AVANT-PROPOS

Ce cours est consacré à l'étude des microcontrôleurs. Il est destiné aux étudiants des établissements de l'enseignement supérieur (ISETs), concernés par les études des systèmes micro-programmés. Il peut être également utile aux praticiens, désirant améliorer leur niveau théorique.

Le livre présente des chapitres qui sont composés pratiquement de la même façon :

- Un aperçu théorique, concernant les connaissances théoriques et les schémas correspondants ;
- Un nombre d'exercices à résoudre et des questions de contrôle.

Le présent ouvrage reflète l'expérience acquise par l'auteur aux cours de trois années de travail pédagogique dans l'enseignement supérieur, ainsi que ses études dans les cycles technicien et ingénieur en sciences de l'informatique industrielle & l'automatique à l'Institut National des Sciences Appliquées et de Technologie de Tunis.

Le cours présenté dans ce recueil, a été élaboré par l'auteur lui-même et l'accessibilité des explications a été testée au cours des travaux pendant deux semestres universitaires avec les étudiants de l'Institut Supérieur des études Technologiques de Nabeul.

L'auteur adresse d'avance ses remerciements aux lecteurs qui voudront bien faire part de leurs critiques et de leurs remarques constructives.

Auteur : *Mme. Yosra Rkhissi Kammoun*

PREAMBULE

Population :	Étudiants du 2eme niveau Electrique de l'ISETs
Spécialité :	Automatique et Informatique Industrielle
Crédits :	42 heures par semestre
Volume Horaire :	3 heures par semaine sur 14 semaines
Nature cours :	Cours Intégrés (1/2 cours , 1/2 TDs)
Pré-requis :	Traitement de données, électronique générale
Langue :	Français

Objectifs du cours

Comprendre l'architecture d'un système à microcontrôleur et être capable d'écrire un programme en langage évolué pour une cible à microcontrôleur 16F877 et plus généralement de transmettre une culture des systèmes micro-programmés.

Évaluations

- Test N°1 de 30 minutes, après avoir achevé les deuxième chapitre relatifs aux structures du microcontrôleurs PIC 16877.
- Devoir surveillé de 1 heure, à la fin du toisième chapitre relatif à la programmation du PIC 16F877 avec le compilateur CCS.
- Test N°2 de 30 minutes, après avoir achevé le chapitre relatif aux interruptions.
- Test N°3 de 30 minutes, après avoir achevé le chapitre relatif aux CAN.
- Test N°4 de 30 minutes, après avoir achevé le chapitre relatif aux Timers.
- Examen final écrit de 1 heure et demi sur tout le programme.

Objectifs Pédagogiques

- Connaître l'architecture d'un systèmes micro-programmésà base de microprocesseur, ses éléments constitutifs, et son fonctionnement et l'interaction entre les différentes unités.

- Comprendre l'architecture d'un microcontrôleur en particulier le PIC 16F877 et étudier ses différents constituants.
- Etre capable d'écrire un programme langage évolué C pour une cible à microcontrôleur PIC 16F877
- Maîtriser les mécanismes d'interruptions, les convertisseurs analogiques numériques du PIC et les timers...

SOMMAIRES

CHAPITRE 1
SYSTEMES MICRO-PROGRAMMES A BASE DE MICROPROCESSEUR

1. Introduction aux systèmes micro-programmés

Le développement de l'électronique numérique a suscité l'apparition de plusieurs types de composants très puissants en particulier les systèmes micro-programmés.

Leur aptitude à s'adapter aux contraintes technologiques de plus en plus complexes, leur capacité de gérer un grand nombre de fonctionnalités variées et leur coût de revient faible a encouragé leur utilisation dans plusieurs applications tant domestiques qu'industrielles.

Le développement de ces composants programmables remplacent de plus en plus l'électronique classique vu que les circuits intégrés analogiques ou logiques ne peuvent plus résoudre des fonctions de plus en plus complexes.

Historiquement, les constructeurs développèrent d'abord les systèmes micro-programmés intégrés dans les calculateurs de bureau ou de poche, avec des codes d'ordre orientés vers le calcul numérique. Puis maitrisant cette technique ils offrirent des circuits d'usage généraux : les microprocesseurs.

Les systèmes micro-programmés ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs. C'est d'ailleurs l'élément de base pour de tels systèmes.

==> 1971 : premier microprocesseur 4 bits 4004 d'Intel .

La miniaturisation des transistors a permis d'augmenter considérablement la capacité d'intégration sur silicium. On est passé rapidement du processeur 4 bits au :

- processeur 8 bits.
- processeurs 16 bits.
- processeurs 32 bits.
- processeurs 64 bits

Cette miniaturisation a offert des possibilités de réaliser des systèmes embarquées.

2. Définition d'un microprocesseur

Un microprocesseur ou processeur ou encore CPU (**C**entral **P**rocessing **U**nit) est l'unité intelligente de traitement des informations. c'est un circuit intégré complexe)est un circuit

intégré à très grande échelle d'intégration (VLSI) chargé d'organiser les tâches précisées par le programme, de les décoder et d'assurer leur exécution. Il doit aussi prendre en compte les informations extérieures au système et assurer leur traitement.

Ils présent trois avantages principaux : ils sont économiques et offre une souplesse d'emploi inhérente à la programmation.

3. Modèle de base d'un microprocesseur

Dans un microprocesseur, on retrouve :

- une **U**nité **A**rithmétique et **L**ogique (UAL)
- une **U**nité de **C**ontrôle (UC)
- des registres
- des bus ou chemins de données

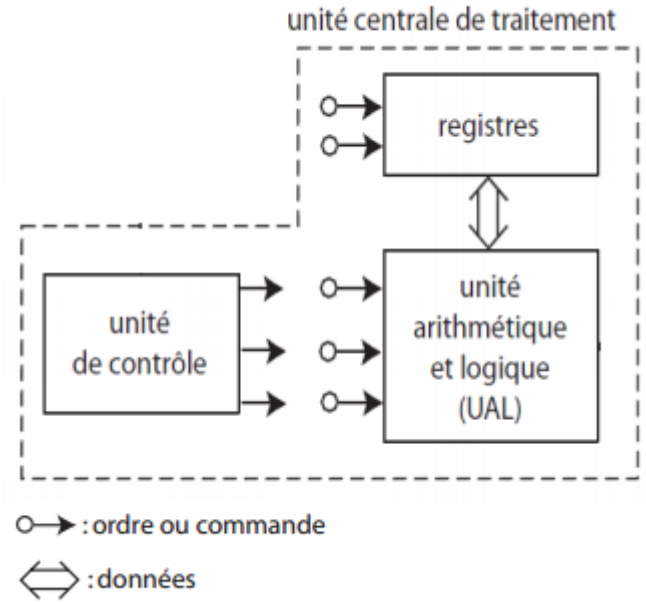


Fig. 1.1 : Structure de base d'un microprocesseur

3.1 Unité Arithmétique et Logique

Elle dispose de circuits réalisant des opérations des fonctions logiques (ET, OU, comparaison, décalage,...) ou arithmétique (addition, soustraction,...).

En entrée de l'UAL, on a des commandes permettant d'activer les opérations, venant de l'unité de contrôle En sortie, on a les résultats des opérations et les conditions qui sont en fait les entrées de l'unité de contrôle.

L'UAL est composé de :

- **Les accumulateurs :** Ce sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.
- **L'Unité Arithmétique et Logique:** C'est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage, etc...) ou arithmétique (Addition, soustraction...)
- **Le registre d'état :** Il est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateur d'état ou flag ou drapeaux (Retenue, débordement, zéro, ...).

3.2 Unité de contrôle ou séquenceur

L'unité de contrôle est une circuit logique séquentiel chargée de séquencer l'algorithme et de générer les signaux de contrôle pour piloter les éléments du chemin de données.

Elle envoie des commandes à l'unité de traitement qui va exécuter les traitements.

L'unité de control contient:

- **Le compteur de programme (PC : Programme Counter) appelé aussi Compteur Ordinal (CO) :** Il est constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de la prochaine instruction à exécuter
- **Le registre d'instruction et le décodeur d'instruction** : Chacune des instructions à exécuter est transféré depuis la mémoire dans le registre instruction puis est décodée par le décodeur d'instruction.
- **Bloc logique de commande (ou séquenceur)** : IL organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction de l'instruction qu'il a à exécuter. Il s'agit d'un automate réalisé de façon micro-programmée

3.3 Les bus

Un bus est un ensemble de lignes de communications groupés par fonction. Il permet de faire transiter (liaison série/parallèle) des informations codées en binaire entre deux points.

Il est caractérisé par le nombres de lignes et la fréquence de transfert.

Il existe 3 Types de bus :

- **Bus de données** (bi-directionnel): permet de transférer entre composants des données ,

ex. : *résultat d'une opération, valeur d'une variable, etc.*

Le nombre de lignes du bus de données définie la capacité de traitement du microprocesseur ; selon le microprocesseur la largeur du bus peut être de 8 bits, 16 bits, 32 bits, 64 bits.

- **Bus d'adresses** (uni-directionnel): permet de transférer entre composants des adresses,

ex. : *adresse d'une case mémoire, etc.*

L'espace adressable peut avoir 2n emplacements, avec n est le nombre de lignes du bus d'adresses.

- **Bus de contrôle** (bi-directionnel): permet l'échange entre les composants d'informations de contrôle [bus rarement représenté sur les schémas].

ex. : *périphérique prêt/occupé, erreur/exécution réussie, etc.*

3.4 Les registres

C'est un espace mémoire interne au processeur. On distingue deux types : à usage général qui permettent à l'UAL de manipuler des données et les registres d'adresses qui sont connectés au bus d'adresse.

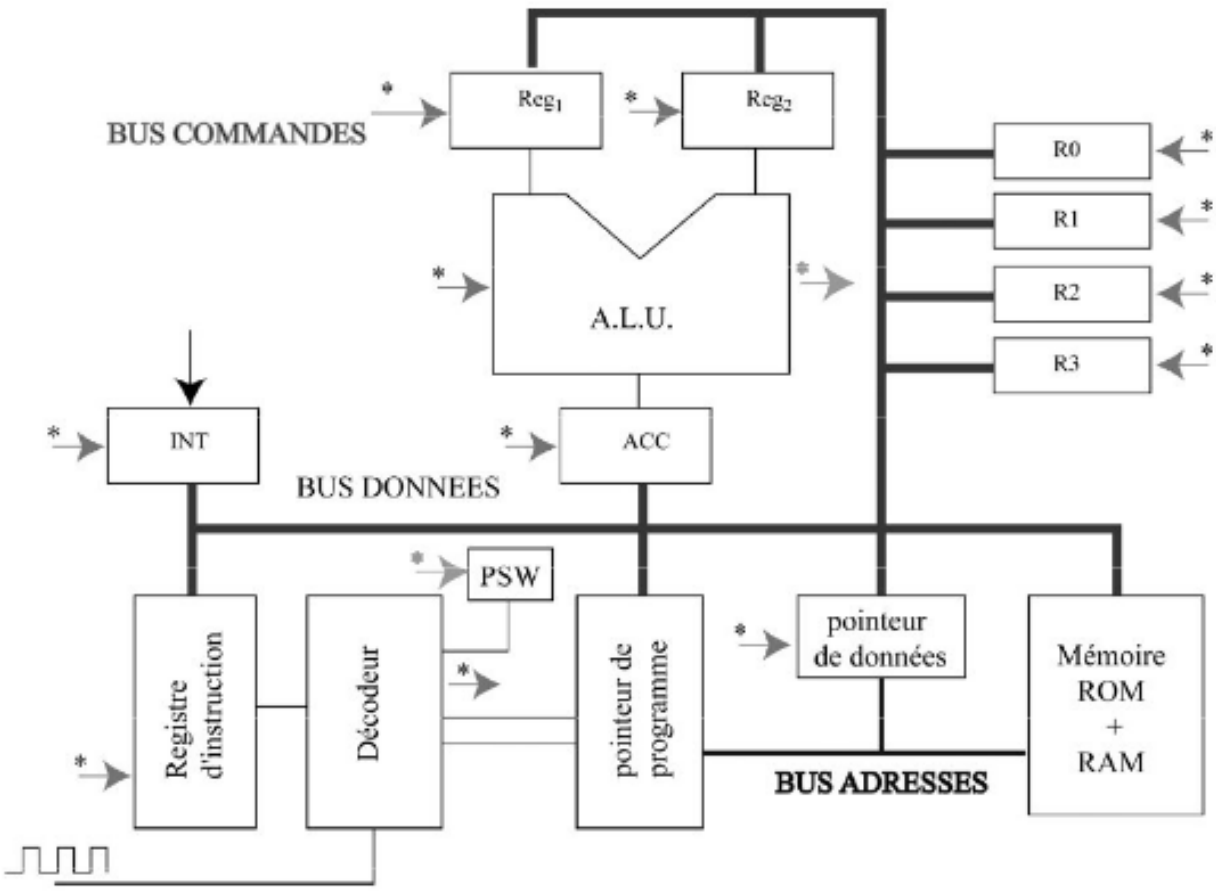


Fig. 1.2 : Structure interne d'un microprocesseur

4. Les mémoires

Le processeur exécute les instructions machines présente dans la mémoire et traite les données qu'elle contient : le fonctionnement du microprocesseur est entièrement conditionné par le contenu de celles-ci.

