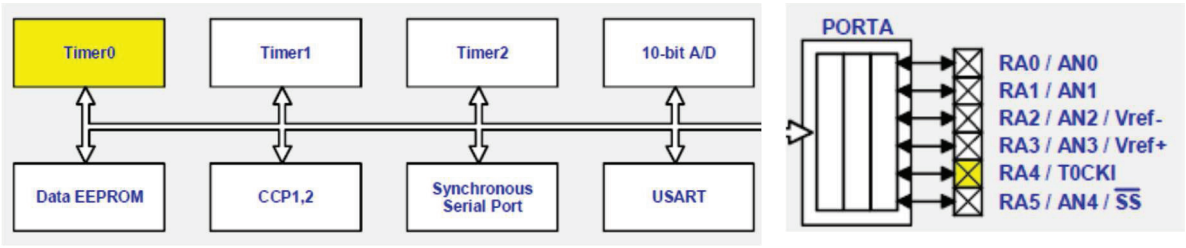
5.7 Les Timers

Dans la majeure partie des applications, il est nécessaire de contrôler le temps ; afin de ne pas occuper le microcontrôleur qu'à cette tâche (boucle de comptage qui monopolise le micro), on le décharge en utilisant un Timer. Le pic 16F84 par exemple dispose de deux Timers, un à usage général (le TMR0) et un autre utilisé pour le chien de garde (watch dog WDG).

5.7.1 Le Timer TMR0

Le TMR0 est un compteur ascendant (qui compte) de 8 bits (de 0 à 256) qui peut être chargé avec une valeur initiale quelconque. Il compte des impulsions soit internes, soit d'une source externe. Il est ensuite incrémenté à chaque coup d'horloge jusqu'à ce que le débordement ait lieu (passage de FF à 00).

►On peut par ailleurs lui appliquer une pré-division programmable entre 1 et 256.
►On peut librement lire ou écrire dans le registre de comptage associé.
►On peut donc le pré charger avec une valeur, à partir de laquelle il comptera jusqu'à atteindre 255.
►Une fois le registre de comptage plein, il peut générer une interruption.
►Étant donné qu'une fois configurer il fonctionne quasi-indépendamment du microprocesseur, on pourra s'en servir comme base de temps



C'est un compteur ayant les caractéristiques suivantes :

►Il est incrémenté en permanence soit par l'horloge interne Fosc/4 (mode timer) soit par une horloge externe appliquée à la broche RA4 du port A (mode compteur).

Le choix de l'horloge se fait à l'aide du bit T0CS du registre OPTION_REG

T0CS = 0 ► horloge interne
T0CS = 1 ► horloge externe appliquée à RA4

►Dans le cas de l'horloge externe, Le bit T0SE du registre OPTION_REG permet de choisir le front sur lequel le TIMER s'incrémente.

T0SE = 0 ► incrémentation sur fronts montants
T0SE = 1 ► incrémentation sur fronts descendants

►Quel que soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport DIV est fixés par les bits PS0, PS1 et PS2 du registre OPTION_REG

Figure 5. 10 Le registre OPTION_REG

L'affectation ou non du prédiviseur se fait à l'aide du bit PSA du registre OPTION_REG

PSA = 0 ► on utilise le prédiviseur
PSA = 1 ►pas de prédiviseur (affecté au chien de garde)

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

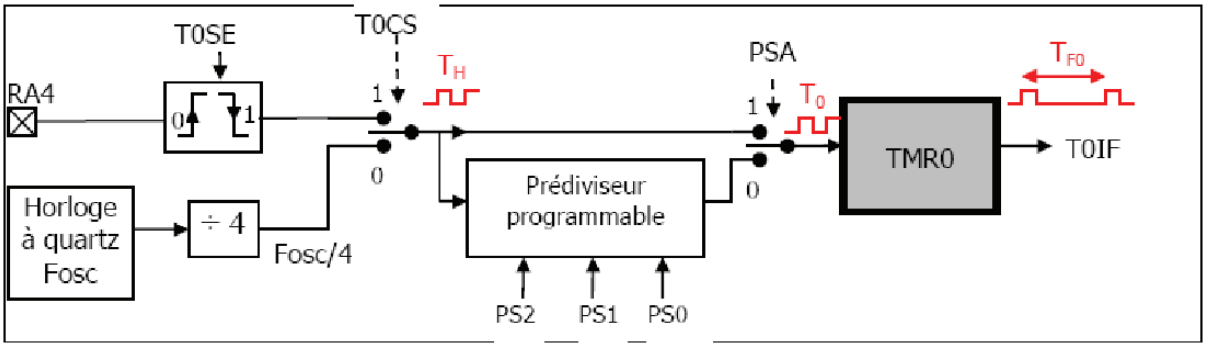


Figure 5. 11 Le Timer TMR0

En résumé, chaque fois que le compteur complète un tour, le drapeau T0IF se lève. Si on note TH la période de l'horloge source, T0 l'horloge de TMR0 et TF0 le temps qui sépare 2 levés de drapeau successifs :

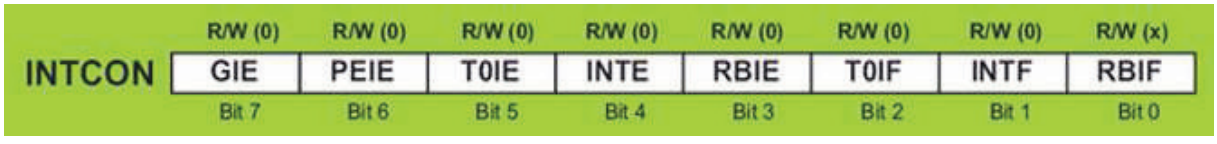
►Sans prédiviseur : TF0 = 256 T0 = 256 TH
►Avec prédiviseur : TF0 = 256 T0 = 256 x (DIV x TH)
►Avec prédiviseur et compteur N dans le programme : TF0 = N x 256 x (DIV x TH)

Mise en œuvre

Exemple d'application Montre numérique

Configuration du TMR0

Pour mettre en œuvre le TMR0 du PIC Il faut mettre **T0IE** et **T0IF** à 1 du registra INTCON. Ouvrir **GIE** et **INTE**. INTCON=0b10110100 ; et par la suite OPTION_REG comme on a vue au part avant. OPTION_reg=0b01010111;



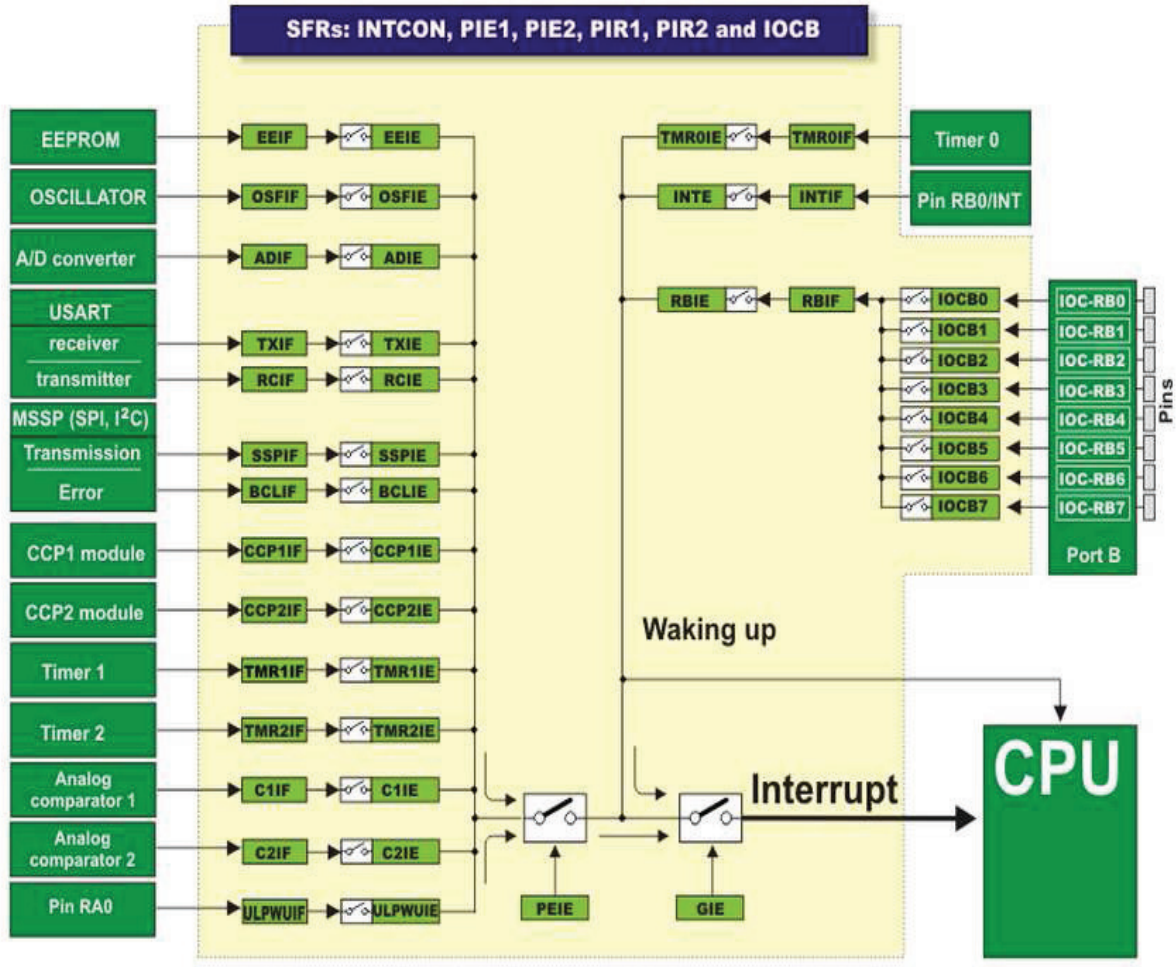
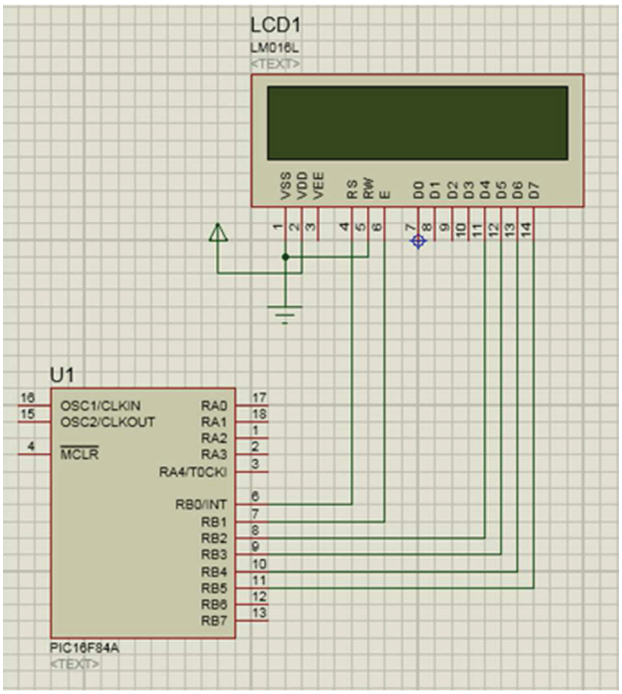


Figure 5. 12 : Configuration TMR0

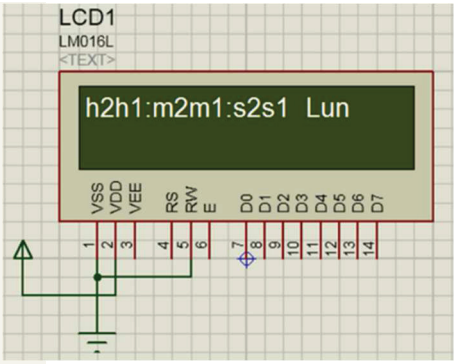
La connexion d'un LCD avec un PIC nécessite un code source pour une bonne communication qui est écris avant le programme principal (source HELP mikroC) :

Source code 5.3 :

```
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB1_bit;
sbit LCD_D7 at RB5_bit;
sbit LCD_D6 at RB4_bit;
sbit LCD_D5 at RB3_bit;
sbit LCD_D4 at RB2_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB1_bit;
sbit LCD_D7_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB2_bit;
```



Source code 5.4 :




```
char s1,s2,p,m1,m2,h1,h2,j   ;
void interrupt ()
{
if ( INTCON.b1==1)  // pour ouvrire le FLAG.
       {
          j++;
        }
INTCON.b1=0; // pour fermer le flag.
}

void main()
{
         INTCON=0b10010000 ;
         S1=s2=m1=m2=h1=h2=48; //code ascci de 0 initialisation
         P=58 ; // code ascii de :
         J=1 ;
         LCD_Init();
         Lcd_Cmd(_LCD_CURSOR_OFF);
Lcd_chr(1,8,s1); Lcd_chr(1,7,s2); Lcd_chr(1,6,p); Lcd_chr(1,5,m1);
Lcd_chr(1,4,m2); Lcd_chr(1,3,p); Lcd_chr(1,2,h1); Lcd_chr(1,1,h2);
for(;;) // For san conditions
{
delay_ms (1000);
s1++;
if(s1==58)   {s1=48;  s2++;}
if(s2==54)   {s1=48; s2=48; m1++;}
if(m1==58)  { s1=48; s2=48;  m1=48; m2++;}
if(m2==54)  { s1=48; s2=48;  m1=48; m2=48; h1++;}
if(h1==58)   { s1= s2=m1=m2=h1=48; h2++;}
if(h2==50&&h1==52)  { s1=s2=m1=m2=h1=h2=48; j++;}
Lcd_chr(1,8,s1); Lcd_chr(1,7,s2); Lcd_chr(1,5,m1);
Lcd_chr(1,4,m2); Lcd_chr(1,2,h1); Lcd_chr(1,1,h2);
if (j==1) Lcd_out (1,10,"Lundi");if (j==2) Lcd_out (1,10,"Mar");
if (j==3) Lcd_out (1,10,"Mer");if (j==4) Lcd_out (1,10,"Jeu");
if (j==5)  Lcd_out (1,10,"Ven");if (j==6) Lcd_out (1,10,"sam");
if (j==7)  Lcd_out (1,10,"Dim");if (j==8) j=1;
}
}
```

Le programme précédant code source 5.3 et 5.4 présente un handicape temporel puisque l'exécution de l'instruction de l'incrémentation (S1++) nécessite le passage du programme par plusieurs instructions pour la vérification et cela implique un temps perdu de lords du µseconde ; l'accumulation de ce temps rend ce programme défaillant du point de vu un temps important. L'utilisation de Timer TMR0 corrige cette erreur temporelle puisqu'une fois configurer il fonctionne quasi-indépendamment du programme principal ; Ainsi, on pourra s'en servir comme base de temps.

Le calcul de la variable temporel t

►f=1000000hz/256 = 3906,25 Hz représente la Fréquence du compteur

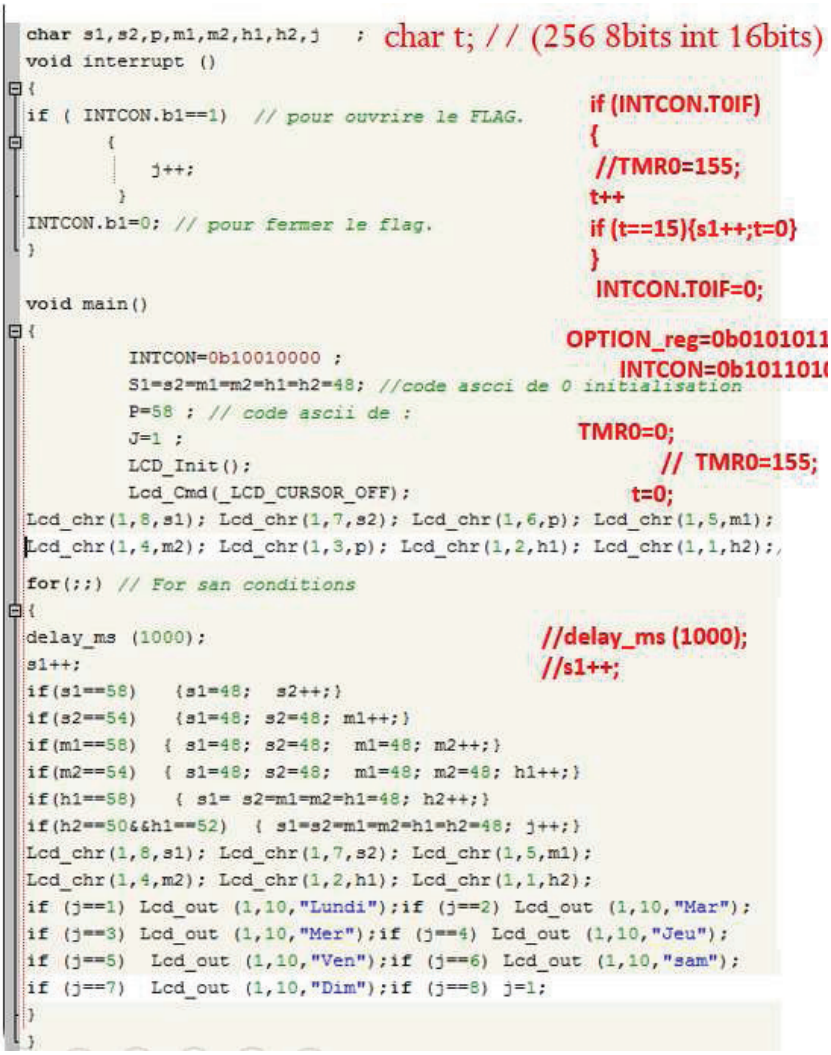
T=1/3906,25=0.000256 s *256 = 0,065536s représente le Temps entre une interruption et une autre

Pour avoir 1 second pour chaque interruption il faut donc faire (1/0.65536=15.25) et t = 15.25

►Si le comptage du Timer commence à partir d'une valeur précharger ex : TMR0=155;

255-155=100 représente les incrémentations.