La mémoire peut être vue comme un ensemble de cellules ou cases contenant chacune une information : une instruction ou une donnée. Chaque case mémoire est repérée par un numéro d'ordre unique : son adresse.

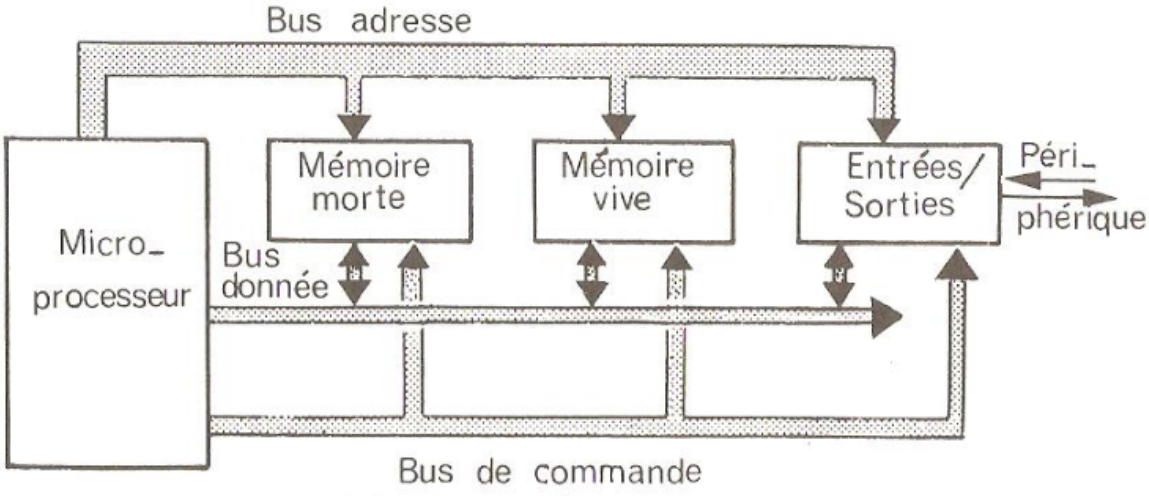
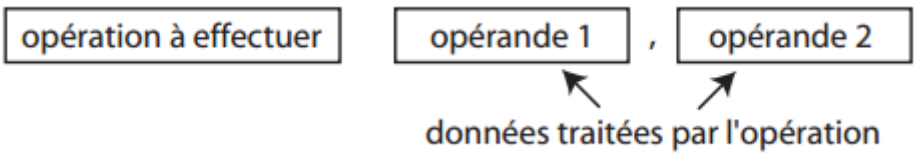


Fig. 1.3 : Structure générale d'un système piloté par un microprocesseur

Une case mémoire peut être lue ou écrite par le microprocesseur (cas des mémoires vives) ou bien seulement lue (cas des mémoires mortes).

- **Format d'une instruction**



Les opérandes sont stockées dans des mémoires RAM.

Un jeu d'instruction est l' ensemble d'opérations élémentaires effectué par le microprocesseur.

5. Fonctionnement d'un microprocesseur

Le traitement d'une instruction peut être décomposé en plusieurs phases. Celles-ci sont au nombre de trois :

- Recherche de l'instruction

- Décodage (decode)
- Exécution (execute)

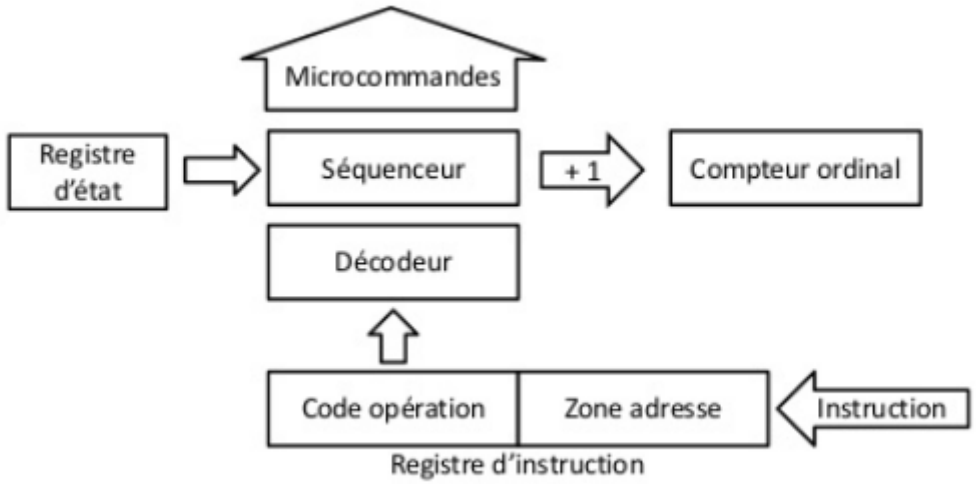


Fig. 1.4 : Fonctionnement d'un microprocesseur

5.1 Recherche de l'instruction

Pour exécuter les instructions dans l'ordre établi par le programme, le microprocesseur doit savoir à chaque instant l'adresse de la prochaine instruction à exécuter. Le microprocesseur utilise un registre contenant cette information : Le **pointeur d'instruction**

Le contenu de PC est placé sur le bus des adresses. L'unité de contrôle (UC) émet un ordre de lecture, au bout d'un certain temps (temps d'accès à la mémoire) le contenu de la case mémoire sélectionné est disponible sur le bus des données. L'unité de contrôle charge la donnée dans le registre d'instruction pour décodage.

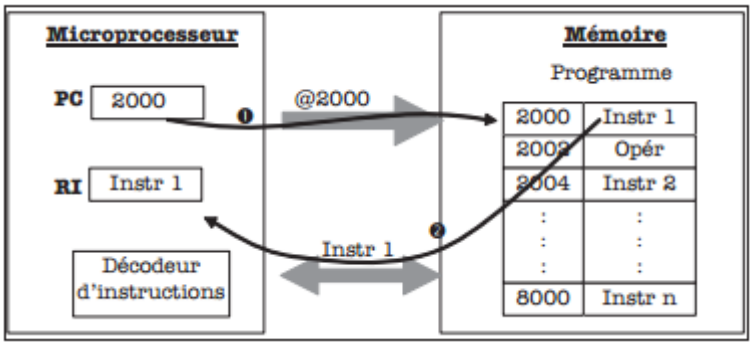


Fig. 1.5 : Recherche d'instruction

5.2 Décodage

Pour savoir quel type d'opération doit être exécuté (addition, soustraction, ...), le microprocesseur lit le premier octet de l'instruction pointée par le pointeur d'instruction (code

opératoire) et le range dans le registre d'instruction. Le code opératoire est décodé par des circuits de décodage contenus dans le microprocesseur. Des signaux de commande pour l'UAL sont produits en fonction de l'opération demandée qui est alors exécutée.

Pendant que l'instruction est décodée, le pointeur d'instruction est incrémenté de façon à pointer vers l'instruction suivante. Puis, le processus de lecture et de décodage des instructions recommence.

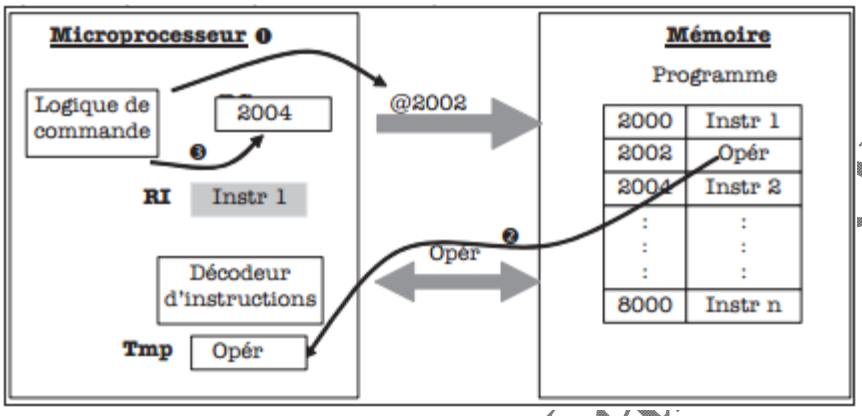


Fig. 1.6 : Récupération de l'opérande

5.3 Exécution

A la suite de chaque instruction, le registre d'état du microprocesseur est actualisé en fonction du dernier résultat.

6. Architecture d'un microprocesseur

Pour l'organisation des différentes unités, il existe deux architectures possibles:

6.1 Architecture de Von Neuman

La mémoire programme, la mémoire données et les périphériques d'entrées/sorties partagent le même bus s'adresses et de données.

Inconvénient: L'exécution d'une instruction nécessite plusieurs échanges de données sur le seul et unique bus dévolu à cet usage puisqu'il faut tout d'abord aller chercher le code de l'instruction puis le ou les données qu'elle doit manipuler.

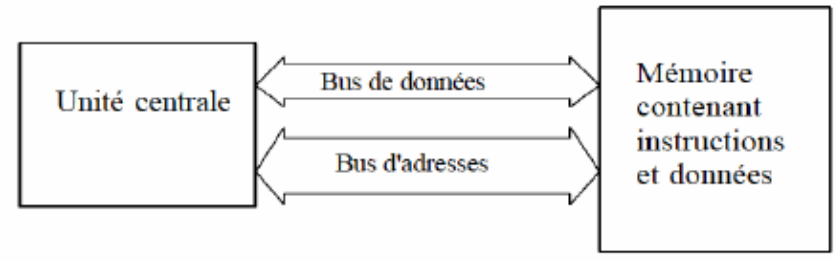


Fig. 1.7 : Architecture de Von Neuman

6.2 Architecture de Harvard

Cette architecture sépare systématiquement la mémoire de programme de la mémoire des données : l'adressage de ces mémoires est indépendant. Ce type d'architecture est utilisé sur des microcontrôleurs qui ont connu un développement important ces dernières années.

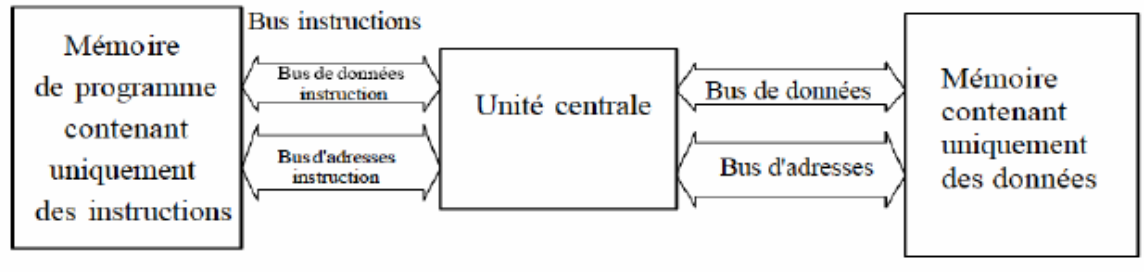


Fig. 1.8 : Architecture de Harvard

Quoique cette architecture puisse être complexe mais elle est performante: Gain en terme de vitesse d'exécution des programmes :

L'exécution d'une instruction ne fait plus appel qu'à un seul cycle machine puisque l'on peut simultanément, grâce au deux bus, rechercher le code de l'instruction et la ou les données qu'elle manipule

Travaux dirigés N°1

Exercice 1 :

On considère le système de la figure 1.

1. Que représente ce schéma ?

2. Expliquer le rôle de chaque partie.

3. Que peut-on conclure quant à la taille du bus de données ?

4. Que peut-on conclure quant à la taille du bus d'adresses ?

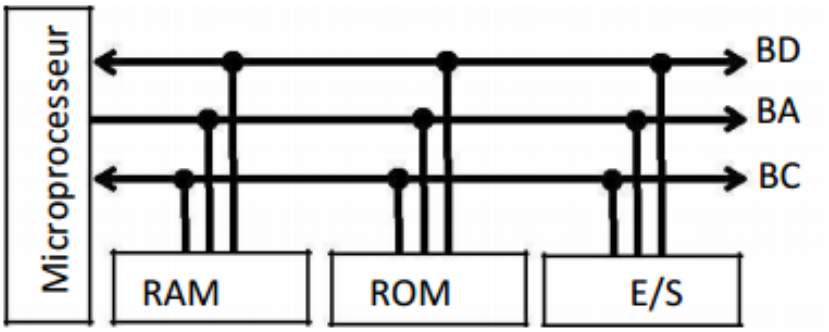


Figure 1.

Exercice 2 :

On considère le schéma synoptique ci-dessous :

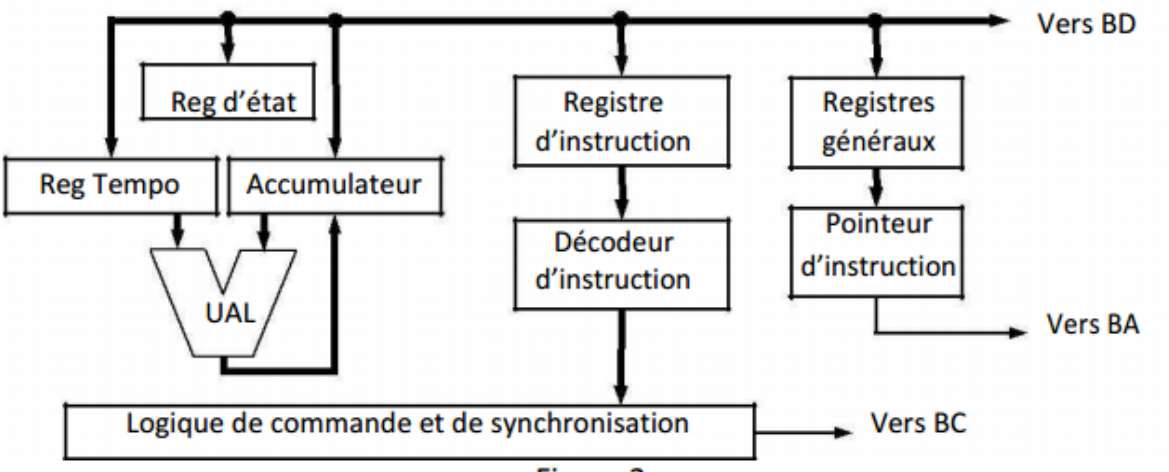


Figure 2.

1.Quel est le rôle de chaque unité ?

2.Donner les différentes étapes par lesquelles passe l'exécution d'une instruction.

Exercice 3 :

1. Combien de lignes d'adresses sont nécessaires pouradresser : 8 Ko, 64 Ko, 12 Ko et 100 Ko?

2. Le nombre de lignes d'adresses dépend-il de la taille du mot du système ?

CHAPITRE 2
MICROCONTROLEUR PIC 16F877

1. Du microprocesseur au microcontrôleur

Le microcontrôleur est un dérivé du microprocesseur. Sa structure est celle des systèmes à base de microprocesseurs. Il est donc composé en plus de l'unité centrale de traitement, d'une mémoire (mémoire vive RAM et mémoire morte ROM), une (ou plusieurs) interface de communication avec l'extérieur matérialisé par les ports d'entrée/sortie.

En plus de cette configuration minimale, les microcontrôleurs sont dotés d'autres circuits d'interface qui vont dépendre du microcontrôleur choisi à savoir les systèmes de comptage (TIMER), les convertisseur analogique/numérique (CAN) intégré, gestion d'une liaison série ou parallèle, un Watchdog (surveillance du programme), une sortie PWM (modulation d'impulsion),…

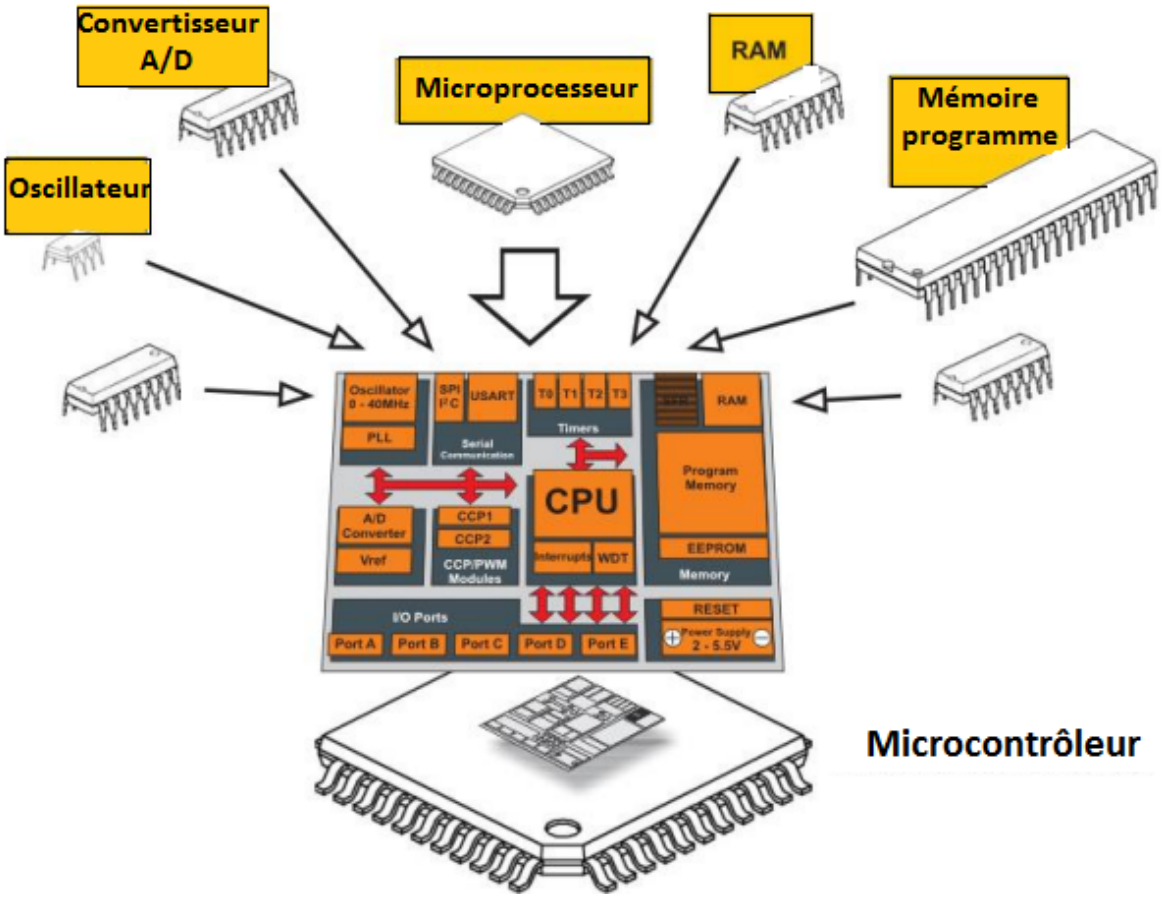


Fig. 2.1 : Contenu type d'un microcontrôleur

Les microcontrôleurs améliorent l'intégration et le coût (lié à la conception et à la réalisation) d'un système à base de microprocesseur en rassemblant ces éléments essentiels dans un seul circuit intégré. On parle alors de "système sur une puce" (en anglais : "System On chip").

Les microcontrôleurs sont plutôt dédiés aux applications qui ne nécessitent pas une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées/sorties. C'est le cas de contrôle de processus.

Les systèmes à microprocesseur sont plutôt réservés pour les applications demandant beaucoup de traitement de l'information et assez peu de gestion d'entrées / sorties. Les ordinateurs sont réalisés avec des systèmes à microprocesseur.

Il existe plusieurs famille de microcontrôleurs dont les plus connues sont :

Atmel : AT; familles AT89Sxxxx, AT90xxxx, …

Motorolla : famille 68HCxxx**, …**

Microship : PIC ; familles 12Cxxx, 16Cxxx, 16Fxxx, 18Fxxx, …

Intel : famille 80C186XX

STMicroelectronics : famille STX

Analog Devices : famille ADuC

Nous allons nous intéresser dans le cadre de ce cours à la famille Microchip **PIC** (**P**rogrammable **I**ntegrated **C**ircuit) de moyenne gamme (MIDRANGE).

2. Présentation d'un microcontrôleur PIC

Ils sont des composants dits **RISC** (**R**éduced **I**nstructions **C**onstruction **S**et), ou encore composant à jeu d'instructions réduit. Chaque instruction complexe peut être programmée par plusieurs instructions simples. Sachant que plus on réduit le nombre d'instructions, plus facile et plus rapide qu'en est le décodage, et plus vite le composant fonctionne.

La famille des PIC à processeur 8 bits est subdivisée à l'heure actuelle en 3 grandes catégories :

- ✓ **Base-Line :** ils utilisent des mots d'instruction de 12 bits.
- ✓ **Mid-Range :** ils utilisent des mots d'instruction de 14 bits.
- ✓ **High-End :** ils utilisent des mots d'instruction de 16 bits.

Il existe aussi des PIC à processeur 16 bits (**PIC24F/PIC24H**) et 32 bits (**PIC32M)** aussi.

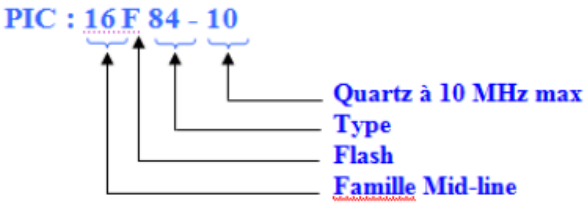
Toutes les PICs Mid-Range ont un jeu de 35 instructions, stockent chaque instruction dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en un cycle machine.

On atteint donc de très grandes vitesses, et les instructions sont de plus très rapidement assimilées.

L'horloge fournie au PIC est divisée par 4.C'est cette base de temps qui donne le temps d'un cycle. Si on utilise par exemple un quartz de 4MHz, on obtient donc 1000000 de cycles/seconde ; or, comme le PIC exécute pratiquement une instruction par cycle, hormis les sauts, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d'Instructions Par Seconde).

Pour identifier un PIC, on utilise simplement son appellation du type : **wwlxxyyy-zz**

- WW: Représente la catégorie du composant (12, 14, 16, 17, 18),
- L: Tolérance plus importante de la plage de tension.
- XX: Type de mémoire de programme:
  - ✓ C: EPROM ou EEPROM.
  - ✓ CR: PROM.
  - ✓ F: FLASH.
- YYY: Identification.
- ZZ: Vitesse maximum tolérable.



Les PICs sont des composants STATIQUES, c'est à dire que la fréquence d'horloge peut être abaissée jusqu'à l'arrêt complet sans perte de données et sans dysfonctionnement.

Ceci par opposition aux composants DYNAMIQUE, donc la fréquence d'horloge doit rester dans des limites précises.

Les microcontrôleurs PIC sont présentés en boîtier DIL (Dual In Line).Un point ou une encoche donne un repérage de la broche 1, ensuite il faut ce déplacer vers la droite pour avoir les autres broches. Ont fait le tour du circuit dans le trigonométrique.