0.000256*100 =0.0256s représente le Temps de comptage de 0 à 100 et le Temps entre une interruption et une autre t=1/0.0256=39,0625.



```
char s1,s2,p,m1,m2,h1,h2,j   ; char t; // (256 8bits int 16bits)
void interrupt ()
{
if ( INTCON.b1==1)  // pour ouvrire le FLAG.
       {
          j++;
        }
INTCON.b1=0; // pour fermer le flag.
}

void main()
{
         INTCON=0b10010000 ;
         S1=s2=m1=m2=h1=h2=48; //code ascci de 0 initialisation
         P=58 ; // code ascii de :
         J=1 ;
         LCD_Init();
         Lcd_Cmd(_LCD_CURSOR_OFF);
Lcd_chr(1,8,s1); Lcd_chr(1,7,s2); Lcd_chr(1,6,p); Lcd_chr(1,5,m1);
Lcd_chr(1,4,m2); Lcd_chr(1,3,p); Lcd_chr(1,2,h1); Lcd_chr(1,1,h2);
for(;;) // For san conditions
{
delay_ms (1000);
s1++;
if(s1==58)   {s1=48;  s2++;}
if(s2==54)   {s1=48; s2=48; m1++;}
if(m1==58)  { s1=48; s2=48;  m1=48; m2++;}
if(m2==54)  { s1=48; s2=48;  m1=48; m2=48; h1++;}
if(h1==58)   { s1= s2=m1=m2=h1=48; h2++;}
if(h2==50&&h1==52)  { s1=s2=m1=m2=h1=h2=48; j++;}
Lcd_chr(1,8,s1); Lcd_chr(1,7,s2); Lcd_chr(1,5,m1);
Lcd_chr(1,4,m2); Lcd_chr(1,2,h1); Lcd_chr(1,1,h2);
if (j==1) Lcd_out (1,10,"Lundi");if (j==2) Lcd_out (1,10,"Mar");
if (j==3) Lcd_out (1,10,"Mer");if (j==4) Lcd_out (1,10,"Jeu");
if (j==5)  Lcd_out (1,10,"Ven");if (j==6) Lcd_out (1,10,"sam");
if (j==7)  Lcd_out (1,10,"Dim");if (j==8) j=1;
}
}
```

```
if (INTCON.T0IF)
{
 //TMR0=155;
t++
if (t==15){s1++;t=0}
}
 INTCON.T0IF=0;

OPTION_reg=0b0101011
   INTCON=0b1011010

TMR0=0;
      // TMR0=155;
    t=0;

//delay_ms (1000);
//s1++;
```

Exemple 2 :

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

5.7.2 Le Watchdog Timer (chien de garde)

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

►Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur par un processus indésirable non contrôlé

►Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent

exploitée pour réaliser des temporisations. L'horloge du WDT a une période voisine de 70 µs ce donne un Time-Out toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passé le signal Time-Out dans un prédiviseur programmable (partagé avec le timer TMR0). L'affectation se fait à l'aide du bit PSA du registre OPTION_REG, Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (voir tableau précédent)

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

Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG.
L'utilisation du WDT doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un WDT timeOut lors de l'exécution d'un programme, on a deux possibilités :
Inhiber le WDT d'une façon permanente en mettant à 0 le bit WDTE dans l'EEPROM de configuration

►Remettre le WDT à 0 périodiquement dans le programme à l'aide de l'instruction CLRWDT pour éviter qu'il ne déborde

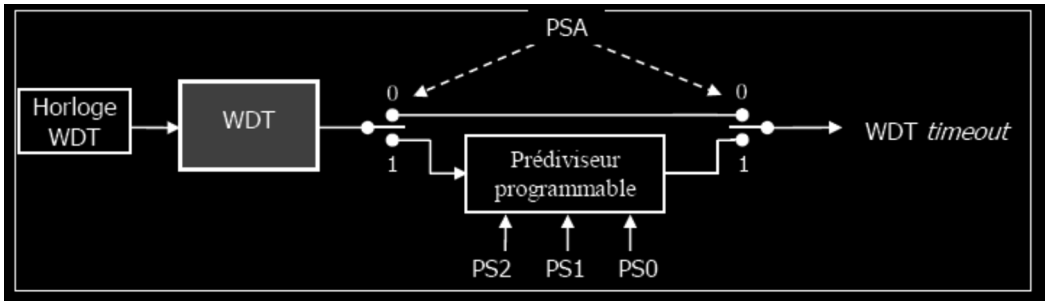
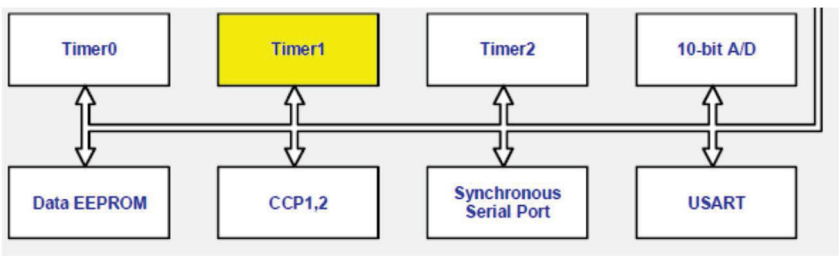


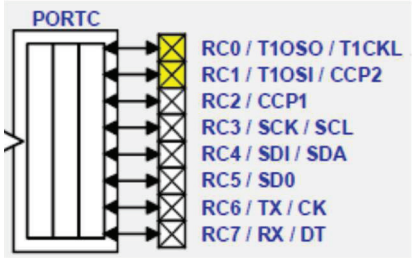
Figure 5. 13 : Le Watchdog Timer.

5.7.3 Le Timer TMR1

TMR1 est un Timer/Compteur 16 bits accessible en lecture/écriture par l'intermédiaire des registres 8 bits TMR1H (bank0) et TMR1L (bank0) qui constituent sa partie haute et sa partie basse.

Le registre TMR1 (constitué de TMR1H et TMR1L) s'incrémente de h'0000' jusqu'à h'FFFF' et repasse ensuite à h'0000' pour continuer le comptage. Ce module peut fonctionner en mode TIMER, quand il s'incrémente à chaque cycle instruction (Fosc/4 avec le pré diviseur considéré à "1") ou en mode compteur, quand il s'incrémente à chaque front montant de l'horloge externe appliquée sur le Port C0.
L'horloge externe peut également être l'oscillateur interne, dont la fréquence est fixée par un quartz externe branché entre la broche Port C0 et la broche Port C1. Le contrôle du TIMER 1 se fait par le registre T1CON en h'10' (bank0).





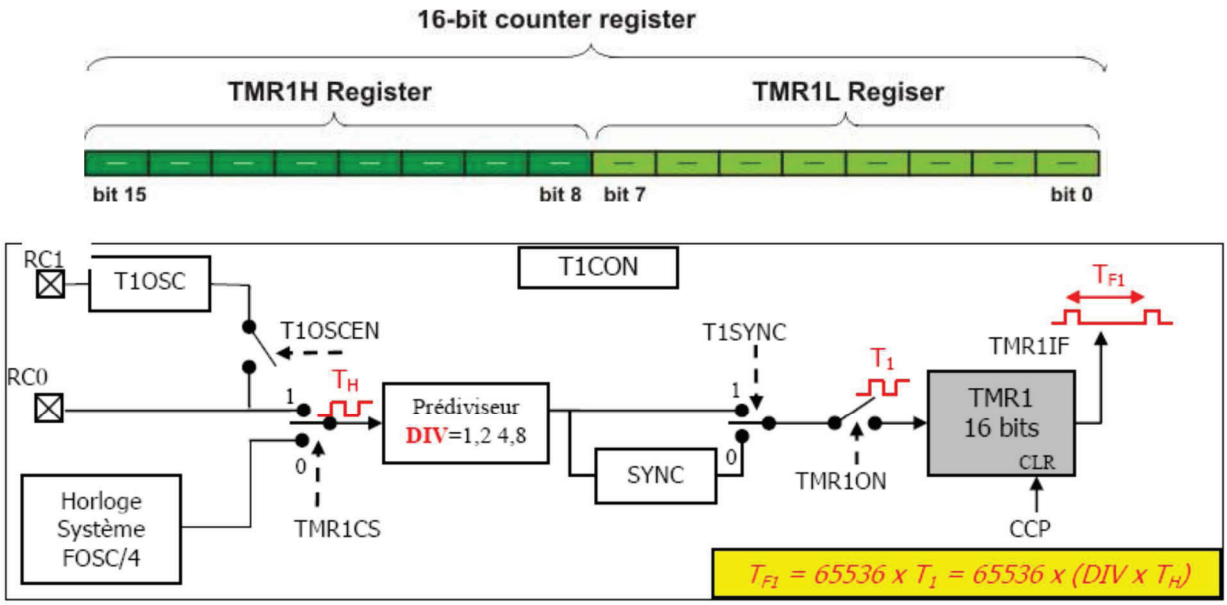


Figure 5. 14 : Le Timer TMR1.

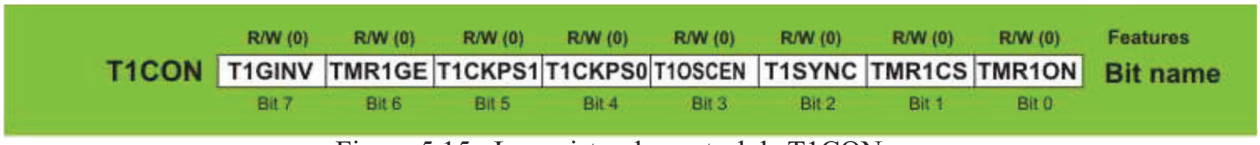


Figure 5.15 : Le registre de control de T1CON

T1CKPS1, T1CKPS0 : Control du prescaler
00 : division par 1
01 : division par 2
10 : division par 4
11 : division par 8

T1OSCEN : Validation de l'Oscillateur associé à TMR1
0 : Oscillateur arrêté
1 : Oscillateur activé

T1SYNC : Synchronisation de l'horloge externe (ignoré en mode timer)
0 : Synchronisation
1 : pas de synchronisation

TMR1CS : Choix de l'horloge du Timer
0 : horloge système (Fosc/4) : mode timer
1 : Horloge externe : mode compteur

TMR1ON : Démarrer arrêter le timer
0 : Timer stoppé
1 : Timer en fonctionnement

Oscillateur Interne du Timer 1 :

Un oscillateur à quartz a été embarqué sur le chip. Il est branché entre les broches PC0 (oscillateur out) et PC1 (oscillateur in). Il est mis en service par la mise à "1" du bit T1OSCEN. Cet oscillateur à faible consommation est limité à 200 KHz. Il continu à osciller en mode SLEEP du PIC. Il est principalement destiné pour générer un événement temps réel toutes les secondes par utilisation d'un quartz 32,768 KHz

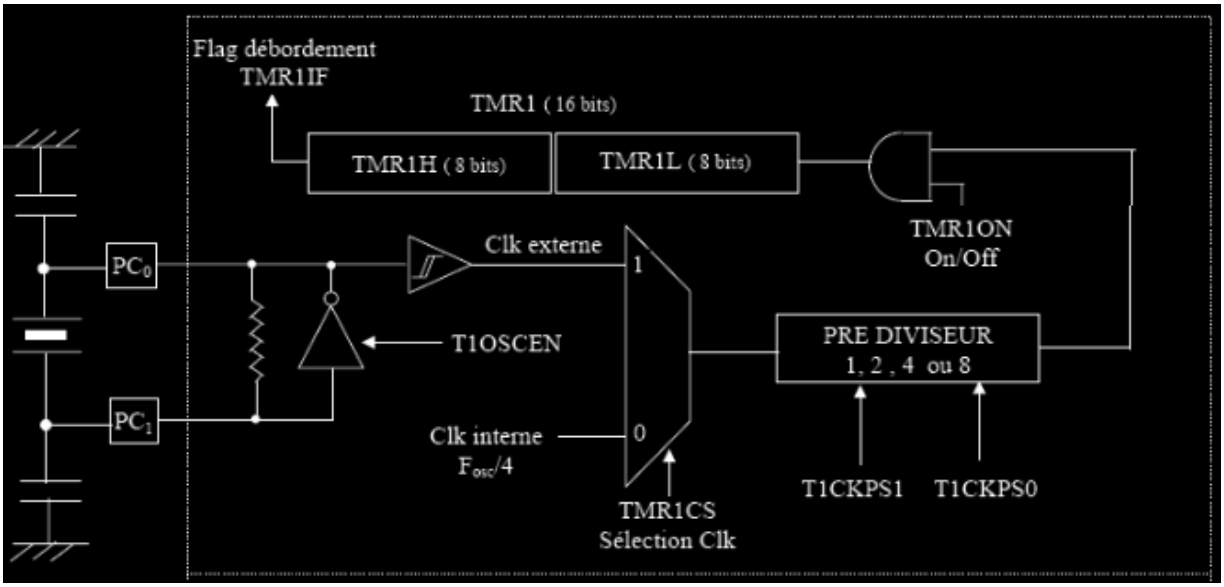


Figure 5.16 : Oscillateur Interne du Timer 1

5.7.4 Le Timer TMR2

Le module Timer 2 est un compteur 8 bits avec pré diviseur et post diviseur. On s'en sert pour générer des signaux carrés, ou, en association avec le module CCP, des signaux PWM. PWM étant l'acronyme de « Pulse Width Modulation » ou, en français, Modulation de Largeur d'Impulsion (MLI).

Ce compteur TMR2 en h'11 ' page 0 est un registre en lecture ou écriture. Il possède Un registre 8 bits pour la période : PR2 en h'92' page 1. Le compteur s'incrémente de h'00' jusqu'à la valeur contenue par PR2 et repasse ensuite à "0" pour continuer le comptage. Au reset PR2 est initialisé à "FF". L'entrée du compteur est l'horloge cycle interne : Fosc/4 qui passe à travers un pré diviseur programmable par 1, 4 ou 16. La sortie du compteur passe dans un post diviseur programmable sur 4 bits entre 1 et 16.

Quand la sortie du compteur passe par la valeur programmée dans PR2, il y a génération d'une interruption (si elle a été autorisée par TMR2IE=1) et le flag TMR2IF est positionné à "1". Ceci bien entendu en considérant le post diviseur programmé à "1". Le contrôle du Timer 2 se fait par le registre T2CON en h'12' page 0.

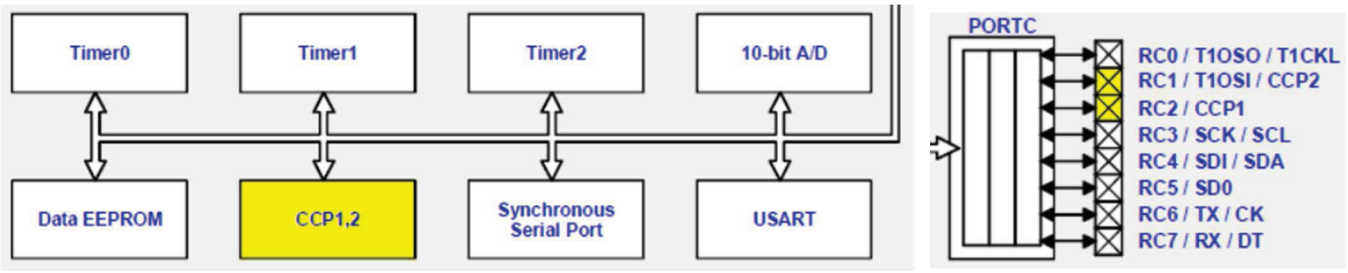
		R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Figure 5.17 : Le registre de control de T2CON

Au reset : T2CON = 00000000
bit 7 : **bit non implémenté.**
bit 6 à bit 3 : **TOUTPS :** Programmation du Post diviseur.
0 0 0 0 = post divise par 1.
0 0 0 1 = post divise par 2.
0 0 1 0 = post divise par 3.

. . . .
1 1 1 1 = post divise par 16.
bit 2 : **TMR2ON :** mise en service du Timer 2.
1= Timer 2 : On.
0= Timer 2 : Off.
bit 1 et bit0 : **T2CKPS :** Programmation du pré diviseur.
0 0= pré divise par 1.
0 1= pré divise par 4.
1 X= pré divise par 16.

5.7.5 Le module CCP (CAPTURE COMPARE et PWM)



Il y a deux modules identiques **CCP1** et **CCP2** composés chacun d'un registre 16 bits. Ils peuvent opérer soit comme un registre 16 bits de **capture**, soit comme un registre 16 bits de **comparaison**, soit enfin comme un **registre 8 bits** pour générer du **PWM**.

►Le module **CCP1** est constitué de deux registres de 8 bits : CCPR1L en h'15' page 0 et CCPR1H en h'16' page 0. Ce module est contrôlé par le registre CCP1CON en h'17' page 0. La sortie en mode COMPARE ou mode PWM et l'entrée en mode CAPTURE se font par la broche PC2.

►Le module **CCP2** est constitué de deux registres de 8 bits : CCPR2L en h'1B' page 0 et CCPR2H en h'1C' page 0. Ce module est contrôlé par le registre CCP2CON en h'1D' page 0.

►La sortie en mode COMPARE ou mode **PWM** et l'entrée en mode CAPTURE se font par la broche PC1.
En mode COMPARE ou CAPTURE, les modules utilisent le TIMER 1. En mode PWM, ils utilisent le TIMER 2. Les registres de contrôles CCP1CON et CCP2CON sont identiques. On ne décrira que CCP1CON. CCP1CON : (h'17' : page 0). (et CCP2CON en h'1D' : page 0.)

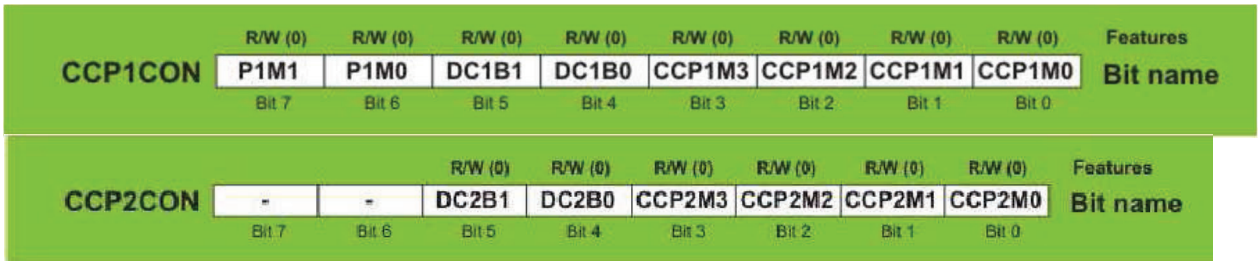


Figure 5.18 : Les registre de control CCP1CON et CCP2CON.

Au reset : CCP1CON = 00000000

bit 7 et bit 6 : **bits non implémentés.**

bit 5 et bit 4 : **CCP1X et CCP1Y :** Bits non utilisés en modes Compare et Capture. Ce sont les 2 bits LSB pour le Duty cycle en mode PWM. Les 8 bits MSB sont dans le registre CCPR1L en h'15' page 0.

bit 3 à bit 0 : **CCP1M3 à CCP1M0 :** bits de sélection du mode.
0 0 0 0 = Module CCP stoppé.
0 1 0 0 = Mode Capture à chaque front descendant.
0 1 0 1 = Mode Capture à chaque front montant.
0 1 1 0 = Mode Capture tous les 4 fronts montants.
0 1 1 1 = Mode Capture tous les 16 fronts montants.
1 0 0 0 = Mode Compare. Pin de sortie mise à "1" et Flag CCP1IF = 1 à l'égalité.
1 0 0 1 = Mode Compare. Pin de sortie mise à "0" et Flag CCP1IF = 1 à l'égalité.
1 0 1 0 = Mode Compare. Génération d'une Interrup. et Flag CCP1IF = 1 à l'égalité.
1 0 1 1 = Mode Compare. Evénement spécial généré et Flag CCP1IF = 1 à l'égalité.
1 1 x x = Mode PWM.

MODE COMPARE :

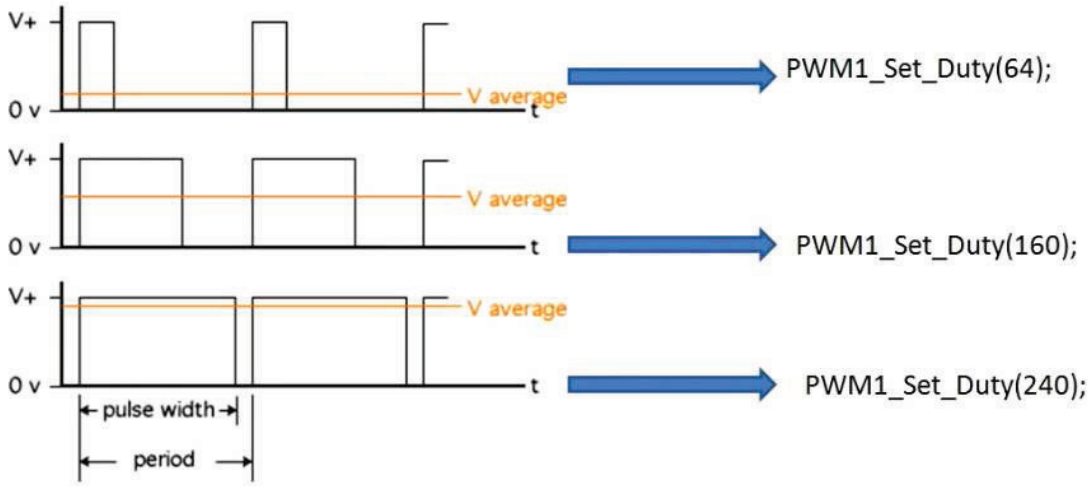
Les deux modules CCP étant identiques on ne décrira que le module 1.
Les 16 bits des registres CCPR1 (CCPR1H et CCPR1L) sont constamment comparés avec le valeur sur 16 bits des registres du Timer 1 (TMR1H et TMR1L). Quand il y a égalité, la broche préalablement programmée en sortie PC2, passe soit à "1" soit à "0" suivant la configuration des 4 bits CCP1M du registre CCP1CON. Au même instant le Flag CCP1IF est mis à "1".
En mode Compare, les événements spéciaux générés quand il y a égalité sont :
- Pour CCP1: reset du Timer 1.
- Pour CCP2 : reset du Timer 1 et démarrage d'une conversion A/D.
Dans ce cas la broche de sortie n'est pas affectée, mais le Flag CCP1IF est mis à "1". Il est rappelé que ce Flag doit être remis à "0" par soft.

MODE CAPTURE :

Quand un événement extérieur apparaît sur la broche préalablement programmée en entrée PC2, la valeur des 16 bits des registres du Timer 1 (TMR1L et TMR1H) est recopiée dans les registres CCPR1 (CCPRIH et CCPR1L). Cet événement est programmable par les 4 bits CCP1M du registre CCP1CON. La capture peut avoir lieu à chaque front descendant, à chaque front montant, tous les 4 ou tous les 16 fronts montants. Quand la capture a eu lieu, le flag CCP1IF est mis à "1". Ce bit doit être remis à "0" par soft. Si une nouvelle capture survient alors que la valeur dans CCPR1 n'a pas été lue, l'ancienne valeur est perdue. Les fonctions de Capture et de Compare sur le Timer 1 par les modules CCP1 et CCP2 peuvent générer une interruption quand le Flag CCP1IF passe à "1" si le bit d'autorisation CCP1IE du registre PIE1 est mis à "1".

MODE PWM :

Un signal PWM est caractérisé par une période, et un temps de travail ou le signal est à "1". Ce temps est appelé DUTY CYCLE.

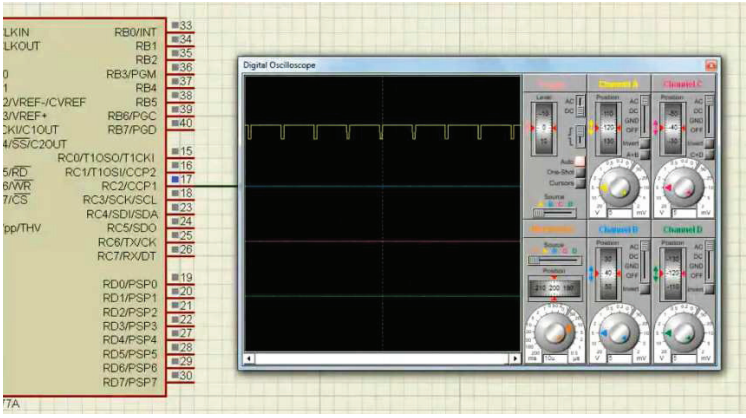


Mise en œuvre

Exemple d'application 1 : Génération d'un signal carré au niveau de RC1.

Source code 5.5 :

```
char a;

void main() {
a=10;
trisb.b0=1;
PWM1_Init(37000);
PWM1_Set_Duty(a);
for(;;)
{
PWM1_Start();
if (portb.b0==0) {delay_ms (100); a=a+10};
PWM1_Set_Duty(a);
}
}
```

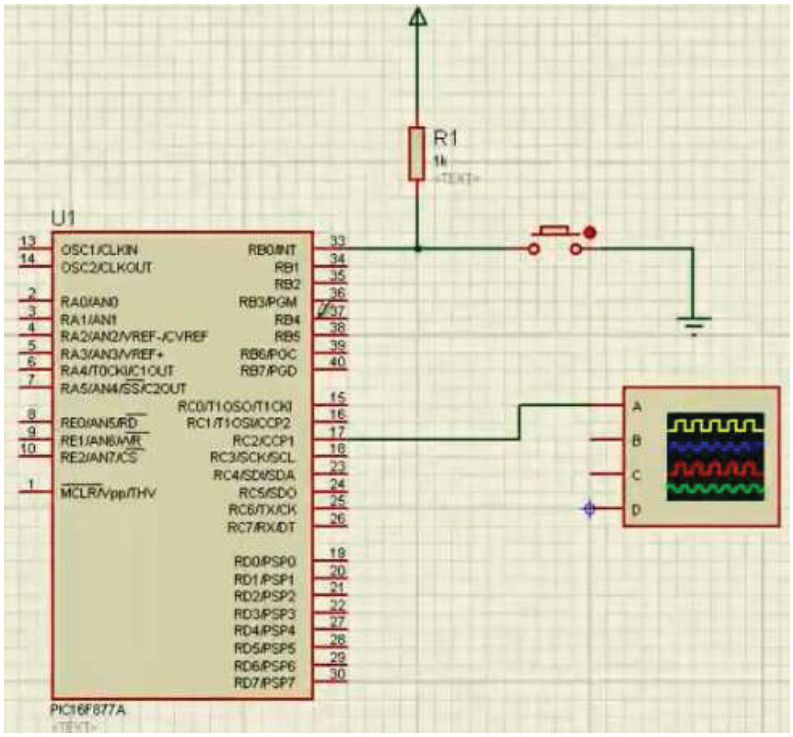


Le programme source code 5.5 peut être modifier pour générer le signal PWM au niveau de CCP1 (RC2) avec un interrupteur comme montre la figure suivante :

A chaque fois que l'interrupteur sera enfoncé le temps de l'état haut sera incrémenté par la valeur de 20.

Source code 5.6 :

```
char a;
void interrupt ()
    {
      if (Intcon.INTF)
      {
      a=a+20 ;
      }
      intcon.intf=0;
      }
void main() {
OPTION_reg.b7=0;
intcon=0b11010000;
a=40;
trisb.b0=1;
PWM1_Init(37000);
PWM1_Set_Duty(a);
for(;;)
   {
PWM1_Start();
PWM1_Set_Duty(a);
   }
}
```



Exemple d'application 2 : Emission et Réception Infrarouge.

Pour cet exemple Il faut choisir une fréquence pour l'oscillateur égale à 8MHz Plus que 5 MHz. La figure suivante représente un signal infrarouge transmis et émis.

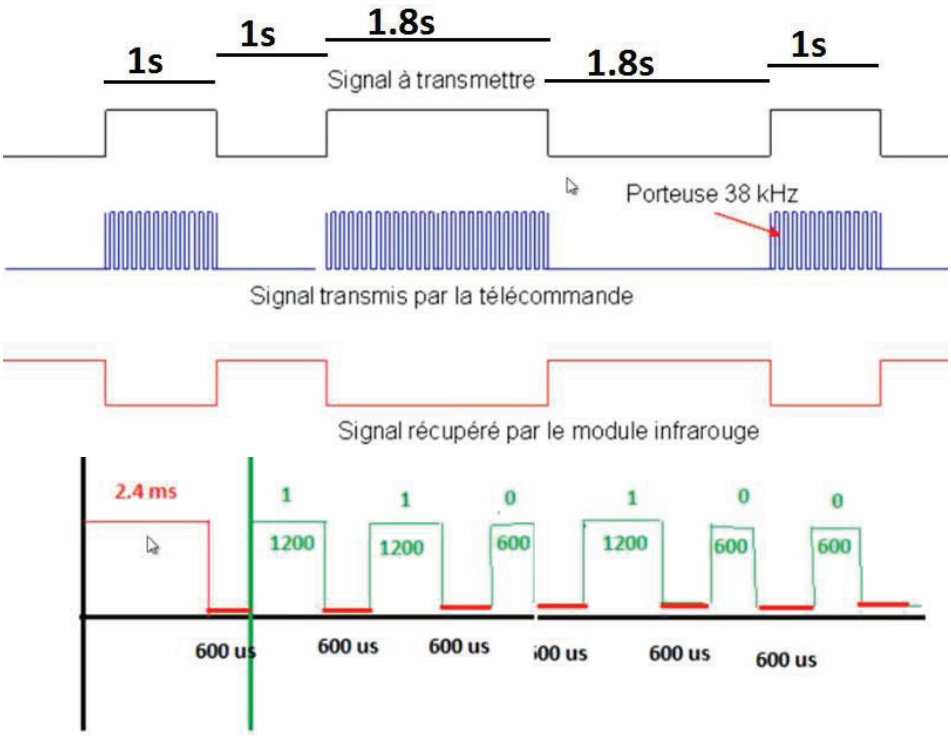
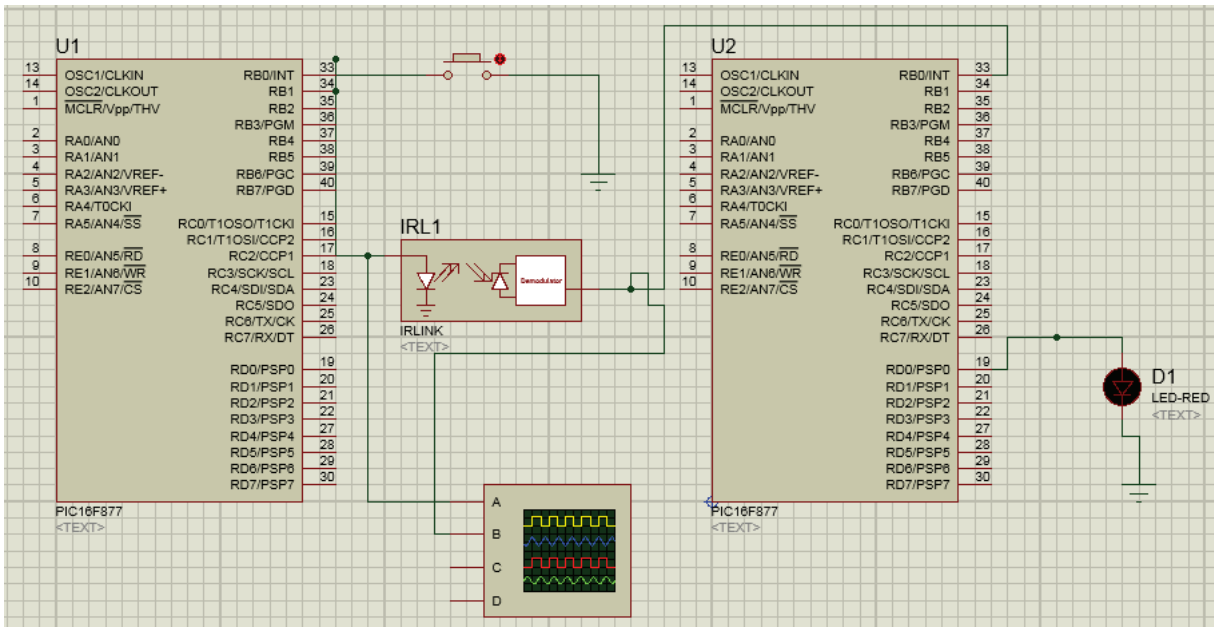


Figure 5.19 : Exemple de signaux transmis par la télécommande

Exercice 1 : Commande d'une seul LED



Source code 5.5 : Emission

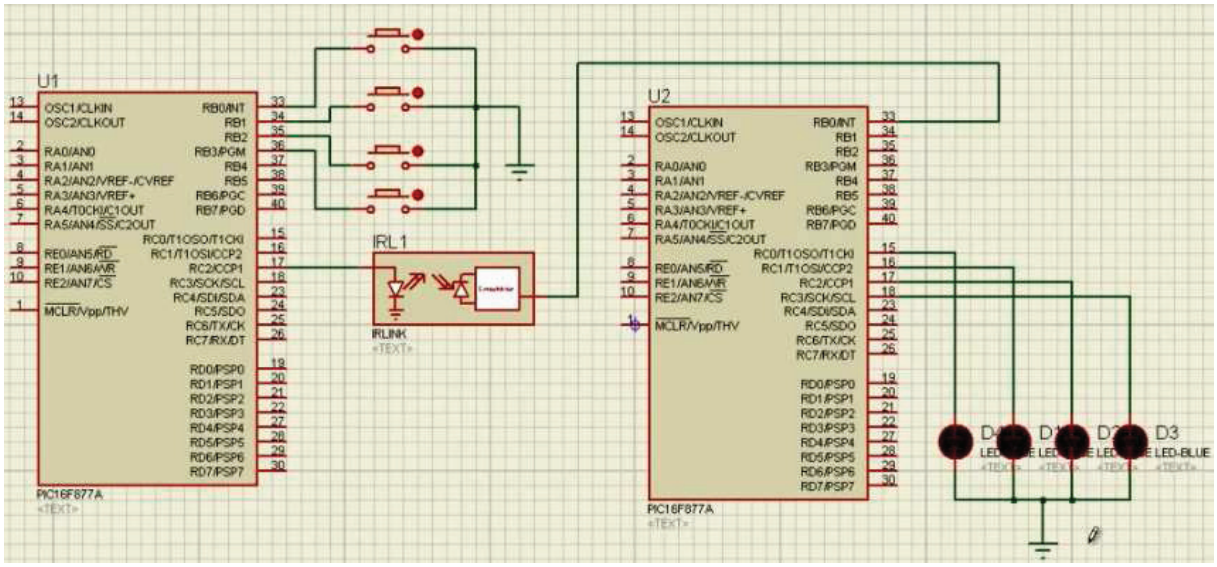
```
void main()
{
TRISB=0xff;
OPTION_REG.B7=0;
PWM1_Init(38000);
PWM1_Set_Duty(128);

for(;;)
{
 if(PORTB.B0==0)
   {
     PWM1_Start(); delay_ms(1000); PWM1_STOP();
   }
}

}
```

Source code 5.6 : Transmission

```
void main() {
      TRISB=0xff;
      OPTION_REG.B7=0;
      trisd.b0=0;
      portb.b0=0;
      for(;;){
            if (portb.b0=0)
            {
            delay_us(200);
             if (portb.b0=0)
             {
              delay_ms (900);
              if (portb.b0=0){
             portd.b0=1;
             delay_ms(1000);
             portd.b0=0;
             }
            }
      }
                  }
}
```

-Expliquer le fonctionnement de cette application

Exercice 2 : Commande de plusieurs LED



Source code 5.5 : Emission

```
void main() {
          TRISB=0xff;
      OPTION_REG.B7=0;
       PWM1_Init(38000);
 PWM1_Set_Duty(128);
        for(;;)
        {
        if(PORTB.B0==0)      //00
        {
           PWM1_Start(); delay_ms(100) ;
           PWM1_Stop(); delay_ms(300) ;
        }
         if(PORTB.B1==0)      //01
        {
           PWM1_Start(); delay_ms(200) ;
           PWM1_Stop(); delay_ms(200) ;
        }
           if(PORTB.B2==0)      //10
        {
           PWM1_Start(); delay_ms(100) ;
           PWM1_Stop(); delay_ms(100) ;
           PWM1_Start(); delay_ms(100) ;
           PWM1_Stop(); delay_ms(100) ;
        }
           if(PORTB.B0==0)       //11
        {
           PWM1_Start(); delay_ms(300) ;
           PWM1_Stop(); delay_ms(100) ;
        }
}
```

Source code 5.6 : Transmission

```
char a;
void main() {
      TRISB=0xff;
   OPTION_REG.B7=0;
   trisc.b0=0;
   portc.b0=0;
   a=0;
   for(;;)
   {
     if (portb.b0==0)
     {
     delay_us(200);   a.b7=1;
          if (portb.b0==0)
            {
             delay_ms(150); if (portb.b0==0) {a.b0=1;}else {a.b0=0;}
             delay_ms(100); if (portb.b0==0) {a.b1=1;}else {a.b1=0;}
             delay_ms(150);
             if(a==0b10000000) portc.b0=1;  else portc.b0=0;
             if(a==0b10000001) portc.b1=1;   else portc.b1=0;
             if(a==0b10000010) portc.b2=1;  else portc.b2=0;
             if(a==0b10000011) portc.b3=1;  else portc.b3=0;
             a=0;
            }
     }
   }
}
```

-Expliquer le fonctionnement de cette application

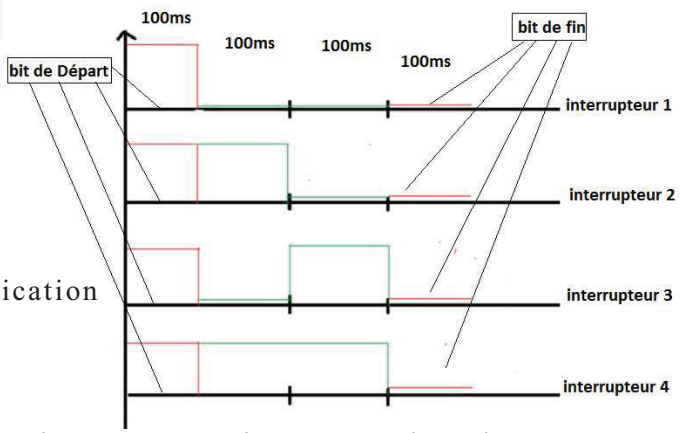
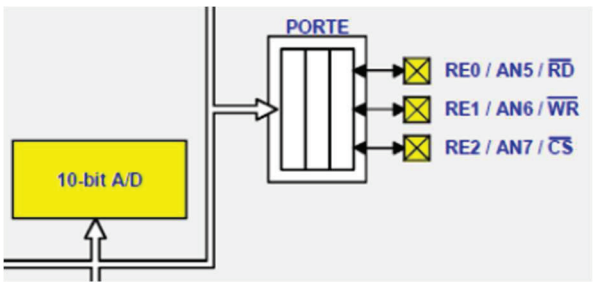
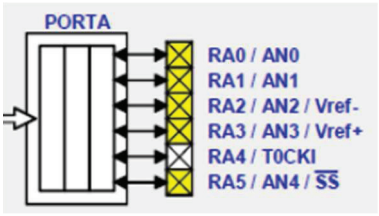


Figure 5.20 : Les signaux transmis par le PIC U1

5.8 Le module de conversion A/N [6]

Le convertisseur A/D convertit le signal analogique présent sur une de ses 8 entrées en son équivalent numérique, codé sur 10 bits. On peut donc numériser jusqu'à 8 signaux analogiques. Pas tous en même temps, bien sûr, étant donné qu'il n'y a qu'un seul module de conversion pour 8 signaux d'entrée multiplexés. Mais si vos signaux n'évoluent pas trop vite (fréquence basse), vous pouvez numériser le signal sur la patte AN0, puis celui sur AN1…





Les paramètres importants dont il faudra tenir compte sont :

• La résolution du convertisseur. Ici 10 bits, donc meilleur qu'un convertisseur 8 bits, mais moins précis qu'un 12 bits…

• Le temps de conversion.

• La rapidité d'évolution des signaux présents sur les entrées (leur fréquence pour des signaux périodiques).

• Le nombre de signaux à numériser.

En effet, pour un signal périodique, la fréquence d'échantillonnage doit être au moins deux fois supérieure à la fréquence du signal.

Les entrées analogiques doivent être configurées en entrée à l'aide des registres TRISA et/ou TRISE. **L'échantillonneur bloqueur** est intégré, il est constitué d'un interrupteur d'échantillonnage et d'une capacité de blocage de 120 pF. Les tensions de références permettant de fixer la dynamique du convertisseur. Elles peuvent être choisies parmi Vdd, Vss, Vref+ ou Vref- (Les pattes AN2 et AN3).