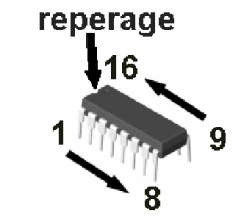


Fig. 2.2 : Repérage des broches

3. Microcontrôleur PIC 16F877

Dans la suite du chapitre, on va prendre comme exemple le PIC 16F877 et présenter sa structure interne et externe. Les éléments essentiels du PIC 16F877 sont :

3.1 Structure interne

➢ Caractéristiques de la CPU

- CPU à architecture RISC (8 bits)
- Mémoire programme de 8 Kmots de 14 bits (Flash),
- Mémoire donnée de 368 Octets,
- EEPROM donnée de 256 Octets,
- 14 sources interruptions
- Générateur d'horloge de type RC ou quartz (jusqu‟ à 20 MHz)
- 05 ports d'entrée sortie
- Fonctionnement en mode sleep pour réduction de la consommation,
- Programmation par mode ICSP (**I**n **C**ircuit **S**erial **P**rogramming) 12V ou 5V,
- Possibilité aux applications utilisateur d'accéder à la mémoire programme

➢ Caractéristiques des périphériques

- Timer0 : Timer/Compteur 8 bitsavec un prédiviseur 8 bits
- Timer1 : Timer/Compteur 16 bits avec un prédivision de 1, 2, 4, ou 8 ; il peut être incrémenté en mode veille (Sleep), via une horloge externe,
- Timer2 : Timer 8 bits avec deux diviseurs (pré et post diviseur)
- Deux modules « Capture, Compare et PWM » :
  Module capture 16 bits avec une résolution max. 12,5 ns,
  Module Compare 16 bits avec une résolution max. 200 ns,
  Module PWM avec une résolution max. 10 bits,
- Convertisseur Analogiques numériques multi-canal (8 voies) avec une conversion sur 10 bits,
  Synchronous **S**erial **P**ort (SSP) SSP, Port série synchrone en mode I2C (mode maitre/escalve),
- Universel **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter (USART) : Port série universel, mode asynchrone (RS232) et mode synchrone

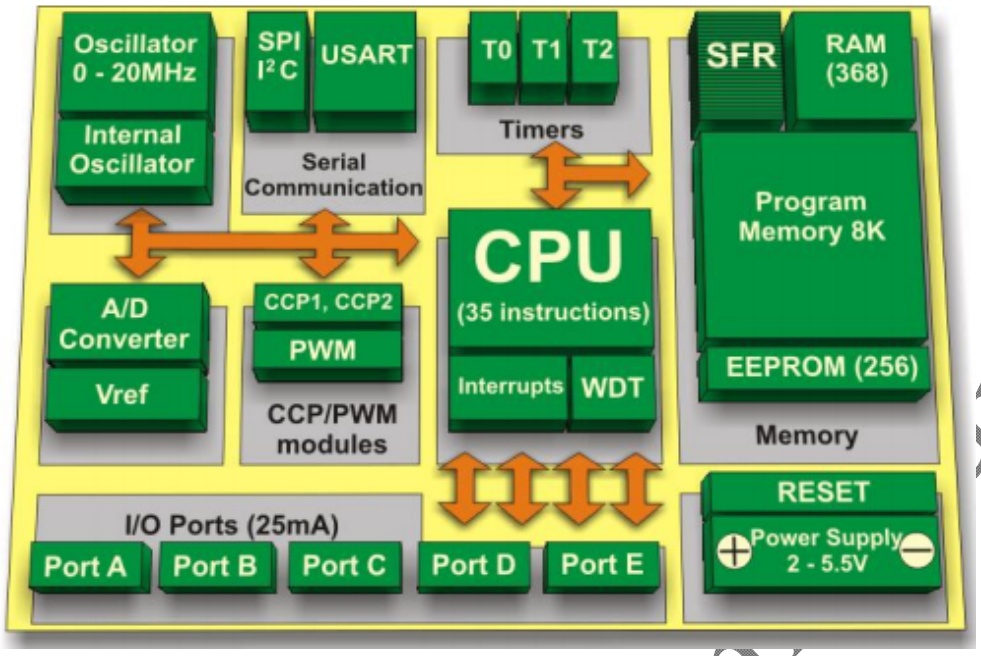


Fig. 2.3 : Architecture interne du PIC 16F877

3.2 Structure externe

Le PIC16F877 est un circuit intégré de 40 broches :

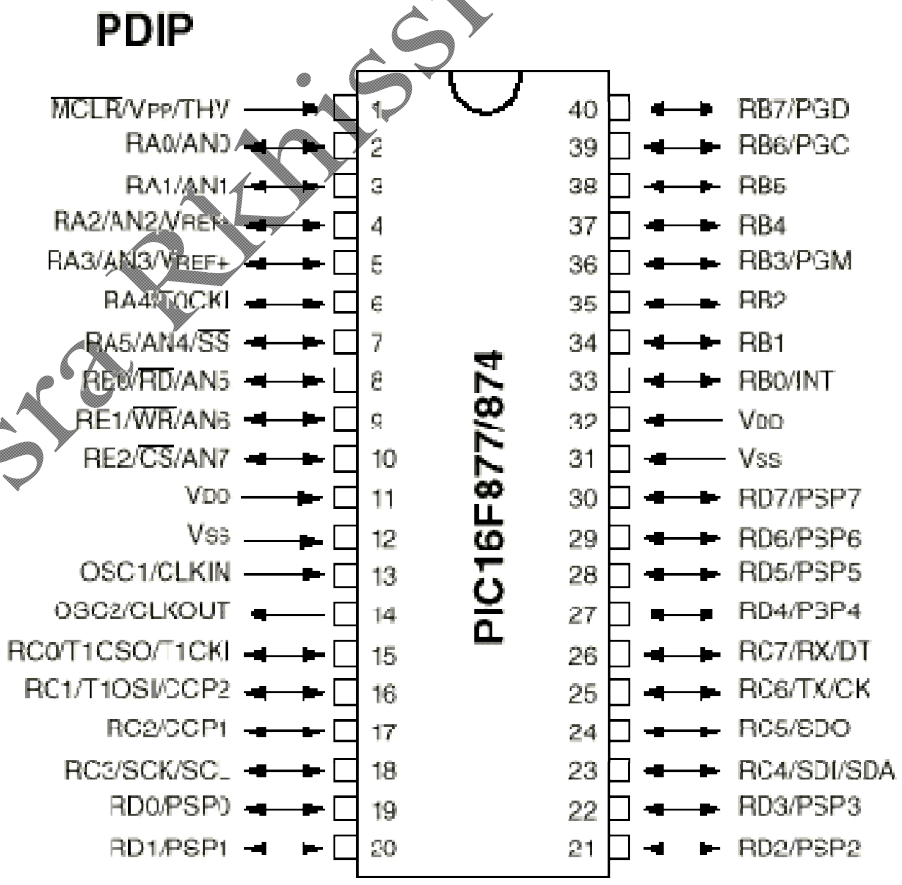


Fig. 2.4 : Brochage du PIC 16F877

Certaines pattes ont plusieurs fonctions :On dit que les fonctions sont multiplexée

3.2.1 L'alimentation

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.

3.2.2 Cadencement du PIC

Le PIC 16F877A peut fonctionner en 4 modes d'oscillateur.

- ✓ **LP** : Low Power crystal : quartz à faible puissance.
- ✓ **XT** : Crystal/Resonator : quartz/résonateur en céramique.
- ✓ **HS** : High Speed crystal/resonator : quartz à haute fréquence/résonateur en céramique HF.
- ✓ **RC** : Circuit RC (oscillateur externe).

Dans le cas du 16F877, on peut utiliser un quartz allant jusqu'à 20Mhz relié avec deux condensateurs de découplage, du fait de la fréquence importante du quartz utilisé.

Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs

3.2.3 Circuit Reset MCLR

La broche MCLR (Master Clear) a pour effet de provoquer la réinitialisation du microprocesseur lorsqu'elle est connectée à 0.

Lorsque le signal de "RESET" est activé, tous les registres sont initialisé et le compteur programme se place à une adresse spécifique appelée "Vecteur de RESET".

3.2.4 Ports d'entrées/sortie

Le PIC 16F877 dispose de 5 ports :

- ✓ Port A : 6 pins I/O numérotées de RA0 à RA5.
- ✓ Port B : 8 pins I/O numérotées de RB0 à RB7.
- ✓ Port C : 8 pins I/O numérotées de RC0 à RC7.
- ✓ Port D : 8 pins I/O numérotées de RD0 à RD7.
- ✓ Port E : 3 pins I/O numérotées de RE0 à RE2.

A chaque port correspondent deux registres :

- ✓ Un registre direction pour programmer les lignes soit en entrée, soit en sortie *TRISA, TRISB, TRISC, TRISD* et *TRISE.*
- ✓ Un registre de données pour lire ou modifier l'état des broches. *PORTA, PORTB, PORTC, PORTD* et *PORTE*

Pour déterminer les modes des ports (I/O), il faut sélectionner leurs registres TRISX:

- ✓ Le positionnement d'un bit à « 1 » place le pin en entrée.
- ✓ Le positionnement de ce bit à « 0 » place le pin en sortie.

La plupart des broches des PORTs sont partagées avec des périphériques. En général si un périphérique est utilisé, les broches correspondantes ne peuvent pas être utilisées comme broches d'entrée/sortie.

Au reset, les lignes des ports A et E sont configurées en entrées analogiques, les autres lignes sont configurées en entrées digitales.

Le courant absorbé ou fourni peut atteindre 25 mA.

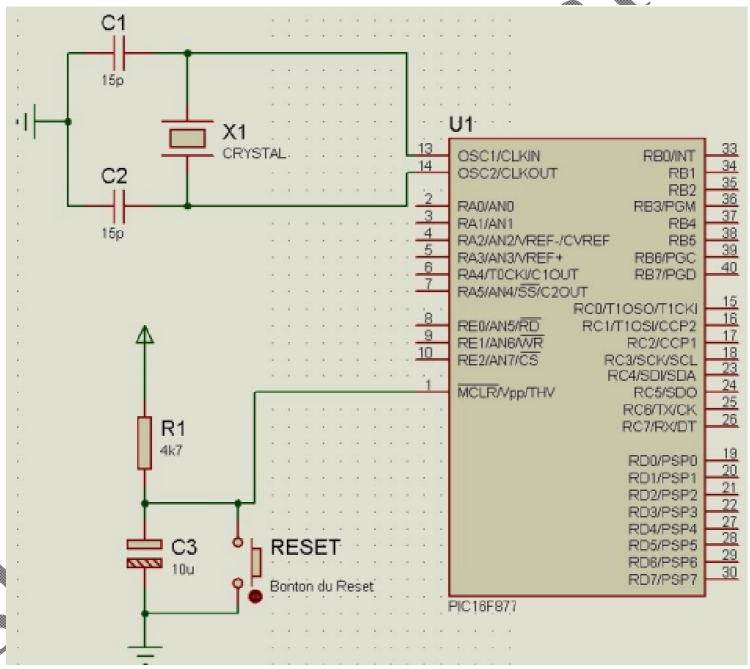


Fig. 2.5 : Circuit RESET et oscillateur d'un PIC 16F877

- ➢ **Port A**

Les broches port A, excepté RA4, sont multiplexées, avec les entrées du convertisseur analogique numérique (AN0 .. AN4) .

La broche RA4 est multiplexé avec l'entrée d'horloge externe du timer0 (RA4/T0CKI).

- ➢ **Port B**

Le port B peut être programmé pour un tirage à 5V (*pull up*) de toutes ses lignes que l'on peut mettre ou non en service en mode entrée uniquement. Elles sont automatiquement désactivées quand le port est configuré en sortie.

En mode entrée, chaque broche du PORTB doit être maintenue à un niveau haut par l'intermédiaire de résistances de 10 k pour ne pas déclencher d'interruptions imprévues.

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.

- **Port C**

Le port C est partagé avec liaisons, les timers 1 et 2 et les modules CCP.

- **Port D et E**

En plus de leur utilisation comme PORTS E/S; les ports D et E, permettent au microcontrôleur de travailler en mode PSP (Parallel Slave Port) c'est-à-dire, qu'il peut être interfacé avec un autre microprocesseur. Dans ce cas le PORTD représente le bus de données et le PORTE les signaux de contrôle(RD\, WR\ et CS\).

Le PORTE peut être aussi, configuré en mode analogique pour former avec le PORTA les 8 entrées du convertisseur analogique numérique. Par défaut, le PORTE est configuré comme port analogique, et donc, comme pour le PORTA,

3.2.5 Chien de garde

Un chien de garde est un circuit électronique ou un logiciel utilisé en électronique numérique pour s'assurer qu'un automate ou un ordinateur ne reste pas bloqué à une étape particulière du traitement qu'il effectue. C'est une protection destinée généralement à redémarrer le système, si une action définie n'est pas exécutée dans un délai imparti ;

Dans le PIC, il s'agit 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, 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

CHAPITRE 3

PROGRAMMATION C DES PIC

AVEC LE COMPILATEUR CCS - C

1. Outils de programmation d'un PIC

Comme toute solution programmable, le microcontrôleur nécessite un outillage informatique et éventuellement un programmateur. A chaque microcontrôleur correspond son outil de développement.