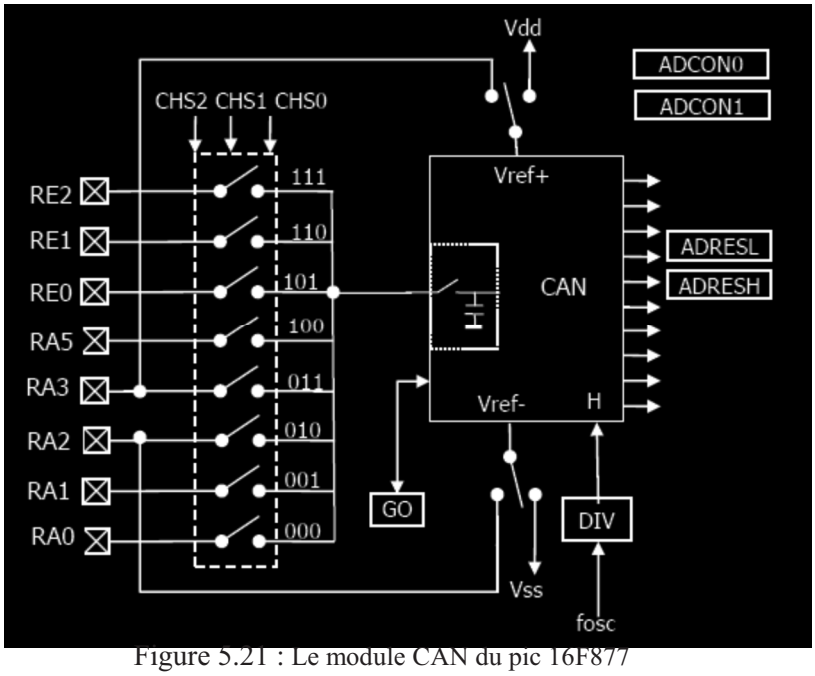


Figure 5.21 : Le module CAN du pic 16F877

Il y a 4 registres associés au module CAN du 16F877A :

►ADRESH (1EH Banc 1). le registre de résultat “Haut”.
►ADRESL (9EH Banc 2): le registre de résultat “Bas”.
►ADCON0 : premier registre de contrôle du module CAN.
►ADCON1 : seconde registre de contrôle du module CAN.

Les registres ADRESL et ADRESH :

Le convertisseur du PIC16F877A donne un résultat sur 10 bits sauvegarde dans 2 registres. ADRESL et ADRESH. Ces deux registres ensemble contiennent 16 bits, et que nous n’en utilisons que 10, « Microchip » a laissé le choix sur la façon dont est sauvegardé le résultat. On peut soit justifier le résultat à gauche, soit à droite. Il est possible de définir la justification du résultat au sein de ces 2 octets à l’aide d’un bit **ADFM** du registre **ADCON1** suivant le tableau ci-dessous :

	ADRESH								**ADRESL**							
ADFM	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	X_{10}	X_9	X	X	X	X	X	X	X	X_0
0	X_0	X	X	X	X	X	X	X	X_9	X_{10}	0	0	0	0	0	0

La justification à droite sera principalement utilisée lorsque on a besoin de l’intégralité des 10 bits de résultat tandis que la justification à gauche est très pratique lorsque 8 bits suffisent. Dans ce cas, les 2 bits de poids faibles se trouvent isolés dans ADRESL, il suffit donc de ne pas en tenir compte.

Le registre ADCON1

Ce registre permet de déterminer le rôle de chacune des pins AN0 à AN4. Il permet donc de choisir si un pin sera utilisé comme entrée analogique, comme entrée/sortie standard (numérique), ou comme tension de référence. Il permet également de décider de la justification du résultat.

ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0

PCFG3 : PCFG0 :

Configuration des E/S et des tensions de références.
Les 5 broches de PORTA et les 3 de PORTE peuvent être configurés soit en E/S digitales, soit en entrées analogiques. RA2 et RA3 peuvent aussi être configurées en entrée de référence.

PCFG3:PCFG0	RE2	RE1	RE0	RA5	RA3	RA2	RA1	RA0	VREF+	Vref-	A/R/N
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0/0
0001	A	A	A	A	Vref+	A	A	A	RA3	VSS	7/1/0
0010	N	N	N	A	A	A	A	A	VDD	VSS	5/0/3
0011	N	N	N	A	VREF+	A	A	A	RA3	VSS	4/1/3
0100	N	N	N	N	A	N	A	A	VDD	VSS	3/0/5
0101	N	N	N	N	VREF+	N	A	A	RA3	VSS	2/1/5
011x	N	N	N	N	N	N	N	N	VDD	VSS	0/0/8
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2/0
1001	N	N	A	A	A	A	A	A	VDD	VSS	6/0/2
1010	N	N	A	A	VREF+	A	A	A	RA3	VSS	5/1/2
1011	N	N	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2/2
1100	N	N	N	A	VREF+	VREF-	A	A	RA3	RA2	3/2/3
1101	N	N	N	N	VREF+	VREF-	A	A	RA3	RA2	2/2/4
1110	N	N	N	N	N	N	N	A	VDD	VSS	1/0/7
1111	N	N	N	N	VREF+	VREF-	N	A	RA3	RA2	1/2/5

Tableau 5.1 : Tableau de configuration des E/S et des tensions de référence du pic 16F877. A : le nombre des entrées analogiques ; D : le nombre des entrées/sorties numériques, R : le nombre d'application des tensions de référence.

Pour les utiliser en E/S numériques, il faut écrire '00000110' dans le registre ADCON1.

Le registre ADCON0

Ce registre est le dernier utilisé par le convertisseur analogique/numérique. Il contient les bits à manipuler lors de la conversion.

ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON

Bit 7-6: **ADCS1:ADCS0**: (A/D Conversion Clock Select bits).
00 = FOSC/2, pour fréquence maximale 1,250 MHz.
01 = FOSC/8, pour fréquence maximale 5,000 MHz.
10 = FOSC/32, pour fréquence maximale 20,000 MHz.
11 = Osc RC, Si > 1MHz, uniquement en mode « sleep ».
(**Sleep** : mise en sommeil de l'horloge principale).

ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 6 **ADCS2**: A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Tableau 5.2 : Tableau de configuration de l'horloge de conversion du pic 16F877

Afin de déterminer l'horloge du convertisseur en fonction de la fréquence du quartz utilisé (Fosc) les bits **ADCSx** peuvent être utilisés pour choisir le diviseur.

Bit 5-3: **CHS2:CHS0**: (AnalogChannel Select bits).
000 = canal analogique 0, (RA0/AN0).
001 = canal analogique 1, (RA1/AN1).
010 = canal analogique 2, (RA2/AN2).
011 = canal analogique 3, (RA3/AN3).
100 = canal analogique 4, (RA5/AN4).

A partir de ces Bits on peut sélectionner les canaux utilisés comme des entrées analogiques.

Bit 2:$GO/\ D\ O\ N\ E$ **:** (A/D Conversion Status bit, DONE: fait la conversion).
1 = Conversion A/N en marche (ce Bit fait commencer la conversion A/N)
0 = Conversion A/N non en marche (ce Bit automatiquement délibéré par le matériel quand la conversion A/N est complète).

Remarque : Ce Bit est très important pour le démarrage de conversion A/N.

Bit 1 : Inutilisés, lu comme « 0 ».
Bit 0 : **ADON :** (A/D On bit).
1 = Le module convertisseur A/N est active.
0 = Le module convertisseur A/N non active.
L'interruption <<fin de conversion>> associée au module CAN

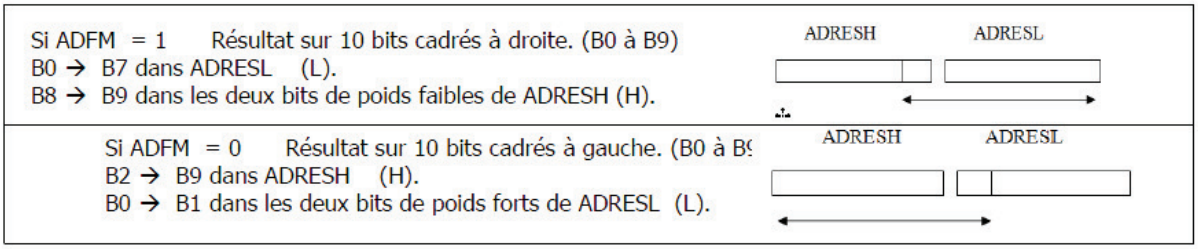
Le Registre PIR1
Le bit le plus utilisé pour la conversion est le bit 6 ADIF (Analog to Digital interrupt Flag), ce bit change son état pour indiquer la fin de conversion.

Bit 7							Bit 0
SPIF	**ADIF**	**RCIF**	**TXIF**	**SSPIF**	**CCP1IF**	**TMR2IF**	**TMR1IF**

Mise en œuvre

Comprendre le cadrage : ADFM

Le cadrage Permet facilement de lire un résultat de 8 bits sur un convertisseur de 10 bits.



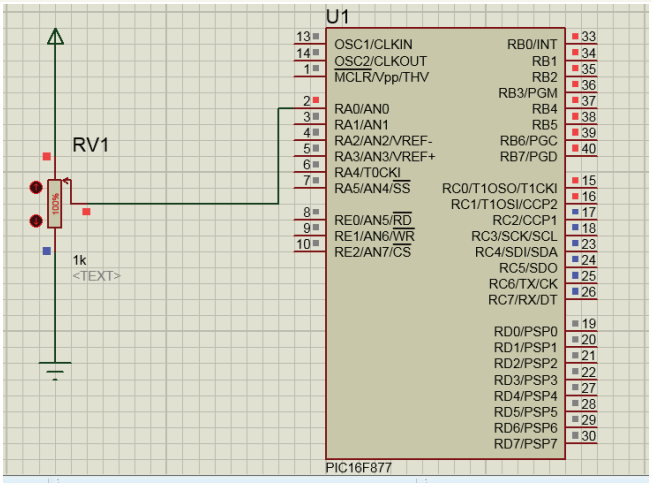
►Le dernier cas permet de récupérer le résultat de la conversion sur 8 bits (poids forts de la conversion) en lisant uniquement le registre ADRESH.
►L'initialisation du registre ADCON1 est faite une seule fois en début de programme

Exemple :

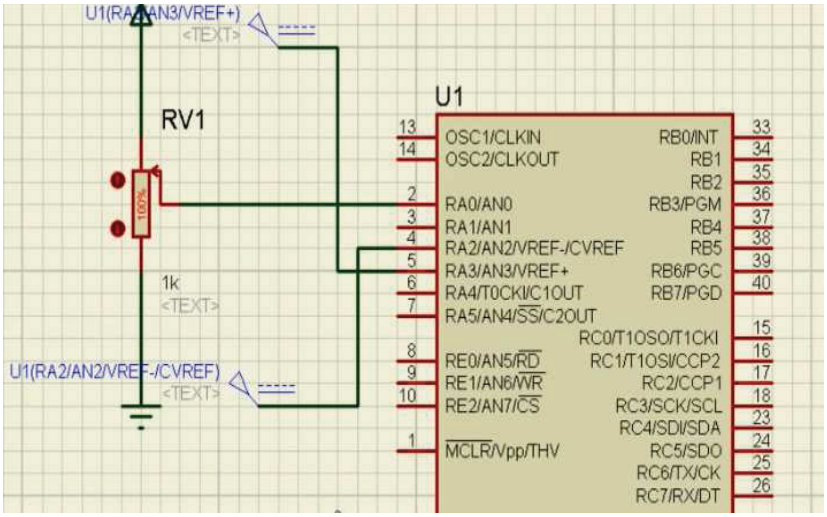
```
PORTB = a ;        // Envoyer les 8 bits inférieurs à PortB
PORTC = a >> 8; // Décalage à droite de 8 bits
PORTB = a >> 2; // Décalage à droite de 2 bits
```

Exemple d'application 1 : Conversion Analogique numérique d'une Tension.
Source code 5.7 :

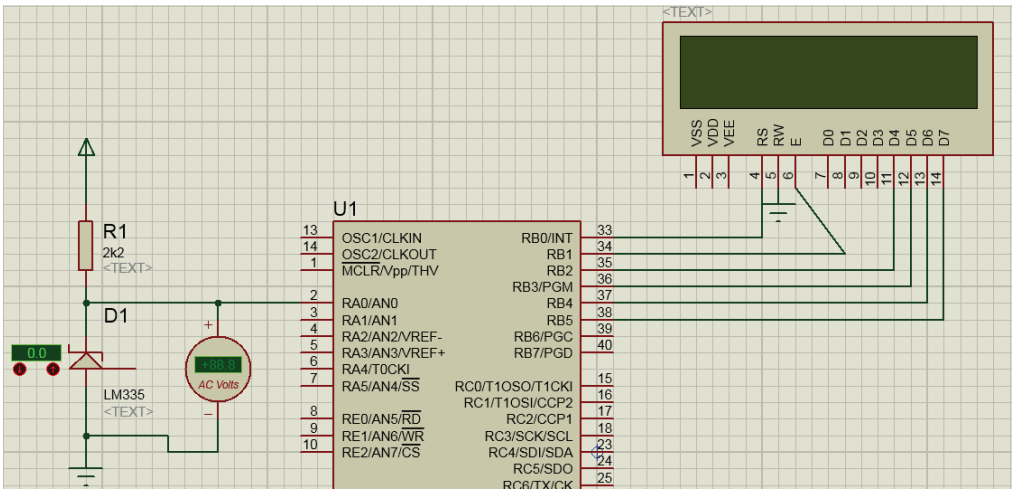
```
unsigned int a;  // déclarer une variable à 16 bits
void main() {
trisb=0x00;portb=0x00;
trisc=0x00;portb=0x00;
a=0;
ADC_Init();  // Initialiser module de ADC avec les paramètres par défaut (tous les RA sont des ADC)
for(;;)
   {
   a= ADC_Read(0);    // Obtenez des résultats 10 bits de conversion AD (0 entrée RA0)
   PORTB = a;          // Envoyer les 8 bits inférieurs à PortB
   PORTC = a >> 8;     // Envoyer les 2 bits les plus significatifs à RC1, RC0
   }
}
```



Exercice :
Faite la conversion analogique numérique de la tension
A l'entrée RA0 en respectons les tensions de référence positif et négatif.

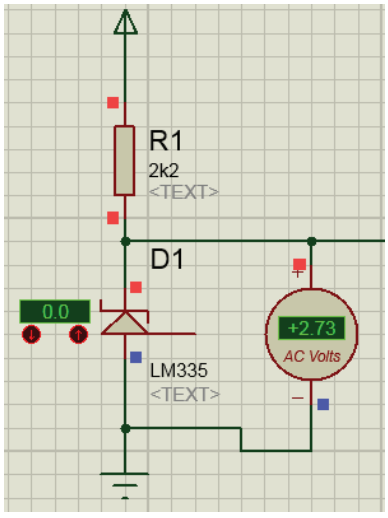


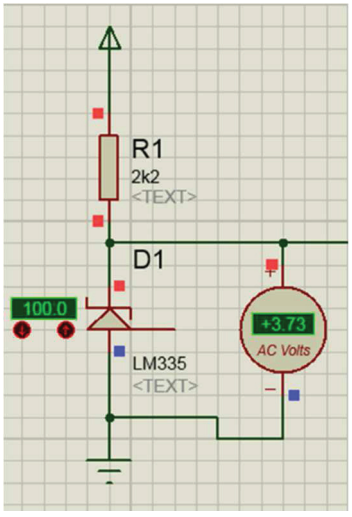
Exemple d'application 2 : Thermomètre numérique



Le thermomètre numérique au niveau de cette application fonctionne avec un capteur de température LM335. Pour avoir des informations sur son fonctionnement son branchement avec le PIC est nécessaire sans LCD pour voir les valeurs traduis par la tension un décalage et aussi nécessaire pour l'affichage sur les 8 bits (portc=a ; portd=a >> 8; on aura donc : 0.01 v ► 1 °C

Le Code binaire sur 10bits pour une Température de :
100°C ►1011111011 (code binaire) ►763(code décimale)
0 °C ► 1000101111(code binaire) ►559(code décimale)



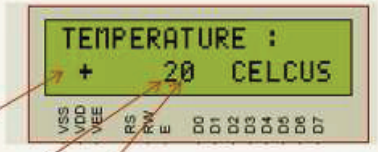


Apres l'écriture du programme qui établie la liaison entre le LCD et le PIC ; Le programme suivant affichera le changement de température sur le LCD.

Source code 5.7 :

```
char af1,af2;
unsigned int a,de;

void main() {
//trisc=0x00;portc=0x00;
//trisd=0x00;portd=0x00;
LCD_Init();
ADC_Init();
Lcd_Cmd(_LCD_CURSOR_OFF);
Lcd_Out(1,1,"TEMPERATURE :"); Lcd_Out(2,11,"CELCUS");
     for(;;)
   {
   a= ADC_Read(0);
   //portc=a;
   //portd=a >> 8;
       if (a>=559) {Lcd_Out(2,2,"+");}  else  {Lcd_Out(2,2,"-");}
        de=(a-559)/2;
        //if (de>99)  de=99;
        af1=de/10; Lcd_chr(2,7,af1+48);
        af2=de-(af1*10) ; Lcd_chr(2,8,af2+48);
   }
```

100°C → 1011111011 → 763
0°C → 1000101111 → 559

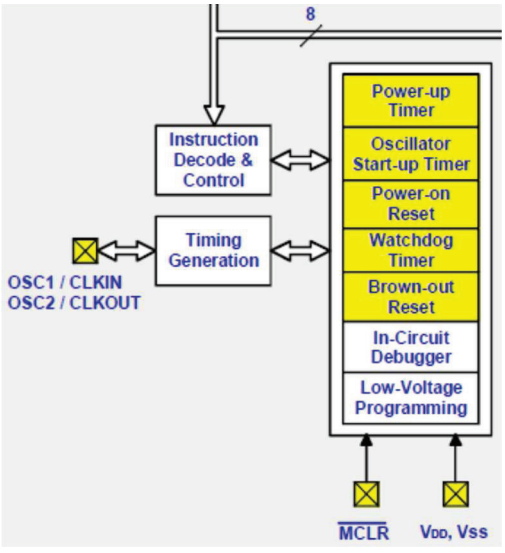


5.9 La logique de RESET [10]

La logique de reset comprend plusieurs modules :

►Le POWER-UP TIMER :

C'est une temporisation au démarrage. Lors d'un démarrage ou redémarrage et pour donner suite à un reset, cette temporisation permet à la tension d'alimentation VDD d'atteindre une valeur suffisante et stable. On évite ainsi au PIC de subir les inévitables phénomènes transitoires à la mise sous tension. Cette temporisation fonctionne avec son propre circuit RC.



► L'OSCILLATOR START UP TIMER :

Une fois la temporisation du Power-Up Timer écoulée, c'est au tour de l'Oscillator Start Up Timer de prendre le relais. Il s'agit d'une temporisation dont le but est de permettre au circuit de l'oscillateur de démarrer proprement. En effet, de la même manière que la tension d'alimentation met un certain temps à se stabiliser, l'oscillateur a besoin de temps pour démarrer et se stabiliser

► Le POWER-ON RESET :

À la mise sous tension, lorsque VDD est détecté, ce module génère une impulsion de reset, ce qui lance le Power-Up Timer et remet un certain nombre de registre du PIC dans un état déterminé. Habituellement on utilise un circuit externe à résistance / condensateur pour générer ce « pulse » de reset. Mais sur le PIC, à condition que l'apparition de la tension d'alimentation soit assez « énergique » vous pouvez vous passer de ce circuit RC. On ramène juste la tension d'alimentation à travers une résistance appropriée sur la patte « MCLR » (Memory CLeaR).

► Le BROWN-OUT RESET :

Ce module surveille la tension d'alimentation. Si celle-ci descend en-dessous d'une valeur minimum pour laquelle le fonctionnement correct du PIC ne peut plus être assuré, le module Brown-Out Reset redémarre (reset) le PIC.