Pour développer une application fonctionnant à l'aide d'un microcontrôleur, il faut disposer de :

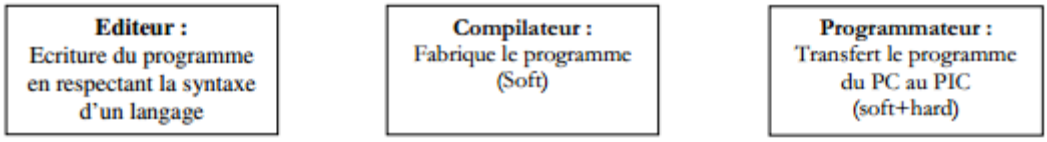


Fig. 3.1 : Outils de développement pour un PIC

- **Le compilateur :** Logiciel traduisant un programme écrit dans un langage donné (C, basic, assembleur) en langage machine.
- **Le programmateur** : Transfert le programme compilé (langage machine) dans la mémoire du microcontrôleur. Il est constitué d'un circuit branché sur le port série du PC, sur lequel on implante le PIC, et d'un logiciel permettant d'assurer le transfert. Il existe différents logiciels, nous utiliserons Icprog

Les microcontrôleurs PIC utilisent la plate-forme logiciel de développement MPLAB **I**ntegrated **D**evelopment **E**nvironment IDE.

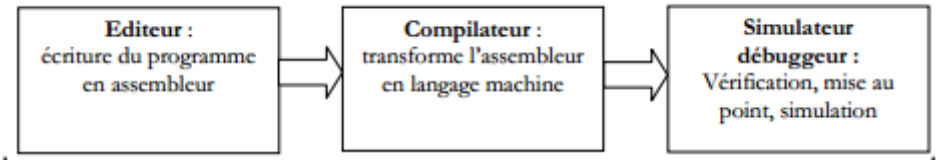


Fig. 3.2 : Environnement MPLAB IDE

Dans l'environnement MPLAB, Le programme doit être écrit en assembleur, langage proche de la machine et donc nécessitant un long apprentissage.

Le langage de programmation adopté pour programmer le PIC 16F877 est le langage évolué : Le code source écrit en langage c doit donc être compilé en assembleur à l'aide d'un compilateur C.

2. Le langage C

Le langage C dispose de beaucoup d'avantages. Il est :

- ✓ PORTABLE : Les modifications d'un programme pour passer d'un système à un autre sont minimes.
- ✓ COMPLET : Un texte C peut contenir des séquences de bas niveau (proches du matériel) en assembleur.
- ✓ SOUPLE : Tout est possible en C mais une grande rigueur s'impose.
- ✓ EFFICACE : On réfléchit (devant une feuille de papier) et on écrit (peu)

Le compilateur C de la société CCS (**C**ustom **C**omputer **S**ervices) est un compilateur C adapté aux microcontrôleurs PICs. Il ne respecte pas complètement la norme ANSI, mais il apporte des fonctionnalités très intéressantes.

3. Notion de filière de développement

On désigne par filière de développement l'ensemble des outils qui interviennent pour passer du fichier texte (source codé en C) au code objet (code machine) téléchargé dans le microcontrôleur.

Les étapes de génération d'un programme écrit en langage C sont :

- ✓ L'édition du fichier source mon_programme.C avec un éditeur de texte (simple sans mise en forme du texte).
- ✓ La compilation du fichier source pour obtenir un fichier objet : mon_programme.ASM. La compilation est la transformation des instructions C en instructions assembleur pour microcontrôleur PIC.
- ✓ L'édition de liens permet d'intégrer des fonctions prédéfinies. Le programme auxiliaire Éditeur de liens (linker ou binder) génère à partir du fichier mon_programme.ASM un fichier exécutable mon_programme.HEX compatible avec le PIC.

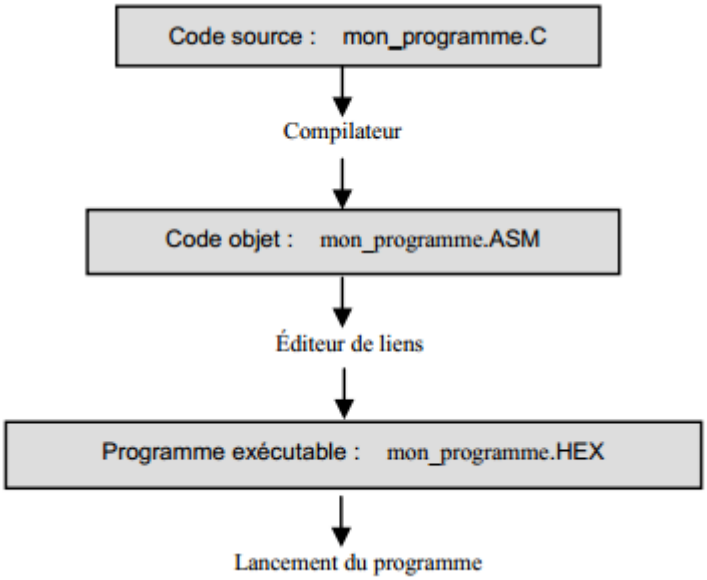


Fig. 3.3 : Etapes de génération d'un programme écrit en langage C

Fichiers (Extensions)	Description du contenu du fichier	Fichiers (Extensions)	Description du contenu du fichier
.C	Fichier source en langage C.	.HEX	Code objet (exécutable) téléchargé dans le µC
.H	Entête de définition des broches, Registres, Bits de Registres, Fonctions, et directives de pré-compilation.	.TRE	Montre l'organisation du programme sous forme d'arbre (découpages en fonction) et l'utilisation de la mémoire pour chaque fonction.
.PJT	Fichier de projet (pas obligatoire).	.COF	Code machine + Informations de débugage
.LST	Fichier qui montre chaque ligne du code C et son code assembleur associé généré.	.ERR	Erreurs éventuelles rencontrées durant la compilation.
.SYM	Indique la correspondance entre le nom des symboles (variables, bits, registres) et leur adresses hexadécimale en mémoire.		
.STA	Fichier statistique sur l'espace mémoire occupé. etc.		

Fig. 3.4: Fichiers générés

4. Règles de bases

Toutes instructions ou actions se terminent par un point virgule **;**

Une ligne de commentaires doit commencer par /*et se terminer par */ ou commencer par //.

Un bloc d'instructions commence par **{**et se termine par **}**.

Un programme en C utilise deux zones mémoires principales :

- ✓ La zone des variables est un bloc de RAM où sont stockées des données manipulées par le programme.

- ✓ La zone des fonctions est un bloc de ROM qui reçoit le code exécutable du programme et les constantes.

5. Les variables et les constantes

5.1. Les constantes

Les constantes n'existent pas, c'est-à-dire qu'il n'y a pas d'allocation mémoire, mais on peut affecter à un identificateur (Nom : Il ne doit pas dépasser 32 caractères, sans accent)
Une valeur constante par l'instruction #**define**.

Syntaxe : **< #define> <identificateur> <valeur> ;**

Exemple: #define PI 3.14

5.2. Déclarations spécifiques au compilateur CCS

#bit id = x,y

- ✓ Id : identifiant (Nom d'un bit)
- ✓ X : Nom du variable ou d'une constante
- ✓ Y : position du bit

Exemple : #bit RW = PORTA,2
#bit LED_R=PortB.4

#byte id = X

- ✓ Id: identifiant
- ✓ X: valeur 8 bits

Exemple :

```
#byte  PORTA  = 5        // adresse  du port A
#byte  PORTB  = 6        // adresse  du port B
#byte  PORTC  = 7        // adresse  du port C
#byte  PORTD  = 8        // adresse  du port D
#byte  PORTE  = 9        // adresse  du port E
```

5.3. Les Variables

Les variables sont définies par signé ou non signé, le type et l'identificateur.

Syntaxe: <signed> <type> <identificateur1>, …, <identificateurn>

L'identificateur : C'est le nom (Il ne doit pas dépasser 32 caractères, sans accent) affecté à la variable.

Le type : Il détermine la taille de la variable et les opérations pouvant être effectuées. On peut rajouter le mot signed devant le type du variable, alors la variable devient signée.

Exemples : Int A,B,C,D ;
Char MESSAGE[10] ;
Signed int A ; // Entier de type signé, de -128 à +127

Les types du compilateur CCS figurent dans le tableau ci-dessous :

Tab 3. 1: Types du compilateur CCS

Type	Taille	Valeur
int1	1 bit	0 ou 1
Int8	8 bits	De 0 à 255
int16	16 bits	De 0 à 655535
int32	32 bits	De 0 à 4 294 967 295
Char	8bits	De 0 à 255
Float	32 bits	
Short	1 bits	0 ou 1
Int	8 bits	De 0 à 255
Long	16 bits	De 0 à 655535

5.4. Représentation des différentes bases du compilateur CCS et du code ASCII

int a = 4 ; // Un nombre seul représente un nombre décimal.

int b = 0b1010 ; // Un nombre précédé de 0best un nombre binaire.

int p = 0x00FF ; // Un nombre précédé de 0xest un nombre hexadécimal.

char c = 'A'; char c = '0x41'; ou char c = '65; // Un caractère entre '' représente son code ASCII.

6. Les fonctions

- **Avec des paramètres d'entrée et un paramètre de sortie**

Syntaxe :

Type de la variable de retour nom de fonction (types nom des paramètres)

{

Instruction 1 ;

.

Instruction n ;

Return (valeur) ; // Valeur à renvoyer

}

Avec :

//Nom de la fonction :

//Description du rôle de la fonction :

//Paramètres d'entrée : Noms et types des paramètres d'entrée

//Paramètre de sortie : Nom et type du paramètre de sortie

- **Une fonction sans paramètres d'entrée et de sortie**

```
Void   nom de fonction (Void)
{
      Instruction 1 ;
.
.      Instruction n ;
}
```

Remarque : Il peut y avoir aussi des fonctions avec paramètres d'entrées et sans paramètre de sortie et vice versa.

7. Les Opérateurs

Tab 3. 2: Les opérateurs d'affectation et arithmétiques

Type	Symbole	Exemple
Affectation	=	x = 3; y = a - b;
Addition	+	a = a + b; b = a+ 5;
Soustraction	-	a = a-b; b = a- 5;
Moins unitaire	-	a = -b;
Multiplication	*	a = b*6;
Division	/	x = a / b;
Reste de la division	%	r = a % b;

Tab 3. 3: Les opérateurs de comparaison

Type	Symbole	Exemple
Egalité	==	a == b;
Différent	!=	a != b;
Supérieur	>	a > b;
Supérieur ou égal	>=	a >=b;
Inférieur	<	a < b;

Inférieur ou égal	<=	a <=b;
Reste de la division	%	r=a%b;

Le résultat de la comparaison peut prendre deux valeurs VRAIou FAUX.

-FAUX correspond à 0.

-VRAI correspond à toute valeur différente de 0

Tab 3. 4: Les opérateurs logique de comparaison

Type	Symbole	Exemple
Et logique	&&	c= a&&b; //c est vrai si a et b sont vrais, sinon c et faux
Ou logiqye	||	c= a||b; //c est vrai si a et b sont vrais, sinon c et faux
Non logique	!	c = !c; //c prend le complément de ce qu'il vaut

Ces opérateurs permettent la comparaison de conditions composées telles que (x > y)&& (u > v). Le résultat fourni est de type logique (0 ou 1).

Tab 3. 5: Les opérateurs binaires bit à bit

Type	Symbole	Exemple
Et logique	&	c= a&b;
Ou logiqye	|	c= a|b;
Et exclusif	^	c= a^b;
Complément à 1	~	a = ~a;
Décalage de n bits à droite	>>	a = b >> n;
Décalage de n bits à gauche	<<	a = b << n;

Exemples : a et b deux entiers tel que : a = 1100 0111 0101 0011 (0xC753)

b = 0001 1001 1010 1110 (0x19AE)

a & b = 0000 0001 0000 0010 (0x0102)

a | b = 1101 1111 1111 1111 (0xDFFF)

a ^ b = 1101 1110 1111 1101 (0xDEFD)

~ a = 0011 1000 1010 1100 (0x38AC)

~ b = 1110 0110 0101 0001 (0xE651)

a<<2 = 0001 1101 0100 1100 (0x1D4C)

8. Les structures répétitives

Tab 3. 6: Les structures itératives

Structure "while" : tant que ... faire ...	**Structure "do ... while"** : faire ... tant que...	**Structure "for"** : Pour <variable> allant de <valeur initiale> à <valeur finale> faire...	**Structure "if ... Else" :** Si <condition> faire ... sinon faire ...	**Structure "switch ... case".**

void main()
{
while
(condition vraie)
{
Action 1;
}
}
	void main()
	{
	do
	{
	Action ;
	}
	while
	(condition vraie)
	}
		void main()
		{
		For (i=m ; i<=n ;
		i=i+p)
		{
		Action ;
		}
		}
			void main()
			{
			if (condition vraie)
			{
			Action 1;
			}
			else
			{
			Action 2;
			}
			}
				switch (choix)
				case c1 : < sequence 1>
				case c2 : < sequence 2>
				case c3: < sequence 3>
				….
				case cn: < sequence n>
				default:
				< sequence_par_defaut> /

9. Les fonctions adaptées aux microcontrôleurs PIC

9.1. Les directives

- :**#use delay**

Syntaxe : **#use delay(*clock=fréquence*)**

Renseigne le compilateur sur la fréquence du quartz utilisé.