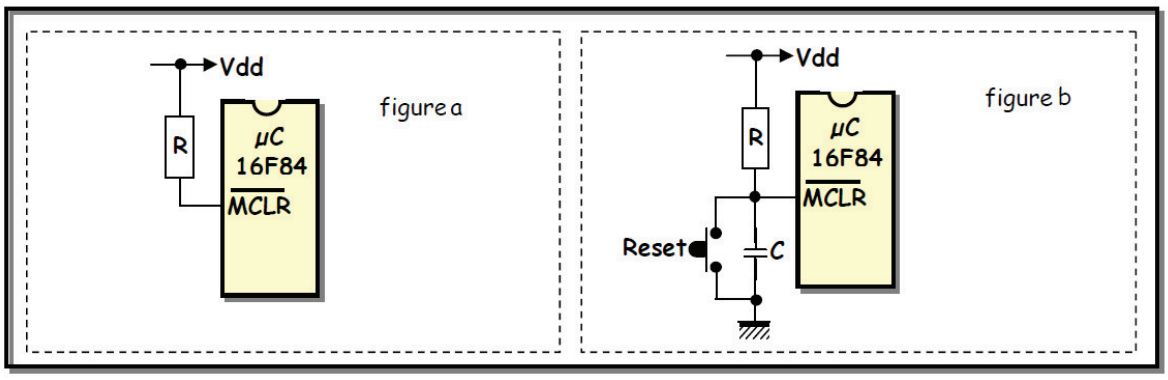
► Le WATCHDOG TIMER :

C'est le « Chien de garde » (watchdog en anglais) du PIC. S'il est utilisé, ce timer doit être périodiquement remis à zéro par le programme, car, s'il « déborde », il place le PIC en mode Reset. Ainsi, en cas de plantage de votre programme le PIC est redémarré automatiquement.

Remarque : Au Reset du µC, les ports PORTA et PORTB sont configurés en entrée.

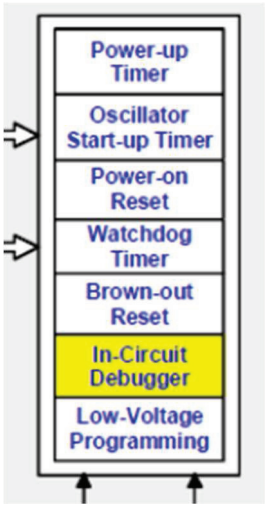
Le **Reset** du µC 16F84 peut avoir plusieurs causes :

☑ Une mise sous tension **POR** (Power On Reset) : voir figure a
☑ Une mise à 0 de la broche **MCLR** (Reset manuelle) : voir figure b.
☑ Un débordement du timer du chien de garde **WDT** : plus tard.



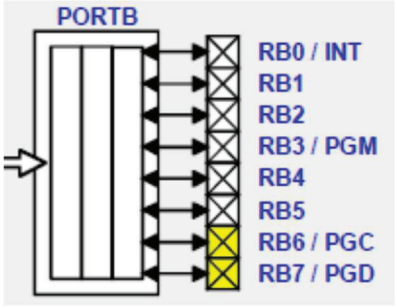
5.10 L'In-Circuit Debugger

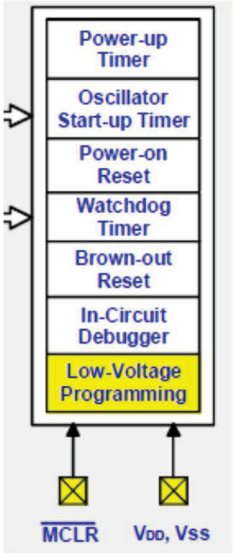
Ce module permet de simplifier la phase de débogage des programmes. Une fois activé, il donne accès à des fonctionnalités avancées de débogage. Celles-ci sont exploitées par le module ICD de MPLAB, l'environnement de développement intégré de Microchip. Vous pouvez alors suivre « en live » l'évolution de votre programme en train de s'exécuter sur le PIC, poser des points d'arrêt, lire le contenu des registres, etc.



5.11 Low-Voltage Programming et ICSP (In-Circuit Serial Programming)

Tout d'abord, voyons comment le PIC est programmé habituellement. Contrairement à des modèles de microcontrôleurs plus anciens qui étaient programmés en « parallèle » (les octets étaient transférés dans la mémoire programme sur 8 lignes), les modèles récents de PIC se programment en série. C'est-à-dire que les octets sont transmis bits après bits sur un nombre restreint de lignes. Le grand avantage c'est qu'on peut, moyennant certaines précautions, programmer le PIC alors qu'il est déjà implanté sur le circuit imprimé de son application ! Pas besoin de le retirer de son support à chaque fois (en risquant de tordre des pattes) pour le placer sur le programmateur. C'est ce qu'on appelle l'ICSP : In-Circuit Serial Programming, ou autrement dit, en français, Programmation Série En Circuit. On programme le PIC déjà implanté, en mode de transfert série.





Comment se passe la programmation ICSP ?

- On alimente le PIC avec sa tension de programmation, **généralement 13V**
- On utilise pour transférer le programme les lignes RB6 (qui devient l'horloge cadençant le transfert) et RB7 (pour transférer les données en écriture ou lecture).

On a donc besoin d'une **tension de programmation**, **plus élevée** que la tension **d'alimentation normale**.

C'est pour le cas standard, mais grâce au module « Low Voltage Programming », il y a moyen de faire la même chose en ayant pas besoin de la **tension de programmation**.

Il y a également un autre moyen de programmer notre PIC : l'utilisation d'un **Boot loader**. Qu'est-ce donc ? Un **boot loader** (traduction littérale : « chausse-pied ») est un petit bout de programme que l'on implante dans la mémoire du PIC et qui va « charger » le programme que vous voulez implanter dans la mémoire du PIC. Ici, plus besoin de programmateur. Si votre PIC est équipé d'une interface série, ou USB, ou autre, vous pouvez transférer le programme via cette interface. C'est le **boot loader**, en coordination avec le logiciel **ad' hoc** sur votre ordinateur, qui va se charger du transfert et de l'écriture en mémoire programme. Le **boot loader** reste ensuite inactif dans la mémoire du PIC pendant que votre programme s'exécute normalement. Au prochain démarrage du PIC, si le **boot loader** détecte une nouvelle tentative de transfert de programme sur l'interface sélectionnée (série, USB,…) il le charge et remplace l'actuel. Sinon il passe la main au programme actuel.

Ce mécanisme est surtout très pratique dans les phases de prototypage où de nombreux transferts son nécessaires, d'autant plus que la vitesse de transmission des données est beaucoup plus rapide qu'avec un programmateur. Ici encore, la programmation se fait sur un mode série, mais comme on ne passe pas par un programmateur on a pas accès à la tension de programmation. L'utilisation du module « Low Voltage Programming » prends ici tout son sens.

5.12 Conclusion

L'utilisation d'un microcontrôleur dans un système le rond très puissant. Ce dernier a été détaillé pour bien comprendre ses fonctionnalités. Le chapitre suivant sera consacré aux jeux d'instruction.

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

Chapitre 6 : Le jeu d'instructions du PIC

6.1 Introduction

Les PIC 16F84A, 16F628A, 16F88, 16F876A, 16F886 (famille mid-range) ont le même jeu d'instructions, constitué de seulement 35 instructions (architecture RISC : Reduced Instruction-Set Computer). Une instruction est codée par un mot de 14 bits.

La mémoire programme (de type Flash) a une taille de :

1792 octets (16F84A)
3584 octets (16F628A)
7168 octets (16F88)
14 336 octets (16F876A - 16F886)

Ce qui permet de stocker un programme de :

1024 instructions (16F84A)
2048 instructions (16F628A)
4096 instructions (16F88)
8192 instructions (16F876A - 16F886)

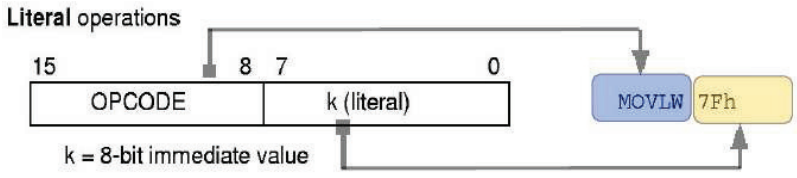
Une instruction nécessite 1 cycle, ou bien 2 cycles dans le cas d'une instruction de branchement (GOTO, CALL ...). Avec une horloge à quartz de 20 MHz, un cycle correspond à 4/(20.106) = 200 nanosecondes. Le microcontrôleur peut donc exécuter jusqu'à 5 millions d'instructions par seconde (5 MIPS) [9].

6.2 Format général

Une instruction est composée au minimum de deux parties :

Instruction = OPCODE + opérande(s)

OPCODE (Operation CODE) : partie d'une instruction qui précise quelle opération doit être réalisée.



Extrait du datasheet (documentation technique) du PIC18F4520.

Il existe trois étapes pour l'exécution d'une instruction :

✓ Lecture de l'instruction (1)
✓ Décodage de l'instruction (2)
✓ Exécution de l'instruction (3)

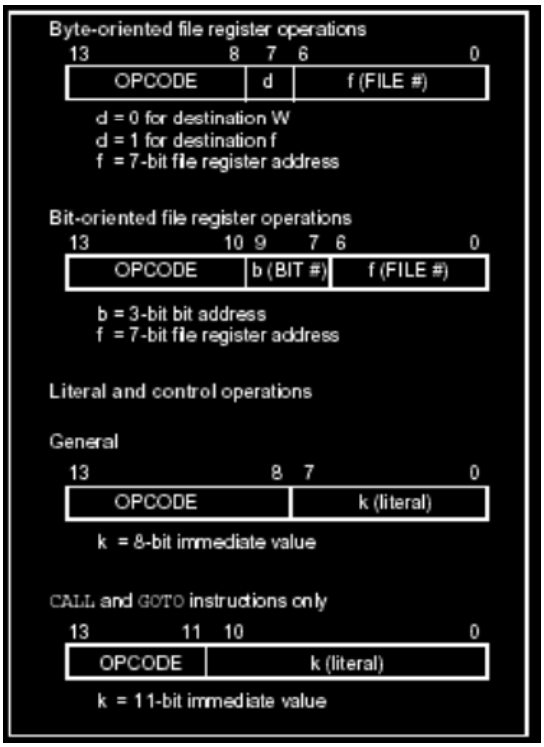


Figure 6.1 : Format général d'une instruction.

Toutes les instructions sont codées sur 14 bits. Elles sont regroupées en trois grands types, figure 6.1 :
- Instructions orientées octets
- Instructions orientées bits
- Instructions de contrôle

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

6.3 Exemple d'instruction – le transfert

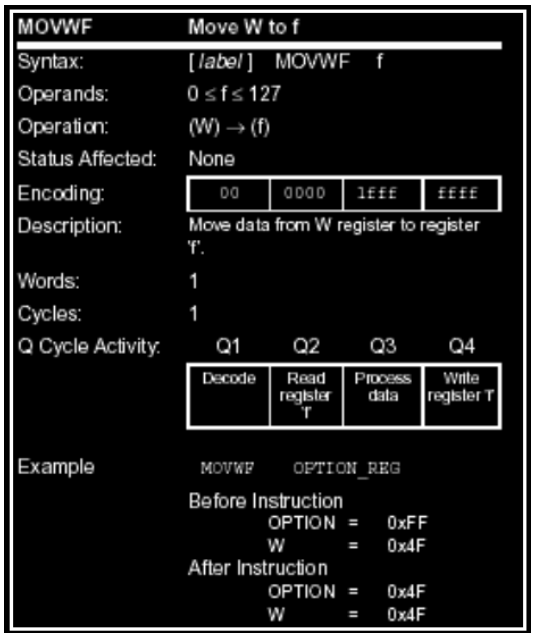
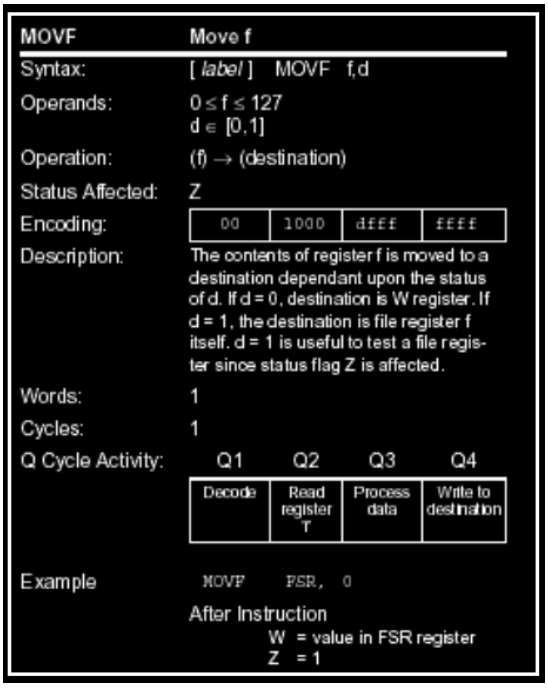


Figure 6.2 : Transfert du registre W dans le registre f

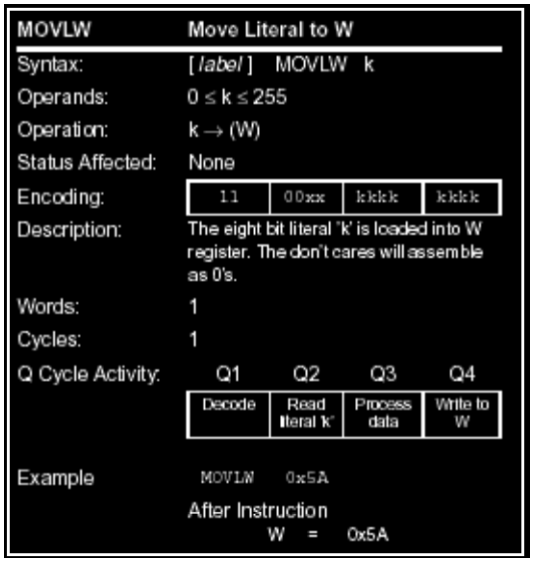
Trois instructions de transfert sont disponibles sur le PIC 16F84 par exemple. La première (Figure 6.2) permet de transférer le contenu du registre W dans un registre f. On peut noter la valeur du bit 7 à 1 et les bits 0 à 6 donnant le registre concerné.



MOVF	Move f			
Syntax:	[*label*] MOVF f,d			
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]			
Operation:	(f) → (destination)			
Status Affected:	Z			
Encoding:	00	1000	dfff	ffff
Description:	The contents of register f is moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register since status flag Z is affected.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register f	Process data	Write to destination
Example	MOVF FSR, 0 After Instruction W = value in FSR register Z = 1			

Figure 6.3 : Transfert du contenu du registre f dans le registre W ou le registre f

La seconde (Figure 6.3) permet de transférer une donnée contenue dans un registre f vers le registre W ou le registre f. Dans ce second cas, l'intérêt est de positionner le bit Z. On peut noter ici le bit 7 qui prend la valeur d fournie dans le code de l'instruction pour choisir la destination : W ou f.



MOVLW	Move Literal to W			
Syntax:	[*label*] MOVLW k			
Operands:	0 ≤ k ≤ 255			
Operation:	k → (W)			
Status Affected:	None			
Encoding:	11	00xx	kkkk	kkkk
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read literal 'k'	Process data	Write to W
Example	MOVLW 0x5A After Instruction W = 0x5A			

Figure 6.4 : Transfert d'une constante dans le registre W.

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

6.4 Liste des instructions

La Figure suivante donne la liste de toutes les instructions

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 6.5 : Liste des instructions.

6.5 Modes d'adressages

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

►Adressage immédiat : La donnée est contenue dans l'instruction.
►Adressage direct : La donnée est contenue dans un registre.
►Adressage indirect : L'adresse de la donnée est contenue dans un pointeur.

►Adressage immédiat

La donnée est contenue dans l'instruction.
Exemple : movlw 0xC4 ; Transfert la valeur 0xC4 dans W.

►Adressage direct

La donnée est contenue dans un registre. Ce dernier peut être par un nom (par exemple W) ou une adresse mémoire.
Exemple : movf 0x2B, 0 ; Transfert dans W la valeur contenue à l'adresse 0x2B.
L'adresse 0x2B peut correspondre à 2 registres en fonction de la banque choisie.
Le bit RP0 permet ce choix, le bit RP1 étant réservé pour les futurs systèmes à 4 banques.

►Adressage indirect

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

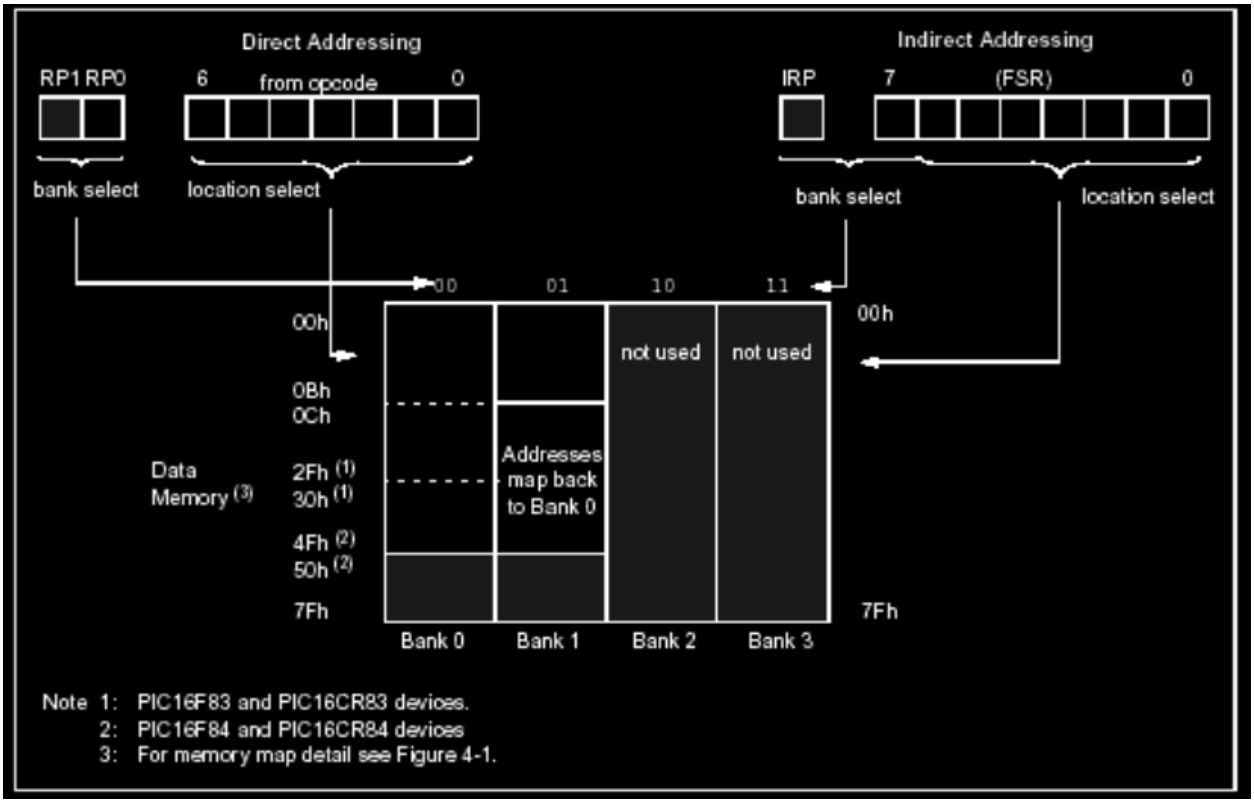


Figure 6.6 : Adressages direct et indirect à la mémoire de données.

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

Exemple :

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

6.6 Conclusion

Les programmes présentés paraissent simples, sauf que leur écriture requière une analyse détaillée du fonctionnement des circuits mis en œuvre. En effet, dans certaines configurations un 1 utilisé dans le programme permet d'activer l'interface alors que dans d'autres c'est le 0 qui le fait. Lors des différentes séances des travaux pratiques souvent les étudiants ont consommé du temps pour trouver la bonne combinaison ont opéré par tâtonnement, une approche qui s'adapte à trouver des solutions dans le cas de quelques LEDs. Cependant, avec un nombre important il est primordial d'effectuer une analyse détaillée du schématique, déclarer convenablement les données dans la mémoire, une partie du travail qui prend presque la moitié du temps de la séance, enfin écrire les instructions dans ordre correcte. Sur les dernières questions les étudiants commencent à comprendre la difficulté de la programmation et l'avantage d'une programmation évoluée.

Contenu du CD-ROM

Afin que cet ouvrage soit un véritable outil de travail, nous lui avons adjoint un CD-ROM dont le contenu vous intéressera très certainement au plus haut point. En effet, vous y trouvez les programmes et documents suivants :

Répertoire PROGRAMMATEUR :

Il contient la version 8 de Proteus qui est une suite logicielle permettant la CAO électronique éditée par la société Labcenter Electronics. Proteus est composé de deux logiciels principaux : ISIS, permettant entre autres la création de schémas et la simulation électrique, et ARES, destiné à la création de circuits imprimés. Grâce à des modules additionnels, ISIS est également capable de simuler le comportement d'un microcontrôleur (PIC, Atmel, 8051, ARM, HC11...) et son interaction avec les composants qui l'entourent.

Répertoire COMPILATEUR C :

Le compilateur "C" mono-poste pour microcontrôleurs PIC, il bénéficie d'une prise en main très intuitive et d'une ergonomie sans faille. Ses très nombreux outils intégrés (mode simulateur, terminal de communication Ethernet, terminal de communication USB, gestionnaire pour afficheurs 7 segments, analyseur statistique, correcteur d'erreur, explorateur de code, mode Débug ICD...) associé à sa capacité à pouvoir gérer la plupart des périphériques rencontrés dans l'industrie (Bus I2C™, 1Wire™, SPI™, RS485, Bus CAN™, USB, gestion de cartes compact Flash et SD™/MMC™, génération de signaux PWM, afficheurs LCD alphanumériques et graphiques et 7 à Leds segments, etc...) en font un outil de développement incontournable.

Cette nouvelle version appelée "MikroC PRO" dispose de très nombreuses améliorations : nouvelles variables utilisables, nouvelle interface IDE, amélioration des performances du linker et de l'optimisateur, cycle de compilation plus rapide, code machine généré plus compact (jusqu'à 40 % suivant les cas), nouveaux PIC supportés, environnement de développement encore plus ergonomique, nouveaux exemples d'applications, etc...

Répertoire FICHIER UTILES :

-Fichier PDF du Code ASCII : L'American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information), plus connu sous l'acronyme ASCII, est une norme informatique de codage de caractères apparue dans les années 1960. C'est la norme de codage de caractères la plus influente à ce jour. ASCII définit 128 codes à 7 bits, comprenant 95 caractères imprimables : les chiffres arabes de 0 à 9, les lettres minuscules et capitales de A à Z, et des symboles mathématiques et de ponctuation.

-Fichier PDF du Cours et Travaux Diriger

-Fichier PPT du Cours.

Répertoire Programmes Utilisés :

Il contient les programmes utilisés au niveau de ce cours en fichier.c

Travaux Dirigés

1. Introduction au langage de programmation mikroC

1.2. Structure d'un programme en mikroC

La structure la plus simple d'un programme en mikroC, c'est le programme représenté dans le code-source 3.1, qui nous permettra de faire clignoter une LED connectée au PORTB (par exemple bit 0 du PORTB) du microcontrôleur PIC avec une période de 2 secondes (1 seconde allumée et une seconde éteinte).

Code-source 1.1

```
void main()
{
for(;;)  // Boucle sans fin
{
TRISB = 0; // Configuration du PORTB  en sortie
PORTB.0 = 0; // RB0 = 0
Delay_Ms(1000);  // Pause d'une seconde
PORTB.F0 = 1;  // RB0 = 1
Delay_Ms(1000);  // Pause d'une seconde
}  // Fin de la boucle
}
```

1.3. Règles générales d'écriture en mikroC

-Les instructions propres au langage microC doivent être écrites en minuscule (**void** main (void)).

-Les instructions particulières aux microcontrôleurs doivent être écrites en majuscule (TRISB).

-Les retours à la ligne et les espaces servent uniquement à aérer le code

-Toutes instructions ou actions se terminent par un point virgule « ; ».

1.4. Commentaires

Les commentaires sont utilisés pour préciser le fonctionnement du programme et pour une annotation du programme. Les lignes de commentaires sont ignorées et non compilé par le compilateur. En mikroC les commentaires de programmes peuvent être de deux types: de longs commentaires, s'étendant sur plusieurs lignes, et de courts commentaires, occupant une seule ligne. Un commentaire au début d'un programme peut décrire brièvement le fonctionnement du programme et de fournir le nom de l'auteur, le nom du fichier programme, la date à laquelle le programme a été écrit, et une liste des numéros de version, ainsi que les modifications dans chaque version.

Comme montre le Code-source 1.1 les longs commentaires commencent par le caractère « /* » et se termine par le caractère « */ ».

De même, de courts commentaires commencent par le caractère « // » et il n'a pas besoin d'un caractère de terminaison.

1.5. Début et fin d'un programme

En langage microC, un programme commence avec les mots-clés : void main() Après cela, une accolade ouvrante est utilisée pour indiquer le début du corps de programme. Le programme se termine par une accolade fermante. Ainsi, comme indiqué dans le Code-source 2.1, le programme a la structure suivante void main() { // Votre code ici}

Le point-virgule « ; » indique la fin d'une instruction, sinon une erreur du compilateur sera générée.
j = 5; *// correcte*
j = 5 *// erreur*

TD N°1 : Arithmétique binaire et expressions en C, PIC.

Pour bien comprendre la signification des expressions, il est essentiel d'avoir 2 notions en tête : la priorité et l'associativité. Nous donnons ci-après un tableau des opérateurs par priorité décroissante :

Catégorie d'opérateurs	Opérateurs	Associativité
fonction, tableau, membre de structure, pointeur sur un membre de structure	( ) [ ] . ->	Gauche -> Droite
opérateurs unaires	- ++ -- ! ~ * & sizeof (type)	Droite ->Gauche
multiplication, division, modulo	* / %	Gauche -> Droite
addition, soustraction	+ -	Gauche -> Droite
décalage	<< >>	Gauche -> Droite
opérateurs relationnels	< <= > >=	Gauche -> Droite
opérateurs de comparaison	== !=	Gauche -> Droite
et binaire	&	Gauche -> Droite
ou exclusif binaire	^	Gauche -> Droite
ou binaire	|	Gauche -> Droite
et logique	&&	Gauche -> Droite
ou logique	||	Gauche -> Droite
opérateur conditionnel	? :	Droite -> Gauche
opérateurs d'affectation	= += -= *= /= %= &= ^= |= <<= >>=	Droite -> Gauche
opérateur virgule	,	Gauche -> Droite

Exercice 1 :

Enlever les parenthèses des expressions suivantes lorsqu'elles peuvent être retirées.

a=(25*12)+b;
if ((a>4) &&(b==18)) { }
((a>=6)&&(b<18))||(c!=18)
c=(a=(b+10));

►Évaluer ces expressions pour a=6, b=18 et c=24

Les types du C PIC sont :
unsigned char a;//8 bits, 0 to 255
signed char b; //8 bits, -128 to 127
unsigned int c; //16 bits, 0 to 65535
signed int d; //16 bits, -32768 to 32767
long e; //32 bits, -2147483648 to 2147483647
float f; //32 bits

Exercice 2 :

b7	*b6*	*b5*	*b4*	*b3*	*b2*	*b1*	*b0*

Si une variable p1 de type signed char (8 bits signés) est déclarée écrire les expressions en C permettant de :
- mettre à 1 le bit b2
- mettre à 1 le bit b3 et b6
- mettre à 0 le bit b0
- mettre à 0 le bit b4 et b5
- inverser le bit b3 (se fait facilement avec un ou exclusif)
- mettre à 1 le bit b2 et à 0 le bit b0
- mettre à 1 les bits b0 et b7 et à 0 les bits b3 et b4

Exercice 3 :

Soit une variable :

char nb;
Écrire les expressions permettant de calculer les centaines, les dizaines et les unités de cette variable.
Il existe plusieurs autres méthodes pour positionner les bits
(http://www.microchipc.com/HiTechCFAQ/index.php)
1° méthode pour positionner bit à bit :
#define bit_set(var,bitno) ((var) |= 1 << (bitno))
#define bit_clr(var,bitno) ((var) &= ~(1 << (bitno)))
unsigned char x=0b0001;
bit_set(x,3); //now x=0b1001;
bit_clr(x,0); //now x=0b1000;*/
2° méthode pour positionner plusieurs bits
#define bits_on(var,mask) var |= mask
#define bits_off(var,mask) var &= ~0 ^ mask
unsigned char x=0b1010;
bits_on(x,0b0001); //now x=0b1011
bits_off(x,0b0011); //now x=0b1000 */

TD N°2 : Expression booléenne vraie.

Dans la suite du cours on appellera expression booléenne (E. B.) une expression qui si elle était affectée à une variable donnerait soit la valeur 0 ou soit la valeur 1. Le C est un langage qui ne permet de fabriquer que des expressions (Expr). C'est très pratique, mais la confusion des deux peut conduire à des erreurs. Donnons un exemple :

expression (Expr)	*expression booléenne (E. B.)*

`char n=10;`
`while(n) {`
`.....;`
`n--; }`
	`char n=10;`
	`while(n!=0) {`
	`.....;`
	`n--; }`

Ces deux constructions sont permises en C car il y a une technique universelle qui permet de transformer une expression en expression booléenne.

- tout ce qui est évalué à 0 est considéré comme faux,
- tout ce qui est évalué différent de 0 est considéré comme vrai.

Si on se rappelle que dans une parenthèse d'un while on a une Expression Booléenne (E. B.). D'un côté n et de l'autre n!=0 sont donc des E. B. La différence entre les deux est que pour n elle peut prendre toutes les valeurs entre -128 et +127 (car n est de type char) alors que n!=0 ne prendra que deux valeurs 0 ou 1. La deuxième sera donc appelée E. B. et la première Expr qui sera transformée comme indiqué.

Le problème du langage C est que l'oubli d'un & ou d'un | conduit en général à une expression souvent en lieu et place d'une expression booléenne ce qui est tout à fait autorisé. Par exemple écrire a & b au lieu de a && b, ou pire encore un "=" au lieu d'un "==".

Exercice 1 :

Différence entre && et &
Évaluer les expressions :
a&b
a&&b
Pour a= 0xF0 et b=0x0F
En déduire les valeurs booléennes correspondantes (si ces expressions étaient utilisées dans un if par exemple).
Construire des expressions booléennes sur les tests suivants expression vraie si
le bit b6 est à 1
le bit b3 est à 0
le bit b6 est l'inverse du bit b3
le bit b2 est à 1 et le bit b4 est à 0
le bit b2 est à 1 ou le bit b7 est à 0

Les tests d'un bit particulier en C peuvent aussi être réalisés de la manière suivante

```
x=0b1000; //decimal 8 or hexadecimal 0x8
if (testbit_on(x,3)) a(); else b(); //function a() gets executed
if (testbit_on(x,0)) a(); else b(); //function b() gets executed
if (!testbit_on(x,0)) b(); //function b() gets executed
#define testbit_on(data,bitno) ((data>>bitno)&0x01)
```

Exercice 2 :

Quelle opération arithmétique est réalisée par un décalage ? Évaluer pour cela les expressions suivantes (avec a=12 et b=23) :
- a = a >> 1 (ou a >>= 1)
- a = a >> 2 (ou a >>= 2)
- b = b << 1 (ou b <<=1)
- b = b << 2 (ou b <<=2)

Généralisation.
Construire une vraie expression booléenne avec opérateur de décalage, & et ^ qui reprend le test de l'exercice précédent : le bit b6 est l'inverse du bit b3

Exercice 3 :

On donne le sous-programme suivant (tiré d'un ancien projet inter-semestre) :

```
void conversion (char nb,char result[8]){
char i;
for(i=7;i>=0;i--) if ((nb & (1<<i)) == (1<<i)) result[i]=1;
else result[i]=0; }
```

1) Peut-on retirer des parenthèses dans l'expression booléenne du if ?
2) Peut-on écrire (nb & (1<<i)) au lieu de ((nb & (1<<i)) == (1<<i)) ?
3) Construire l'expression booléenne qui permettrait l'écriture

```
for(i=7;i>=0;i--) if (E.B.?????) result[i]=0;
else result[i]=1;
```

En donnant toujours le même le même résultat de conversion.

4) Modifier le programme pour qu'il fasse une conversion d'entier (16 bits) vers le binaire.
5) Est-ce que l'algorithme suivant donne le même résultat de conversion :

```
for(i=0;i<8;i++) {
result[i]=nb%(2);
nb = nb / 2;}
```

Exercice 4 :

Soit le programme suivant :

```
#include <stdio.h>
main() {
int n=10,p=5,q=10,r;
r= n == (p = q);
printf("A : n = %d p = %d q= %d r = %d\n",n,p,q,r);
n = p = q = 5;
n += p += q;
printf("B : n = %d p = %d q= %d\n",n,p,q);
q = n++<p || p++ != 3;
printf("C : n = %d p = %d q= %d\n",n,p,q);
q = ++n == 3 && ++p == 3;
printf("D : n = %d p = %d q= %d\n",n,p,q);
return 0;}
```

Que donnera -t-il comme affichage sur l'écran ?

TD N°3 : Architecture des PIC 16FXXX.

Architecture générale :

1-Architecture mémoire : La donnée de base de PIC (16FXXX) est l'octet. L'octet comporte 8 bits. Le bit 0 est le bit de poids faible et le bit 7 est le bit de poids fort.

2-Mémoire programme : la mémoire programme du PIC (16F84) est organisée en mots de 14 bits et elle peut en contenir 1024 (soit 1k). Le bit 0 est aussi le bit de poids faible et le bit 13 est le bit de poids fort. La mémoire programme comporte donc 1k x 14 bits.

Exercice 1 :

On rappelle qu'une mémoire est caractérisée par un bus de donnée de largeur n (n bits de D0 à Dn-1) et un bus d'adresse de largeur m (m bits de A0 à Am-1).

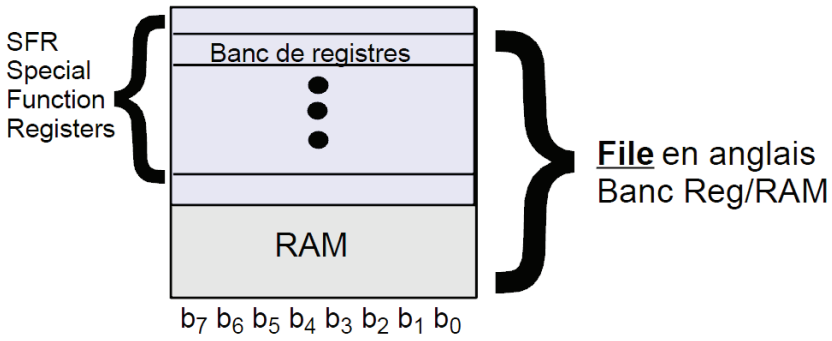
1°) Pour la mémoire programme du 16F84 donner m et n.
2°) La mémoire programme du 16F873 est 4 fois plus grande : donner les nouveaux m et n.
3°) La mémoire programme du 16F887 utilisé en TP est 8 fois plus grande qu'en 1°, donner les nouveaux m et n.

3-Mémoire EEPROM : le PIC (16F84) dispose de 64 octets de mémoire EEPROM.

Exercice 2 :

Calculer m et n pour l'EEPROM du 16F84.

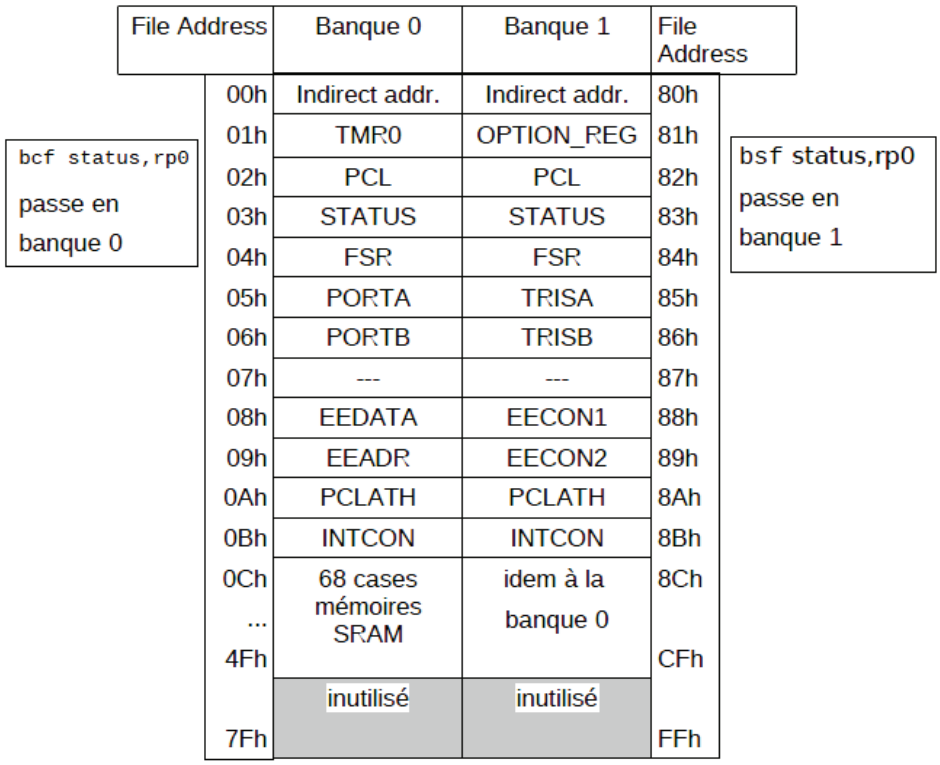
4-Mémoire donnée : Description simplifiée



Il faut ajouter une division verticale que l'on appelle banque. Le PIC (16F84) possède deux banques (Sélectionnées par le bit RP0 du registre **STATUS** subdivisées en deux parties :

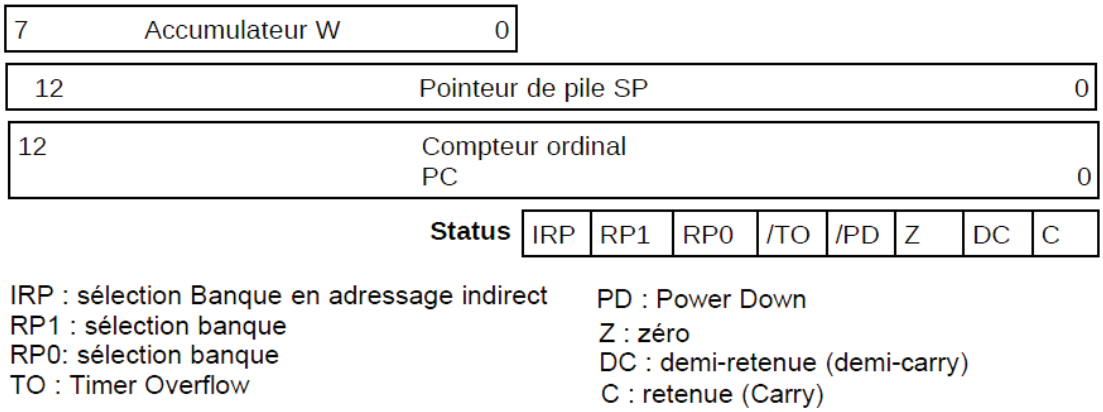
- registres de fonctions spéciales (SFR Special Function Registers)
- cases mémoires libres que vous pouvez utiliser à votre guise.