Exemple : #use delay(clock=4000000)

- **#fuses**

Syntaxe : #fuses ***options***

Permet de définir le mot de configuration. Les options sont :

- LP, XT, HS, RC : Choisir le type d'oscillateur
- WDT, NOWDT : Activer/désactiver le chien de garde

- PUT, NOPUT : Activer/désactiver le **P**ower **U**p **T**imer PUT(Timer spécial qui retarde le démarrage de l'exécution du programme aprèsque le PIC a été réinitialisé. Ce délai donne le temps de PIC oscillateur pour démarrer et se stabiliser).
- PROTECT, NOPROTECT : Activer/Désactiver la protection du code

Exemple : #fuses XT,NOWDT,NOPUT,NOPROTECT

- **#int_xxxx**

Spécifie la source de l'interruption.

Syntaxe :

#int_ext : interruption externe.

9.2. La gestion des entrées et des sorties

9.2.1 Configurer un port en entrée ou en sortie

Syntaxe : **set_tris_X(valeur)**

Avec :

- X : Nom du port : A .. E
- valeur : définie la configuration du port (1 entrée et 0 sortie)

Exemple :

```
set_tris_A(0b00001111) ; PA0 à PA3 en entrée et P A4 à PA7 en sortie
set_tris_A ( 0x0F );  // B7,B6,B5,B4 en sortie (0)
                      // B3,B2,B1,B0 en entrée (1)
```

9.2.2. Gestion des ports d'entrée ou en sortie

- **Mise à l'état bas d'une pine d'un port**

Syntaxe : output_low(pin)

- pin est une constante définie dans le fichier entête PIC16FXX.h

Exemple : output_low(PIN_A0); // mise à 0 de la broche A0

- **Mise à l'état haut d'une pine d'un port**

Syntaxe : output_high(pin)

Exemple : output_high(PIN_B2); // mise à 0 de la broche B2

- **Modifier l'état d'une pine d'un port**

Syntaxe : output_bit(pin,valeur)

Exemple : output_bit(PIN_B0, 0); // même rôle que output_low(pin_B0);

- **Modifier l'état d'un port**

Syntaxe : output_X(valeur)

Exemple : output_b(0xf0);); // mise à 1 de B7,B6,B5,B4
// mise à 0 de B3,B2,B1,B0

- **Lecture de l'état d'une pine**

Syntaxe : valeur=input(pin)

Exemple : if(input(PIN_A0)) // Si A0 est à l'état haut, écrire « A0 est active »
printf("A0 est active \r\n") ;

- **Lecture de l'état d'un port**

Syntaxe : valeur=input_X()

Exemple : Data = input_c(); //data reçoit l'état du port c

- **Mettre à 0 un bit d'une variable**

Syntaxe : BIT_CLEAR(*var, bit*)

Exemple :
a=0x1F
BIT_CLEAR(a, 3) // a devient 17 hexa.

- **Mettre à 1 un bit d'une variable**

Syntaxe : BIT_SET(*var, bit*)

Exemple :
a=0x1F
BIT_SET(a, 6) // a devient 3F hexa.

- **Test de l'état d'un bit d'une variable**

Syntaxe : BIT_TEST(*var, bit*)

Exemple :
a=0x1F
BIT_TEST(a, 2) // Le résultat est 1 car le bit 2 de la variable « a » est à 1.

9.3. Gestion des temporisations

Le compilateur intègre des fonctions très pratiques pour gérer les délais, à savoir :

delay_us(valeur) ; // temporisation en µS

delay_ms(valeur) ; // temporisation en mS

Pour pouvoir utiliser ces fonctions, il faut indiquer par la ligne ci-dessous la fréquence du Quartz de votre application. Cette ligne doit être définie au début du programme.

#use delay (clock=fréquence_du_quartz)

Exemples :

#use delay (clock=4000000) // Quartz de 4Mhz

#use delay (clock=20000000) // Quartz de 20Mhz

9.4. Gestion de la liaison série

Toutes les fonctions d'entrée et de sortie peuvent être redirigées sur le port série dumicrocontrôleur, Il suffit d'ajouter la ligne ci-dessous pour configurer le port série :

#use rs232 (BAUD=9600, xmit=PIN_C6, rcv=PIN_C7)

// Cette ligne configure la liaison série du PIC avec une vitesse de 9600 bauds.

Remarque : Cette ligne doit être définie au début du programme, après la ligne qui définie la Fréquence du quartz.

Alors les fonctions suivantes utiliseront le port série comme moyen de communication.

- **La fonction printf**

Cette fonction permet d'envoyer une chaîne de caractère formatée sur la liaison RS232 ou bien vers une fonction bien déterminée.

Syntaxe : printf (chaine) ou printf (Cchaine, valeurs...)

- Chaine peut être une constante de type chaîne ou un tableau de char terminé par le caractère null.
- Cchaine est une chaîne composée (voir ci-dessous).
- valeurs est une liste de variables séparées par des virgules.

La « Cchaine » doit être composée de chaînes de caractères constante et d'arguments de mise en forme des valeurs représenté par le symbole %wt (formatage) avec :

- w est optionnel et peut être compris entre 1 et 9. il indique sur combien de caractère va être le résultat ou 01-09 ou 1.1 to 9.9 pour les nombres à virgule.
- t est le type et peut être :
  - ✓ c Caractère ascii
  - ✓ c caractère ou chaîne de caractère
  - ✓ u entier (int8 ou int 16) non signé affiché en base décimale
  - ✓ x entier (int8 ou int 16) affiché en base hexadécimale en minuscules
  - ✓ X entier (int8 ou int 16) affiché en base hexadécimale en majuscules
  - ✓ d entier (int8 ou int 16) signé affiché en base décimale
  - ✓ f Float
  - ✓ Lx entier long (int32) affiché en base hexadécimale en minuscules

- ✓ LX long (int32 affiché en base hexadécimale en majuscules
- ✓ lu entier long (int32) non signé affiché en base décimale
- ✓ ld entier long (int32) signé affiché en base décimale
- ✓ : entier (int8 ou int 16) signé affiché en base octale

(Pour les autres arguments voir la documentation de CCS-Compiler)

Exemple :

```
printf("Bonjour !");
printf("\r\nMin: %2X Max: %2X\n\r",min,max);
// \n=LF (saut de ligne), \r=CR (retour à la première colonne)
printf("A/D value = %2x\n\r", value);
```

- **La fonction putc**

Cette fonction permet d'envoyer un caractère formatée sur la liaison RS232.

Syntaxe : putc (char)

- Char est un caractère 8 bits

- **La fonction puts**

Cette fonction permet d'envoyer une chaîne de caractères sur la liaison RS232, terminée par le passage à la ligne (LF)

Syntaxe : variable=puts(chaine)

- chaîne est une chaîne de caractère constante terminé par le caractère null

Exemple : puts(" | Salut ! | ");

- **La fonction getc**

Cette fonction permet de recevoir un caractère formatée sur la liaison RS232.

Syntaxe : variable=getc()

- Variable est un caractère 8 bits

Exemple :

```
void main () {
printf("Continuer (O,N)?");
do
{ reponse=getc(); }
while (reponse!='O' &&reponse!='N'); }
```

- **La fonction gets**

Cette fonction permet de recevoir une chaîne de caractère sur la liaison RS232

Syntaxe : variable=gets()

-Variable est une chaîne de caractère constante terminé par le caractère null

10. Structure d'un programme en C

```
#include  <16F84A.H>      //fichier de déclaration des registres internes du 16F84A.H
//Déclaration des adresses des ports E/S
#byte PORTA  = 5                 //adresse  du port A
#byte  PORTB  = 6               // adresse  du port B
//Déclaration des constantes
#define NB_MAX 100
//Affectation des entrées et sorties de type bit
#bit  BUZZER = PORTD7         //par exemple : Command d'un buzzer
//Fréquence du quartz
#use delay (clock=20000000)
//Configuration de la liaison série de PIC avec une vitesse de9600 bauds
#use rs232 (BAUD=9600, xmit=PIN_C6, rcv=PIN_C7)
//Déclaration du prototype de toutes les fonctions logicielles
//Nom de la fonction suivi d'un point virgule par exemple :
Void INIT_UC(Void) ;
//Déclaration des variables globales Par exemple :
Long DEBIT ;
Long VOULUME ;
Main()  //Programme principale
 {
     Instruction 1 ;
     Instruction n ;
 }
//Déclaration des fonctions logicielles Par exemple :
//Nom de la fonction : DECALAGE_DROITE
//Description du rôle de la fonction : décalage à droite de NB bits
//Paramètres d'entrée : entier VAL, entier NB
//Paramètre de sortie : entier RESULTAT
int  DECALAGE_DROITE (int VAL ,int NB)
{
```

```
    Int RESULTAT;
    RESULTAT = VAL >> NB;
   Return (RESULTAT);}
//Appel de la fonction: Nom de la fonction (nom des paramètres) ;
//Exemple :
A = 16 ;
B = DECALAGE_DROITE (A, 2) ;    //la valeur de A sera affectée à VAL
                                //la valeur 2  sera affectée à NB
                                //le résultat de la fonction sera affectée à B
```

Travaux dirigés N°2

Exercice 1 :

Proposer la ligne d'instruction qui permet de :

1. Mettre à 1 le bit B0 du port B sans modifier les autres bits.

2. Mettre à 0 le bit A3 du port A sans modifier les autres bits.

3. Inverser tous les états du port B.

Exercice 2 :

Ecrire un programme qui permet de lire le port A et d'écrire le complément sur le port B. L'état de RA4 est dupliqué sur les 3 bits du poids fort du port B (RB5-RB7).

Exercice 3 :

Soient les équations suivantes d'un système logique combinatoire :

$$S_1 = A + B + \overline{C}.\overline{B} \qquad S_2 = A.\overline{B} + \overline{C}.B + C$$

$$S_3 = \overline{A + B} + C.\overline{B} \qquad S_4 = A + \overline{C}.B$$

$$S_5 = B.\overline{C}.A$$

On propose le schéma de simulation suivant :

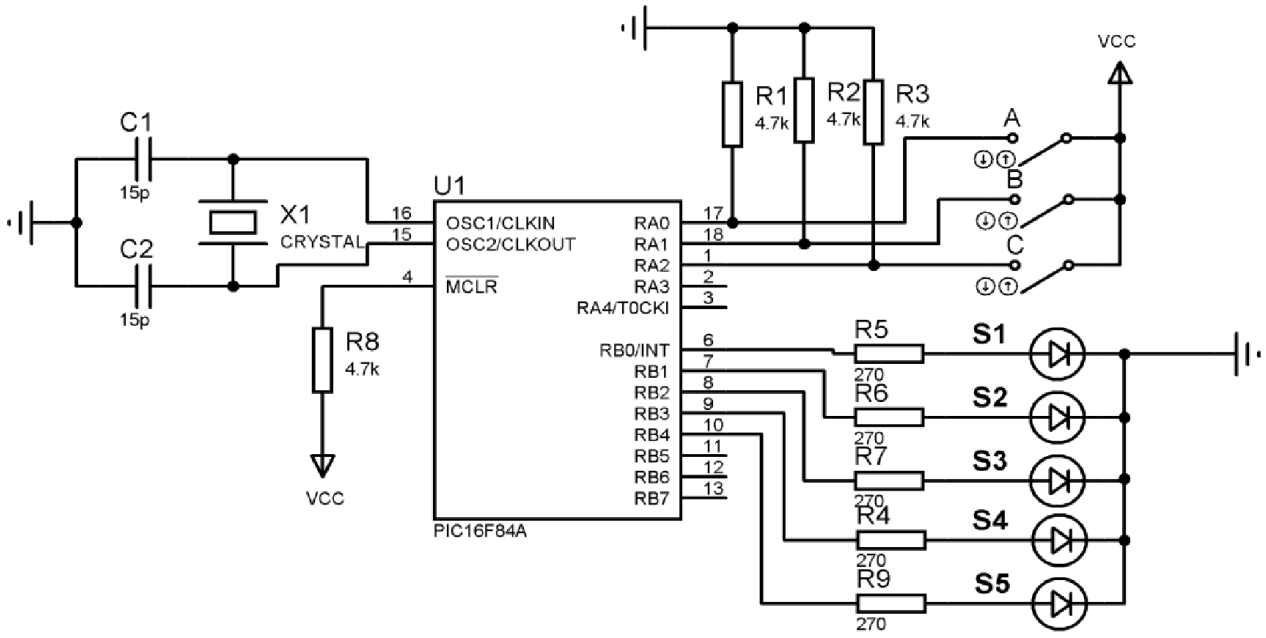


Figure 1

Ecrire un programme en C qui permet d'implanter le système combinatoire précédent.