La mémoire du 16FXXX est organisée en banques sélectionnées par le(s) bit(s) RP0 (et RP1) du registre **STATUS**. Seul RP0 est utilisé pour le 16F84, soit deux banques.



5-Le modèle de programmation du PIC (16FXXX)

Il peut être présenté de la manière très simplifiée suivante :



Le pointeur de pile n'est manipulé directement par aucune instruction. Il ne comporte qu'une profondeur de 8 niveaux.

Le compteur programme est en fait composé de deux registres 8 bits :
- **PCL** (02h/82h) pour les 8 bits de poids faibles (accessible en lecture/écriture)
- **PCLATH** (0Ah/8Ah) pour les 5 bits de poids forts (accessible en lecture/écriture)

Le 16F84 gère seulement 10 bits des 13 bits du PC
Le modèle de programmation complet est plus complexe, en particulier à cause des registres.

6-Utiliser les registres du PIC en C

Le registre **STATUS** présenté ci-dessus comporte des bits qui possèdent un nom. Ce n'est pas le seul registre à avoir cette propriété. Mais peut-on en déduire que les bits des registres possèdent tous un nom que le compilateur C connait ? La réponse est non ! Et pour corser le tout il n'y a aucune règle générale. Si vous voulez vous en sortir le seul moyen est de lire les fichiers d'entête correspondant à votre PIC. En général, le nom des registres et des bits est respecté mais il existe des bits qui ont un nom dans la documentation officielle mais pas avec le MikroC.

-Exemple de fichier d'en-tête pour MikroC

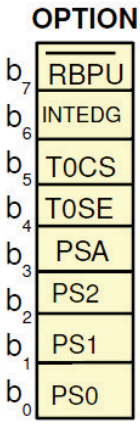
À proprement parler, pour le compilateur MikroC, les définitions ne sont pas dans un fichier avec une extension (.h) mais dans des fichiers qui se trouvent dans un répertoire defs et qui ont l'extension (.c). On en présente un extrait maintenant :

```
//***** MikroC fichier P18F84A.c ******
const unsigned short
W = 0x0000,
F = 0x0001,
IRP = 0x0007,
RP1 = 0x0006,
RP0 = 0x0005,
NOT_TO = 0x0004,
NOT_PD = 0x0003,
Z = 0x0002,
DC = 0x0001,
C = 0x0000,
....
unsigned short register volatile
STATUS absolute 0x0003;
....
unsigned short register
OPTION_REG absolute 0x0081,
....
```

Cela permet de connaître le nom des bits du registre **STATUS** et de confirmer le nom du registre **OPTION** (comme OPTION_REG). Rappelez-vous que le langage C est sensible à la casse des caractères.

Les trois possibilités pour atteindre un bit particulier

Voici à travers un exemple comment accéder à un bit particulier pour le registre **OPTION** :



Exemple en MikroC

```
void main( void) {
....
//*** Toujours *****
  OPTION_REG.F3 =1;
//**** Si on connaît le nom
  OPTION_REG.PSA =1;

...

}
```

Fichier d'entête

```
....
//********** bits de OPTION
NOT_RBPU       = 0x0007,
INTEDG       = 0x0006,
T0CS       = 0x0005,
T0SE       = 0x0004,
PSA      = 0x0003,
PS2      = 0x0002,
PS1      = 0x0001,
PS0      = 0x0000,
...
```

Exercice 3 :

Pour vérifier la bonne compréhension de ce qui vient d'être dit, nous donnons à droite le fichier d'entête des bits correspondant au registre **INTCON**.

Fichier d'entête

```
//********* bits de INTCON
GIE      = 0x0007,
EEIE       = 0x0006,
T0IE       = 0x0005,
INTE       = 0x0004,
RBIE       = 0x0003,
T0IF       = 0x0002,
INTF       = 0x0001,
RBIF       = 0x0000,
```

1°) Dessiner le registre correspondant
2°) En utilisant les masques du TD précédent, écrire les instructions C permettant :

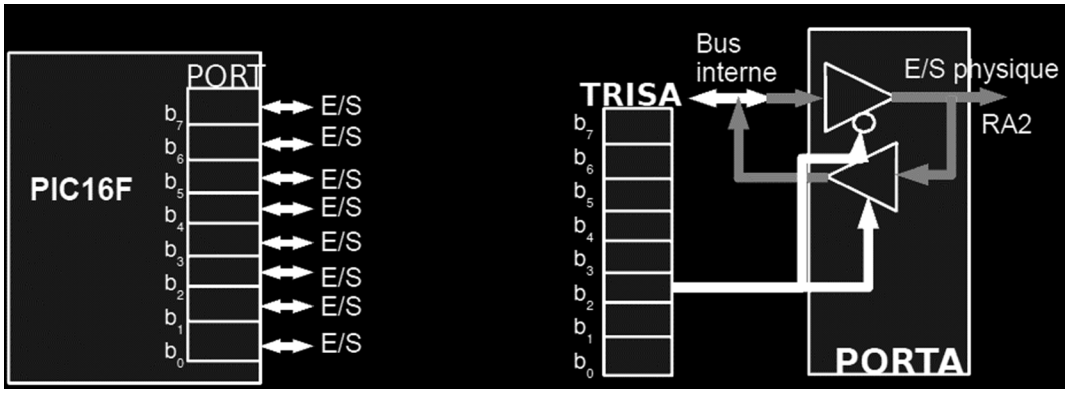
- mise à 1 de GIE
- mise à 0 de INTE.
- tester si T0IE est à 1
- tester si T0IF est à 0

3°) Refaire le même travail en utilisant les noms des bits directement.

TD N°4 : Le PORT pour entrer et sortir des informations

Le registre TRISA : Ce registre est d'un fonctionnement très simple et est lié au fonctionnement du PORTA. Chaque bit de TRISA positionné à 1 configure la broche correspondante en entrée.

Chaque bit à 0 configure la pin en sortie.

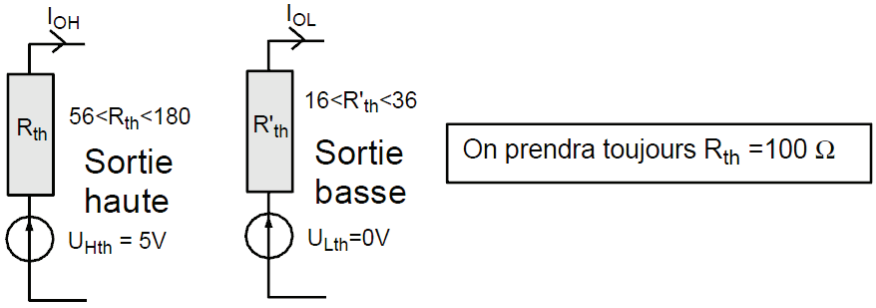


Au Reset du PIC, tous les pins sont mis en entrée, afin de ne pas envoyer des signaux non désirés sur les pins. Les bits de **TRISA** seront donc mis à 1 lors de chaque Reset.

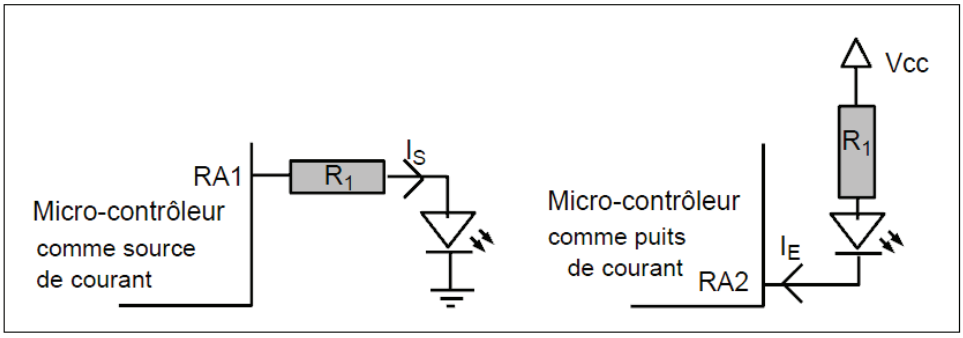
Notez également que, comme il n'y a que 5 pins utilisées sur le **PORTA**, seuls 5 bits (b0 ... b4) seront utilisés sur TRISA. Les bits de TRISA sont désignés par TRISA.F0 ... TRISA.F4

Les registres PORTB et TRISB : Ces registres fonctionnent exactement de la même manière que **PORTA** et **TRISA**, mais concernent bien entendu les 8 pins RB. Tous les bits sont donc utilisés dans ce cas. Voyons maintenant les particularités du **PORTB**. Les entrées du **PORTB** peuvent être connectées à une résistance de rappel au +5V de manière interne, cette sélection s'effectuant par le bit 7 du registre **OPTION** (effacement du bit7 RBPU pour valider les résistances de rappel au +5V). Les bits de **TRISB** sont désignés par TRISB.F0 ... TRISB.F7

Le PORT et sa modélisation électrique : Le modèle électrique est très simple : on le modélise comme d'habitude à l'aide de Thevenin. Il est raisonnable de prendre un Imax de 10 mA pour 16F84 et 15 mA pour 16F877.



Il y a deux façons typiques pour connecter des LEDs :



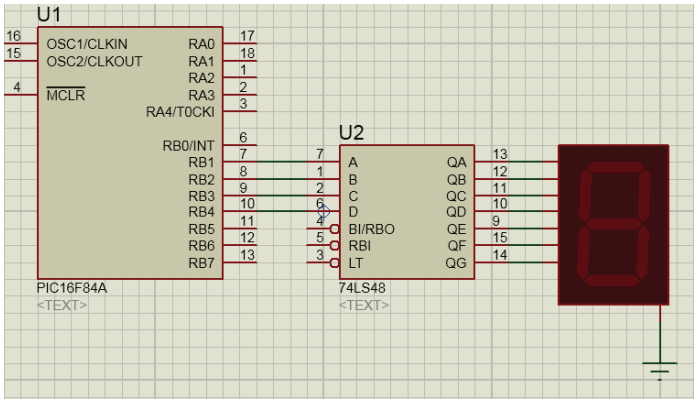
TD N°5 : Ports d'entrée sorties PIC

►Exercice 1 :

Les programmes 1 et 2 sont sur les deux PIC 16F84 . Qu'est-ce que vous avez au niveau des afficheurs ?

Program1 (Question1):

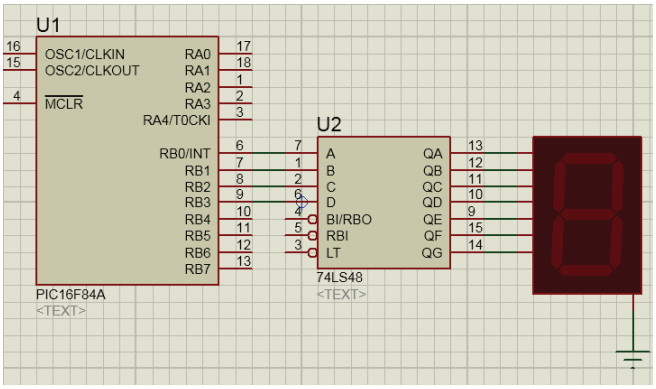
```
void main() {
TRISB=0x00;  PORTB=0x00;
 for(;;)
        {
          PORTB=0 ; delay_ms(1000);
          PORTB=1 ; delay_ms(1000);
          PORTB=2 ; delay_ms(1000);
         // PORTB=3 ; delay_ms(1000);
          PORTB=4 ; delay_ms(1000);
          //PORTB=5 ; delay_ms(1000);
          PORTB=6 ; delay_ms(1000);
          //PORTB=7 ; delay_ms(1000);
          PORTB=8 ; delay_ms(1000);
          PORTB=9 ; delay_ms(1000);    }   }
```



Réponse1	

Program2 (Question2):

```
void main() {
TRISB=0x00;
PORTB=0x00;
PORTB=7;
  for(;;)
      {
        PORTB++;
        if (PORTB==9)(PORTB=0);
        delay_ms(1000) ;       }   }
```



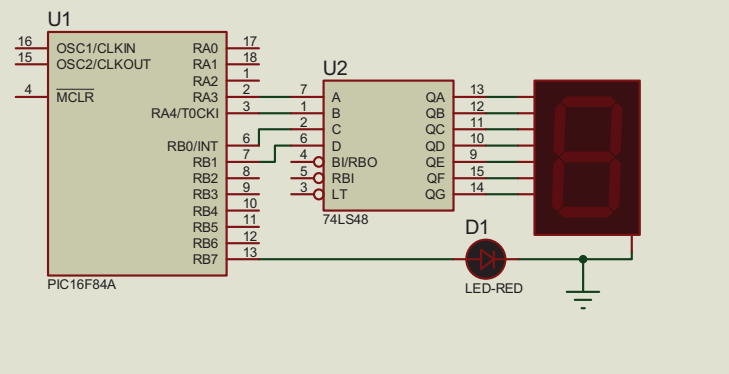
Réponse2	

►Exercice 2 :

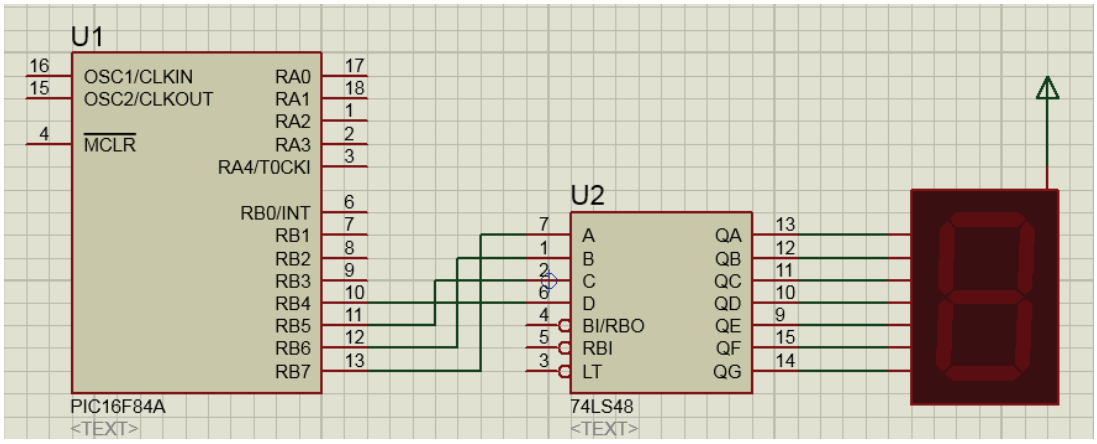
Les schémas ci-dessous représentent une application simple de l'utilisation d'un afficheur 7 segments avec un décodeur 74ls48.

1-Donner le programme qui affiche la suite des nombres de 0 à 9 (sur l'afficheur 7 Segments à cathode commune).

2-Modifier ce programme pour allumer une LED juste lors de l'affichage du chiffre 3.

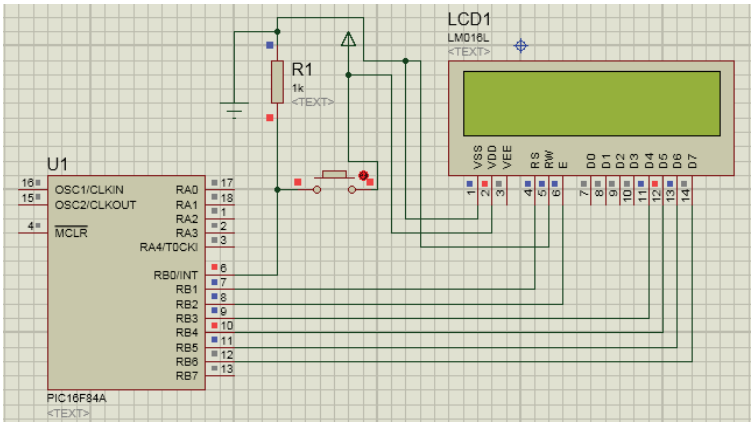


3-Donner le programme qui affiche la suite des nombres de 0 à 9 (sur l'afficheur 7 Segments à anode commune).
4-Modifier ce programme pour que le comptage commence du chiffre 5.



►**Exercice 3** :

Corriger les erreurs qui existent au niveau du programme suivant (Int ouvert = ON , Int fermer = OFF) :

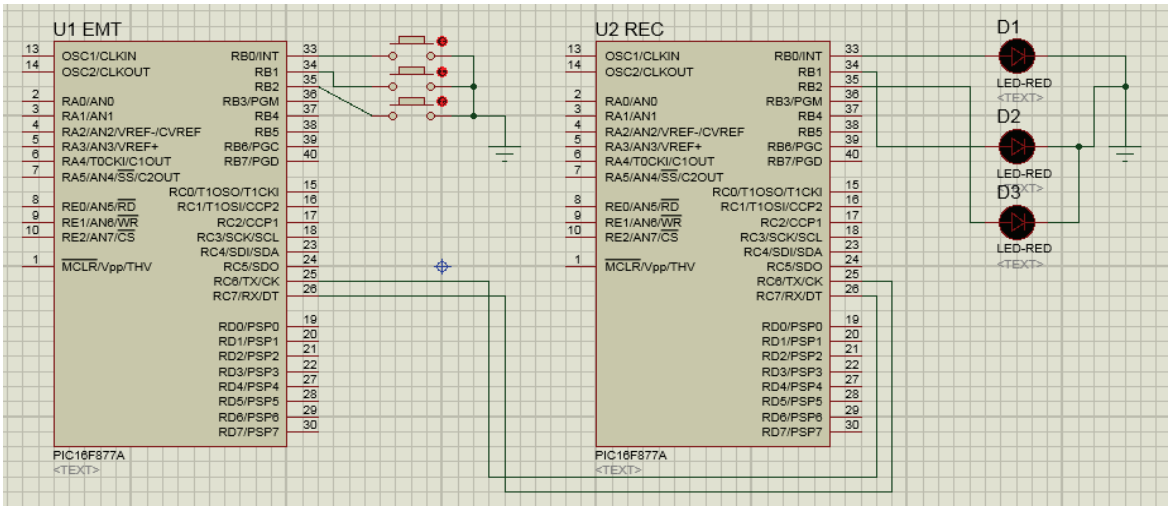



```
sbit LCD_RS at RB1_bit;
sbit LCD_EN at RB2_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D3 at RB3_bit;
sbit LCD_RS_Direction at TRISB1_bit;
sbit LCD_EN_Direction at TRISB2_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D3_Direction at TRISB3_bit;
char i;
void main {
 TRISA=0xff;
 LCD_Init(); Lcd_Cmd(_LCD_CURSOR_OFF);
 for(i=0;i<4;i++);
                    {
        while (PORTB.b0==1) ; lcd_out(1,1,"ON");
        while (PORTB.b0==0) ; lcd_out(1,1,"Off") ;
         Lcd_Cmd(_LCD_CLEAR) }}
```

TD N°6 : Principe de fonctionnement du PIC

►Exercice 1 :

Une communication série est réalisée entre deux PIC16F877.
1-Expliquer le fonctionnement de cette communication.
2- Cette communication est-elle synchrone ou asynchrone ? EXPLIQUER.
Le programme suivant est utilisé au niveau de l'un des deux PIC :




```
void main() {
   TRISB=0x00;
UART1_init(8929) ;
for( ;;)
{
if (UART1_data_ready())
{
PORTB=UART1_read();
}
}
}
```

3-Au niveau de quel PIC ce programme est-il implémenté ? EXPLIQUER.
4-Que représente (8929) sur le programme ?
5-Donner la différence entre une communication série et parallèle.
6-Quel sont les ports utilisés lors d'une communication parallèle ?

►Exercice 2 :

1-Donner une petite définition d'un microcontrôleur.
2-C'est quoi un PIC ?
3-Déchiffrer l'acronyme : RISC.
4-A quoi sert la mémoire de donnée EEPROM ?
5- Le registre OPTION ce présente comme suit :

RBPU/	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

-Quel est le rôle du bit RBPU ?

6-On désire faire l'interface entre un bouton-poussoir et l'entrée RB0/INT. Schématiser l'ensemble avec RBPU =0 et RBPU=1

7-Le registre INTCON ce présente comme suit :

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

-Quel est le rôle du bit GIE ?

-Quel est le rôle du bit EEIE ?

8-Quelle est l'utilité du registre STATUS ?

9-A quoi sert le registre de travail W ?

10-Quel est le rôle du Program Counter ?

11-Donner le schéma utilisé pour piloter un PIC16F84 en utilisant un oscillateur RC.

12-Donner le schéma utilisé pour piloter un PIC16F84 en utilisant un oscillateur à QUARTZ.

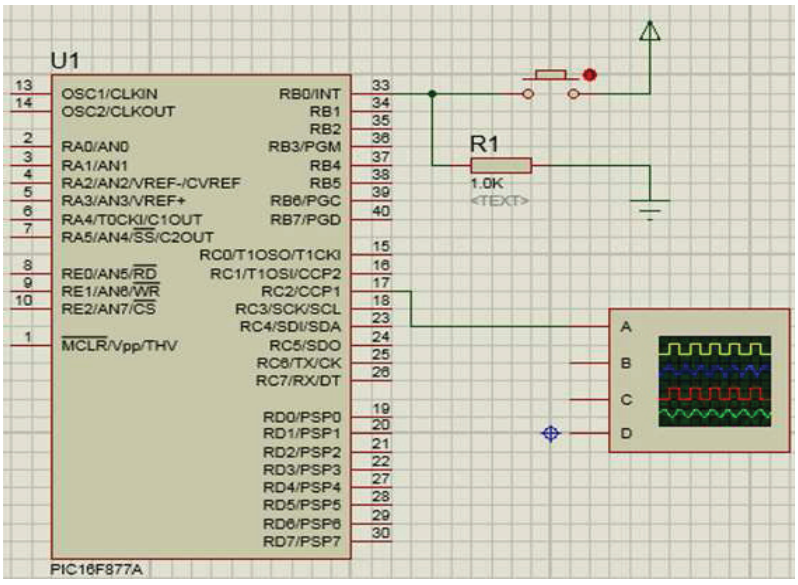
►**Exercice 3** :

Le programme suivant donne un signal carré sur CCP1 du PIC 16F877.

```
void main()
{
PWM1_Init(37000);
PWM1_Set_Duty(64);
PWM1_Start();
delay_ms (5000) ;
PWM1_Stop();
   }
```

-Expliquer chaque instruction du programme et schématiser ce signal.

-Modifier ce programme avec un bouton poussoir (une interruption) (Figure) qui peut incrémenter le **a** pour changer le temps de l'état haut t_H (un pas de 20 par exemple).



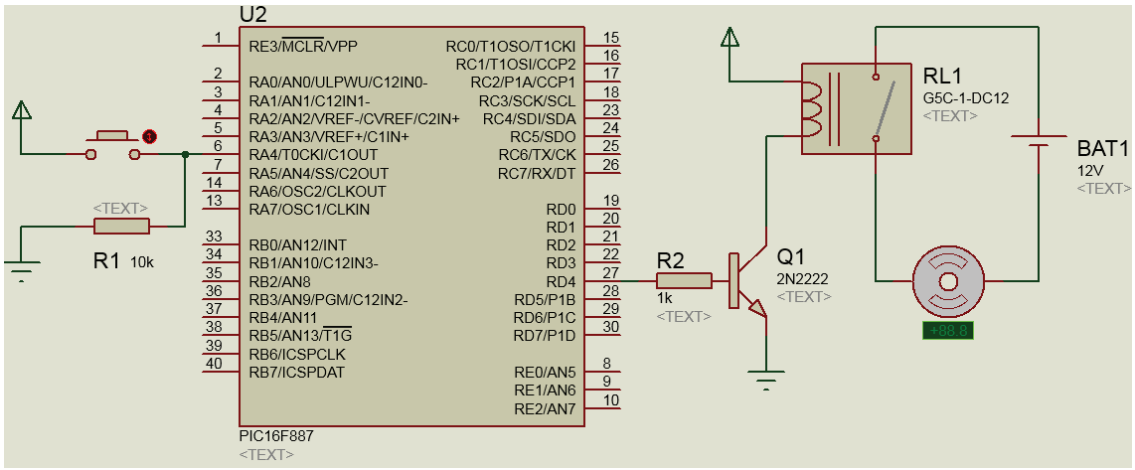
TD N°7 : Registre

►Exercice 1 :

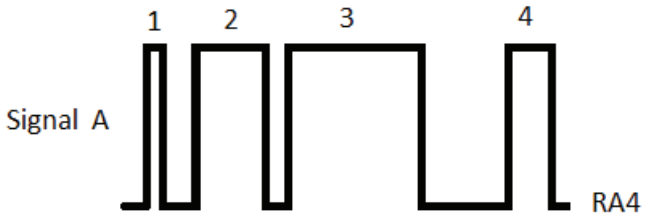
Lors de la programmation d'un PIC 16f877 ; le module TIMER0 est utilisé.
1-Que représente ce module et comment fonctionne-t-il ?
-Avec une fréquence d'oscillation égale à 4MHz (fréquence de l'oscillateur externe).
2-Calculer le temps entre une interruption et une autre.
3-Expliquer comment rendre ce temps (temps d'exécution d'une interruption) égale à une seconde.
4-Peut-on utilisé le TIMER1 pour avoir un temps d'exécution égale à une seconde.
5- C'est quoi un pré-deviseur ?

►Exercice 2 :

La Figure suivante représente un Microcontrôleur : 16F887 qui oscille à 10Mhz (Quartz) avec un Relai, une Batterie de 12V et un Moteur-DC.




```
Char TEST=33;
Void main (){
PORTA =0XFF;
PORTD=0;
TRISD=0b11110111;
OPTION_REG.b5=1;
OPTION_REG.b3=1;
TMR0= 0b 00011110;
for(;;){if(TMR0==TEST)
(PORTD.b3=1);}}
```



Le fonctionnement du moteur dépend du programme C :
1-Modifier le schéma pour un bon fonctionnement.
2-Expliquer comment démarrer le moteur DC.
3-Expliquer le fonctionnement du relai.
4-Modifier le programme pour arrêter ce moteur.
5-Expliquer comment le signal « A » injecter au niveau de RA4 démarrera le moteur.1pts
6-Si l'entrée RA4 est utilisée comme entrée horloge (oscillateur du PIC), comment peut-on démarrer le moteur ?
7-Que signifie OPTION_REG.b5=1 (T0CS=1) ?

►**Exercice 3 :**

Le registre INTCON (figure 1) est un registre très utilisé au niveau d'un PIC.
1-Ce registre à-il une fonction générale, ou une fonction spécifique ? Est-il attaché à un périphérique donné ?
2-Indiquer son emplacement au niveau du PIC ? Existe-il dans un PIC 16F84 ?
3-Expliquer comment Configurer le Pic pour qu'il utilise le registre INTCOM avec un exemple de votre choix.

Figure 1

4- Placer les modules suivants dans leurs emplacements d'origine (tableau) ?
-Registre de configuration (14 bits), la PILE, TIMER0, STATUT, SFR, OPTION_REG, registre de travail (W), Instructions exécutables, mémoire FLASH, program conter PC, TRISB.

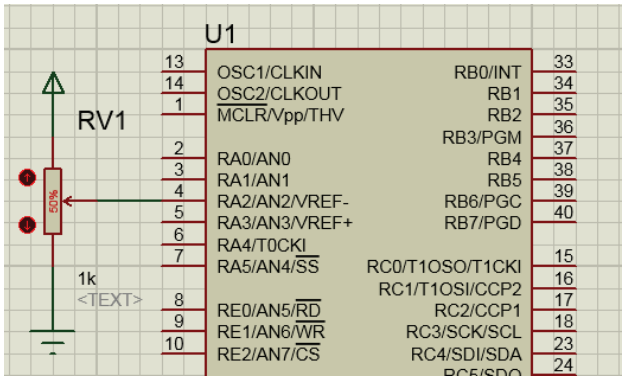
EEPROM	ROM	RAM	Pas d'adresse

TD N°8 : Conversion Analogique Digitale

►Exercice 1 :

Soit la conversion analogique digitale réalisée par le PIC 16F877 avec le programme suivant :

```
unsigned int a;
void main() {
TRISB=0x00; PORTB=0x00;
a=0;
ADC_Init();
for(;;)
 { a= ADC_Read(2);  PORTB = a >> 7;  }}
```

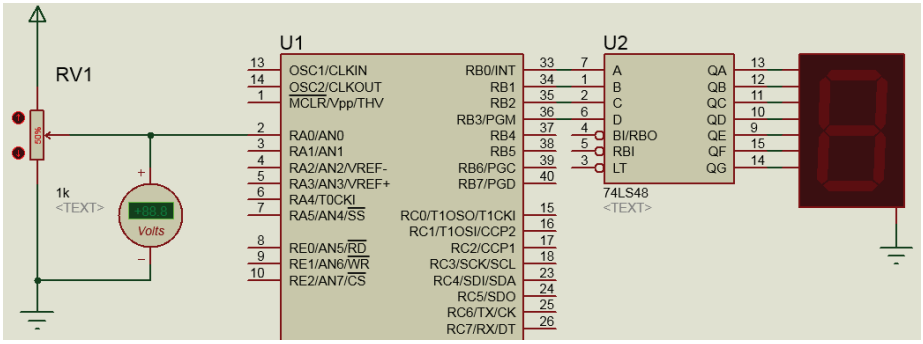


1- Le convertisseur A/D ainsi réalisé est un convertisseur de combien de bits ?
2-Donner le code binaire de 0.5 V.
- L'instruction (**adcon1=0b10001000**) permet d'ajouter deux tensions de référence (0V pour VREF- et 7V pour VREF+).
3-Réécrire le programme avec l'ajout de cette instruction.
4-Donner le code binaire à 100 % du potentiomètre.
5-Est-il possible de réaliser un convertisseur A/D de 12 bits avec ce PIC ? Expliquer !

►Exercice 2 :

Soit la conversion analogique digitale réalisée par le PIC 16F877 avec le programme suivant :

```
unsigned int a;
void main() {
trisB=0x00;portB=0x00;
a=0;
ADC_Init();
for(;;)
    {
    a= ADC_Read(0);
    PORTB = a >> 7;
    }
}
```



1-Avec ce programme et avec une tension a l'entrée égale à 2.5 V. Quel est le chiffre affiché ? Expliquer !
2-Modifier ce programme pour avoir un 7 sur l'afficheur 7 segments pour la valeur 2.5 v à l'entrée. Expliquer !
3-Ajouter deux tensions de référence et modifier le programme pour un fonctionnement normal du convertisseur analogique digital. Expliquer !

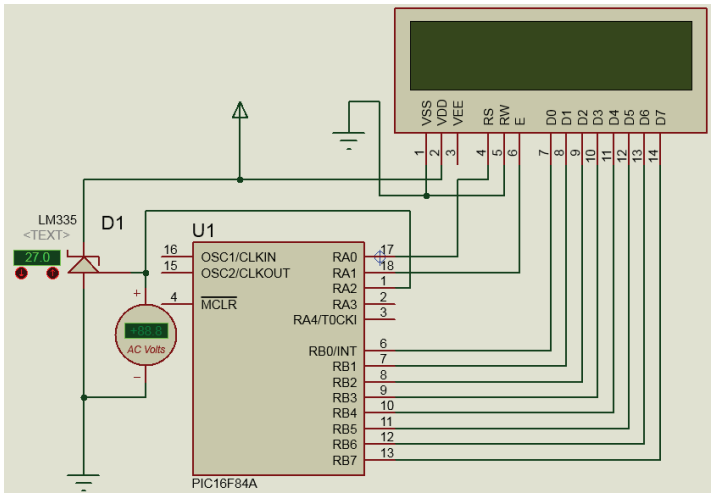
2v ➔ 2 3v ➔ 3 etc.….

►Exercice 3 :

La figure suivante représente une simulation réalisée avec un PIC, pour l'affichage de la température.

Programme :

```
sbit LCD_RS at RA0_bit;
sbit LCD_EN at RA1_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;

// Pin direction
sbit LCD_RS_Direction at TRIS   A0_bit;
sbit LCD_EN_Direction at TRISA1_bit;
sbit LCD_D0_Direction at TRISB0_bit;
sbit LCD_D1_Direction at TRISB1_bit;
sbit LCD_D2_Direction at TRISB2_bit;
sbit LCD_D3_Direction at TRISB3_bit;
char af1,af2;
unsigned int a,de;

void main() {
TRISB=0;TRISA=0;
PORTB=0;PORTA=0;
   LCD_Init();
   ADC_Init();
   Lcd_Cmd(_LCD_CURSOR_OFF);
   Lcd_Out(1,1, "TEMPERATURE :");Lcd_Out(2,11,"CELCUS");
   for(;;)
  a=ADC_Read(0);
   PORTB=a;
    PORTB=a>>8;
   if (a>=559){Lcd_Out(2,2, "+");} else {Lcd_Out(2,2, "-");}
   de= (a-559)/2;
   af1=de/10;lcd_chr(2,7,af1+48);
   af2=de-(af1*10);Lcd_chr(2,8,af2+48);
```



1- Corriger les erreurs dans cette simulation pour un bon fonctionnement.
2- La conversion analogique digitale réalisée est de combien de bits ?
3- Pourquoi on utilise le voltmètre dans cette simulation

Remerciements

Je tiens à remercier Mr ……et Mr……, pour tout l'intérêt qu'ils ont témoigné pour examiner ce travail.

Je tiens en outre à exprimer ma reconnaissance et ma sympathie à tous ceux qui m'ont témoigné amitié et patience le long de mes années d'enseignement.

Enfin, un grand merci pour mes parents pour leurs soutiens durant toutes ces années. Je ne saurais être qu'infiniment reconnaissant quant aux sacrifices qu'ils ont consentis. Un merci spécial à ma femme, qui m'a encouragée, soutenue et motivée sans cesse pour arriver au bout de ce travail, un grand merci pour tout.

Références

[1] Christian Tavernier, Programmation en C des PIC, Edition Dunod, France, 2005.

[2] V. Tourtchine, Programmation en mikroC. Application pour les microcontrôleurs de la famille PIC, Travaux Pratiques, Département Physique, Université M'Hamed Bogara, Boumerdes, 2012.

[3] Jacques Weiss, Microcontrôleurs PIC (Cas du 16F628), Suplec Campus de Renes ; Février 2002.

[4] Christian Dupaty Système à Microcontrôleurs, Edition CMP, Georges Charpak, Ecole Nationale Supérieur des Mines, 2010.

[5] Sylvain Montagny, Microprocesseurs et Microcontrôleurs, Université de Savoie, France, 2014.

[6] V. Tourtchine, Microcontrôleur de la famille PIC, Support de cours et prise en main du logiciel MPLAB, Département Physique, Université M'Hamed Bogara, Boumerdes, 2009.

[7] Hassen Jedid et Hichem Bargaoui, Architecture des microcontrôleurs, Edition Esprit, Ecole Supérieur Privée d'ingénierie et de technologies. 2011.

[8] Jerome Vicente, Les microcontrôleurs, Dpt ME, Option SIIC 2ème année, Ecole polytechnique, Université de Merseille,2006

[9] Philippe Letenneur, Les microcontroleurs PIC 16F87x, GRANVILLE, 2003.

[10] F. Senny, Introduction aux microcontrôleurs et à leur assembleur Illustration par le PIC16F877, Université de Liège, Faculté des Sciences Appliquées, 2007.

Webographie

[11] Bigonoff (bigocours@hotmail.com) : "La programmation des PICs" -

http://fribotte.free.fr/bdtech/cours/pic16f84/ ** :

[12] Lycée Jacquard : "Le PIC 16FXX" - http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm

[13] Lycée Jacquard : "MPLAB" - http://ejacquard.free.fr/dossier_lycee/Pic/cours_pic.htm

[14] Microchip : "PIC16F8X – 18-pin Flash/EEPROM 8-bit micro-controllers" – DS30430C,

http://www.microchip.com/1010/suppdoc/ *

[15] Microchip : "PIC16F8X – EEPROM memory programming specification" – DS30262E,

http://www.microchip.com/1010/suppdoc/

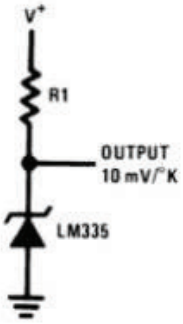
Annexes

LM335 - Precision Temperature Sensor

Features

- Directly calibrated in °Kelvin
- 1°C initial accuracy available
- Operates from 400 µA to 5 mA
- Less than 1Ohm dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200°C overrange
- Low cost

Typical Application



Parametric Table expand

Supply Min	5 Volt
Quiescent Current_	1000 uA
Temperature Min	-40 deg C
Temperature Max	100 deg C
Sensor Gain	10 mV/Deg K

General Description

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors.

MICROCHIP

PIC16F84A

18-pin *Enhanced* Flash/EEPROM 8-Bit Microcontroller

Devices Included in this Data Sheet:

- PIC16F84A
- Extended voltage range device available (PIC16LF84A)

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of data RAM
- 64 bytes of data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 special function hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt on change
  - Data EEPROM write complete

Peripheral Features:

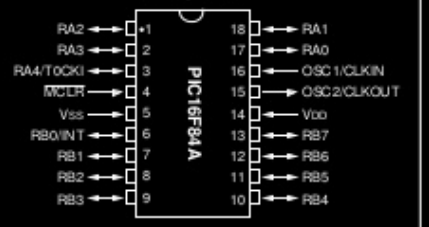
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

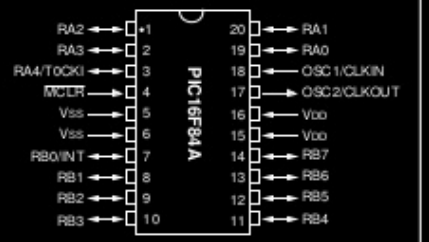
- 1000 erase/write cycles *Enhanced* Flash program memory
- 1,000,000 typical erase/write cycles EEPROM data memory
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Code-protection
- Power saving SLEEP mode
- Selectable oscillator options

Pin Diagrams

PDIP, SOIC



SSOP



CMOS *Enhanced* Flash/EEPROM Technology:

- Low-power, high-speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

MICROCHIP

PIC16F87X

28/40-pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

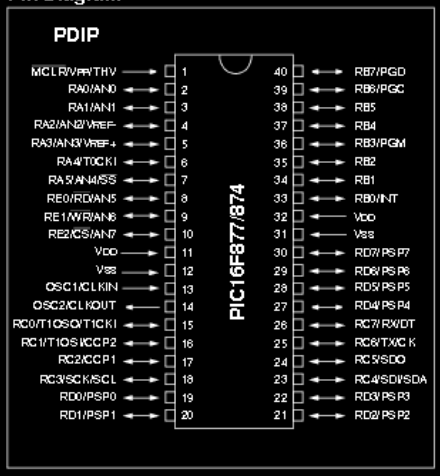
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High-performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM data memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial and Industrial temperature ranges
- Low-power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

PDIP



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master Mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

MICROCHIP

PIC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input
  DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (V_{REF}) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Code ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com