Exercice 4 : Chenillard

Soit le montage suivant permettant de commander 8 diodes LED :

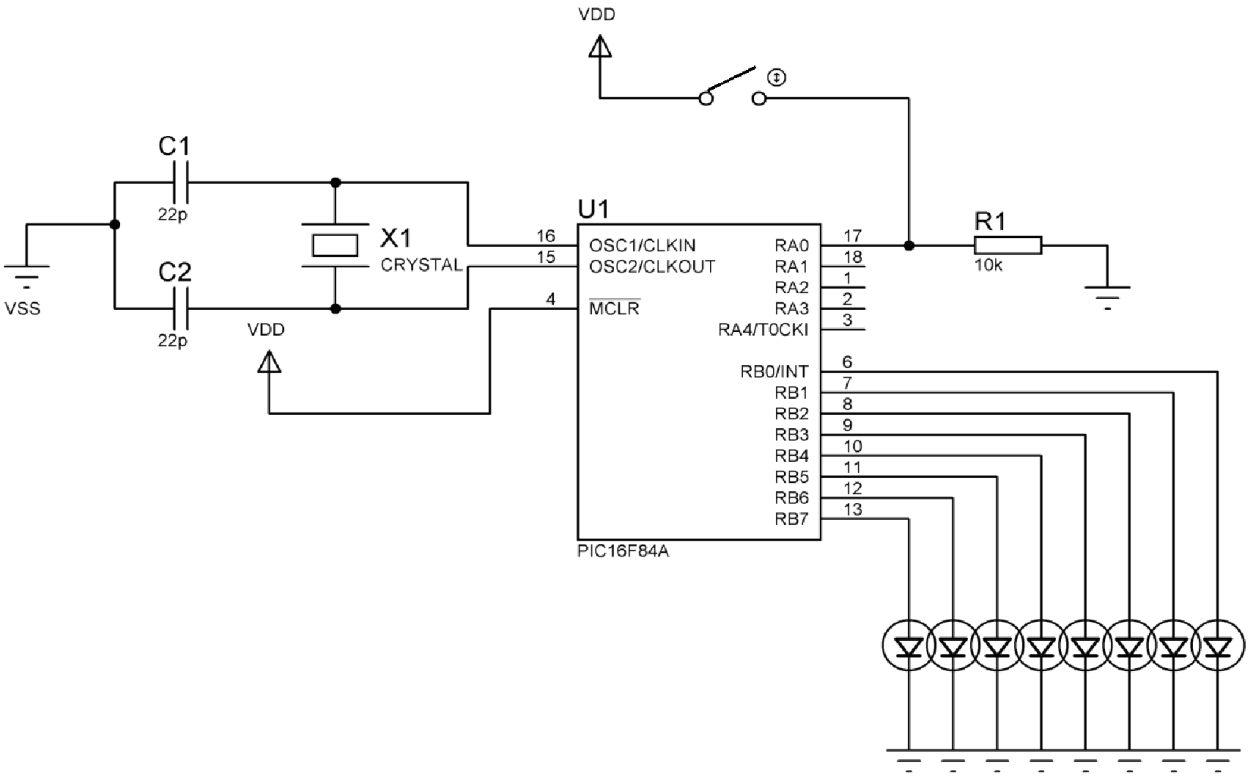


Figure 2

- Si RA0=0, les 8 diodes LED clignotent avec un délai d'une 1 secondes,
- Si RA0=1, on obtient le cycle répétitif suivant :

Diodes LED allumés	Durée
D0D1	1s
D2D3	2s
D4D5	3s
D6D7	4s
Aucune diode	1s

1. Ecrire un programme en C permettant de commander les diodes LED.

2. Ecrire un programme qui permet un défilement de l'allumage de diodes à intervalle de 1s sur le port B. L'allumage est actif à l'état haut ; le cycle de défilement est défini comme suit : une diode allumée, ensuite deux, … à la fin huit diodes.

Exercice 4 : UAL

Pour réaliser des opérations arithmétiques et Logiques, on s'adresse aux ports E/S du microcontrôleur 16F877 tels que les ports PORTC et PORTD désignent les ports d'entrées

binaires à 4bits **A** ($a_3a_2a_1a_0$) et **B** ($b_3b_2b_1b_0$), et le port PORTD représente les ports de la sortie **B** ($b_4b_3b_2b_1b_0$) affectée aux diodes LED permettant un affichage des résultats de sortie en binaire (d_4 étant la retenue).

Les bits PORTE.0, PORTE.1 et PORTE.2 représentent des entrées de sélection des opérations à réaliser, comme le précise le tableau 1 :

Bits de sélection PORTE.2 … PORTE.0	**Opérations**
001	AND
010	OR
011	XOR
100	+
101	-
110	*
111	/

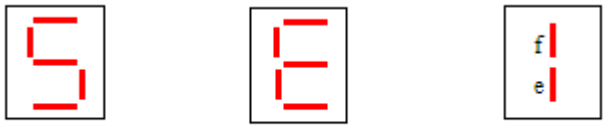
Tableau 1: Les opérations à réaliser

Etablir en premier lieu un programme en C permettant de réaliser les différentes opérations désignées par le tableau 1 selon figure 3.

N.B : Penser à utiliser l'appel de fonctions spécifiques pour chaque opération.

Exercice 5 : comparateur & compteur modulo 10

On se propose d'écrire un programme qui fera la comparaison entre deux mots X et Y à 4bits et afficher le résultat sur un afficheur 7-segment tant que PIN_E0 est active :



(Si X>Y) (Si X=Y) (Si X<Y)

1. Donner l'organigramme et le programme qui permet de réaliser la fonction comparateur.
2. Donner l'organigramme et le programme qui permet de créer un compteur modulo10.

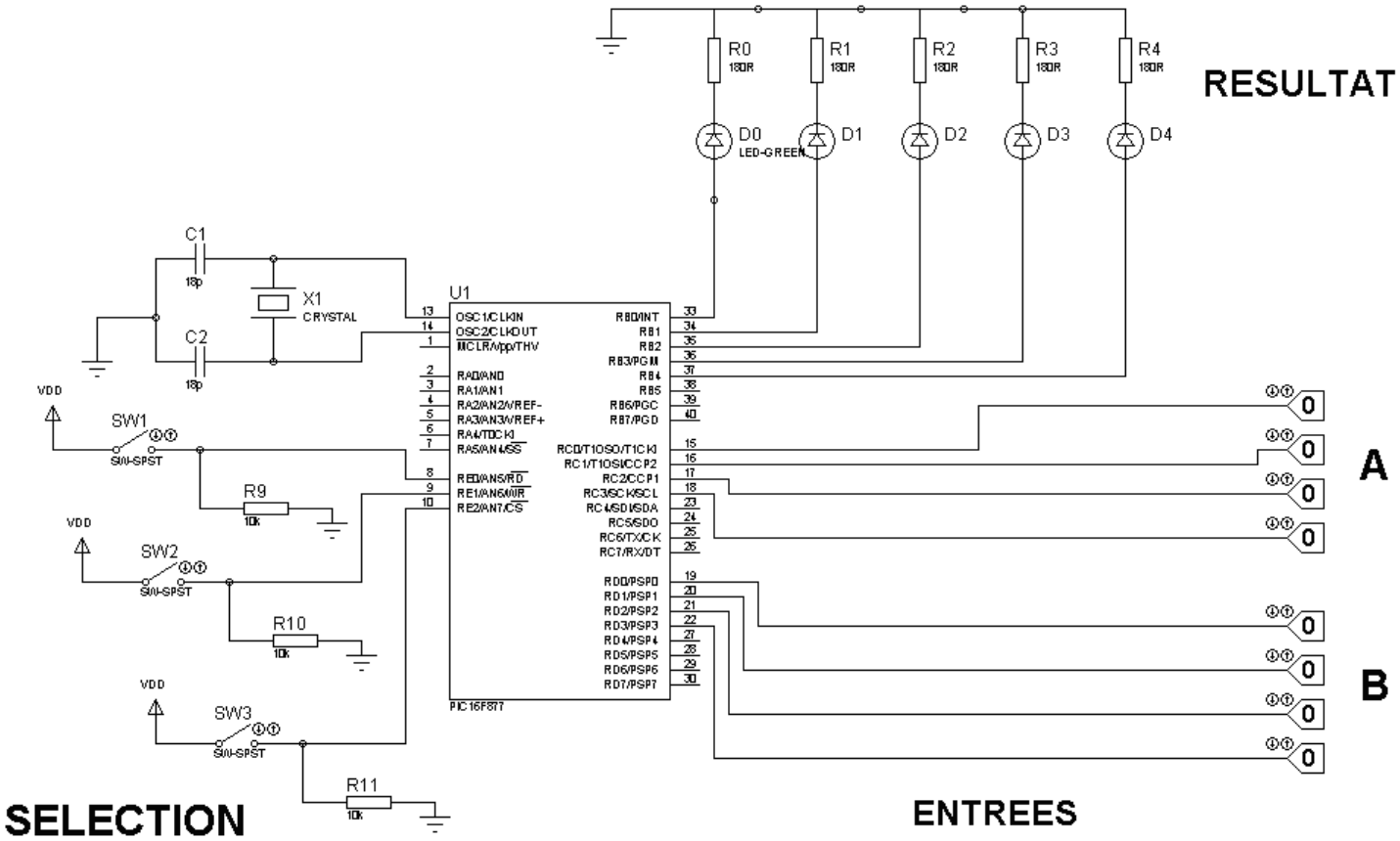


Figure 3 : UAL

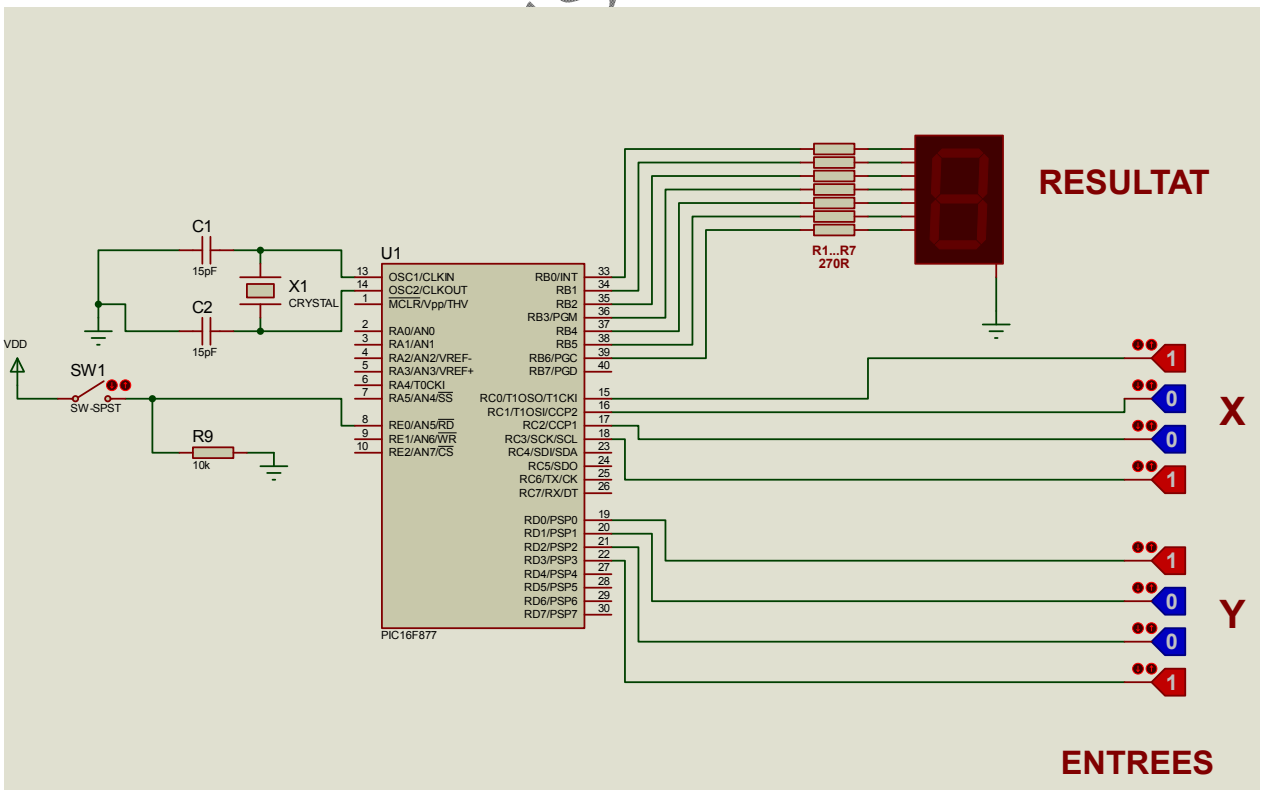


Figure 4 : Comparateur & compteur modulo 10

Exercice 6 : Liaison série et clavier

On propose dans cet exercice d'afficher chaque touche appuyé sur un clavier sur un HyperTerminal. La figure 5 représente l'organigramme de ce programme:

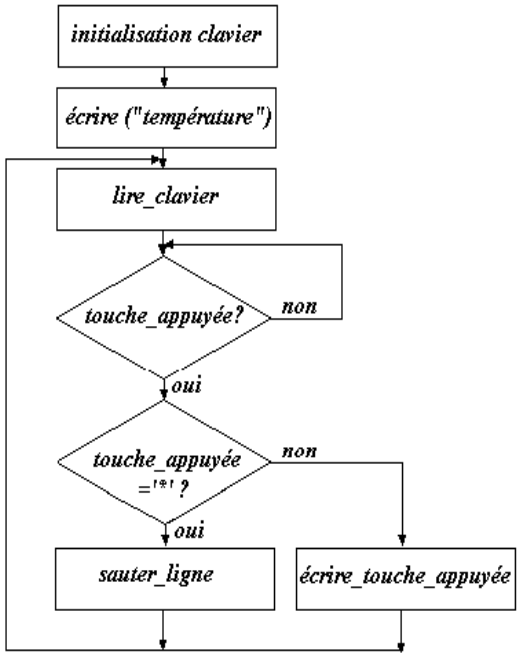


Figure5 : Organigramme du programme test clavier-RS

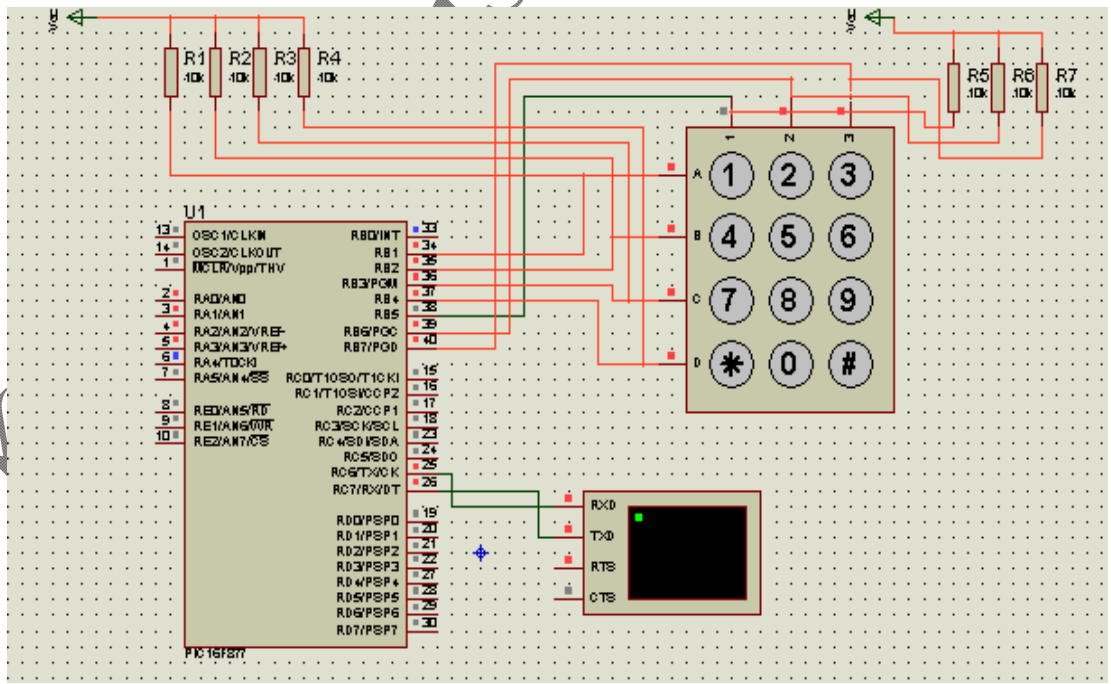


Figure 6 : Schéma du montage

Etablir en premier lieu un programme en C permettant de réaliser le fonctionnement décrit pat la figure 5 et 6.

CHAPITRE 4

LES INTERRUPTIONS

1. Principe

L'interruption est un mécanisme fondamental de tout processeur. Il permet de prendre en compte des événements extérieurs au processeur et de leur associer un traitement spécifique.. Il faut noter que l'exécution d'une instruction n'est jamais interrompue ; c'est à la fin de l'instruction en cours lors de l'arrivée de l'événement que le sous-programme d'interruption est exécuté. A la fin de cette procédure, le microcontrôleur reprend le programme principal à l'endroit ou il l'a laissé.

1. Le Mécanisme général d'une interruption est :
2. Le programme se déroule normalement
3. L'évènement survient
4. Le programme achève l'instruction en cours de traitement
5. Le programme saute à l'adresse de traitement de l'interruption
6. Le programme traite l'interruption
7. Le programme saute à l'instruction qui suit la dernière exécutée dans le programme principal

Les interruptions peuvent être causées par des sources externes ou par des sources internes

- Sources externes:
  - broches parallèles (Exemples: clavier, alarme)
  - ports séries
- Sources internes
  - Timer
  - Convertisseur A-N
  - Reset

2. Source d'interruption dans le PIC 16F877

Le microcontrôleur PIC16F877 dispose 14 sources d'interruptions. Chaque interruption a un bit d'autorisation (Enable) et un bit indicateur (Flag). Les différentes sources d'interruptions sont :

- Timer 0
- Pin RB0
- Ch. RB4/RB7
- Convert. A/D
- Rx USART
- Tx USART
- Port série SSP
- Module CCP1
- Module CCP2
- Timer 1
- Timer 2
- EEPROM
- SSP mode I2C
- Port parallèle

2. 1 Programmer une interruption avec CCS

- **Source de l'interruption**

La source de l'interruption est indiquée au début du sous-programme d'interruption par la directive **#int_XXXX** (XXXX nom de l'interruption) .

```
#INIT_XXXX                //Nom de l'interruption
Void   nom de fonction (Void)
{
    Instruction 1 ;
    Instruction n ;
}
```

Exemple:

#int_ext : Cette directive identifie RB0 comme source de l'interruption : interruption externe

#int_rb : Changement d'état de RB4 à RB7.

#int_TIMER0 : débordement du timer0.

#int_EEPROM : fin d'écriture dans l'EEPROM.

Seules les broches définies en entrée peuvent déclencher une interruption. Ces directives ne discriminent pas quelle entrée est la source, mais on peut utiliser bit_test()

➢ **Validation de l'interruption**

L'interruption ne peut être active que si elle est validée :

enable_interrupts(INT_EXT); *// Valide l'interruption sur RB0*

enable_interrupts(GLOBAL); *// validation globale*

L'interruption doit être validée individuellement (INT_EXT) et globalement.

➢ **Interdire une interruption**

On peut interdire une interruption par l'instruction **disable_interrupt()** Les paramètres sont les mêmes :

disable_interrupts(INT_EXT);

disable_interrupts(GLOBAL);

2. 1 Interruption avec une source externe: RB0

Cette interruption est provoquée par un changement d'état sur l'entrée RB0 du port B quand elle est programmée en entrée.

Exemple 1:

Écrire un programme qui fait clignoter une Led branchée sur RC0 chaque fois que la broche RB0 passe de 1 à 0.

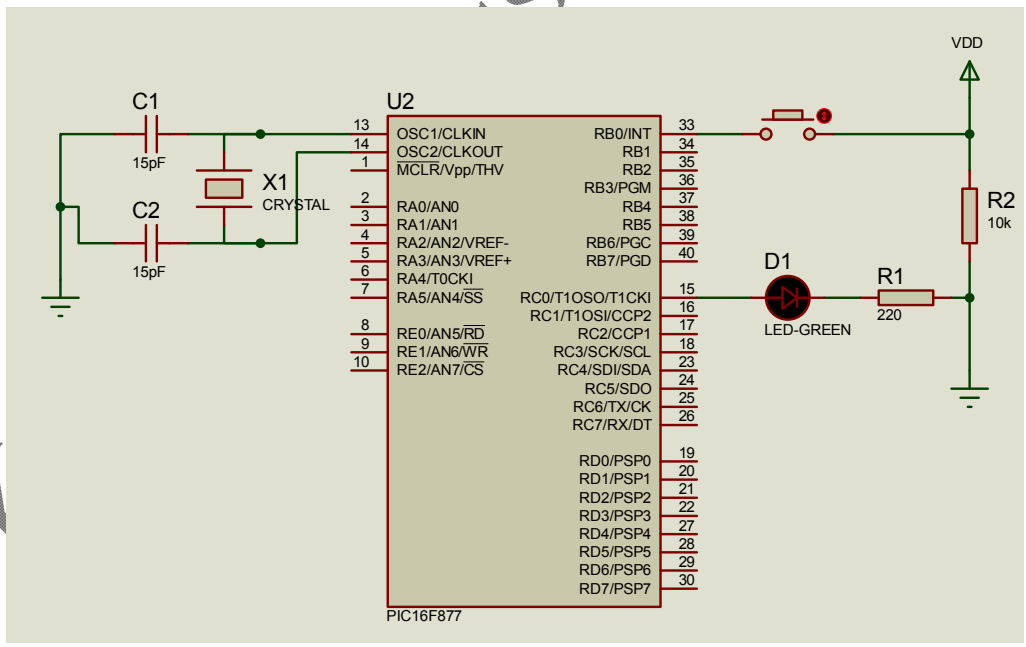


Fig. 4.1 : Montage illustrant l'interruption RB0

```
#include  "16F877.H"
#use delay(clock=20000000)
#define LED  PIN_C0          // Led temoin
```

```
void cligne(int x);              // prototype de la fonction cligne()
//-----------------------------------------------------//
// Sous programme de traitement de l'interruption externe
//-----------------------------------------------------//
#int_ext        // Cette directive indique que la fonction suivante est la tache de l'interruption
rb_ext( )
{
  cligne(3);
}
//-------------------- Programme principal ----------------------
void main(void)
{
 ext_int_edge(H_TO_L);              // Front descendant
 enable_interrupts(INT_EXT);        // Valide l'interruption sur RB0
 enable_interrupts(GLOBAL);         // Valide les interruptions
 set-tris_c(0x00);                  // port c en entrée

 while(1);                          // ce programme ne fait rien
}
//-------------------------------------------------//
//  void cligne(int x)        //  Clignotement de la led verte x fois à intervalle de 1 s.
//------------------------------------------------ //
void cligne(int x)
{
int i;
   for (i=0; i<x; ++i)
    {
      output_low(led);
      delay_ms(1000);
      output_high(led);
      delay_ms(1000);
    }
}
```

2. 2 Interruption BI (RB4 A RB7 du port B)

Cette interruption est provoquée par un changement d'état sur l'une des entrées RB4 à RB7 du port B, Le front n'a pas d'importance.

Exemple 2:

On reprend l'exemple 2 sauf que la Led ne s'allume que s'il y'a un changement sur B4_B7

```
#include  "16F877.H"
#use delay(clock=20000000)
#byte port_b = 6                  // adresse du port B
#define LED  PIN_C0          // Led temoin
void cligne(int x);               // prototype de la fonction cligne()
//-----------------------------------------------------//
// Sous programme de traitement de l'interruption externe
//-----------------------------------------------------//
#int_rb
rb_ext( )
{
  int lecture;
  disable_interrupts(GLOBAL);  // Évite de s'interrompre soi-même
  lecture = port_b & 0xF0;     // isole les 4 bits d'en haut
  if(bit_test(lecture, 4))          // on peut tester les bits pour décider des actions
   cligne(4);
   if(bit_test(lecture, 5))
   cligne(5);
   if(bit_test(lecture, 6))
   cligne(6);
  if(bit_test(lecture, 7))
     cligne(7);
  enable_interrupts(GLOBAL);
}
//-------------------- Programme principal ----------------------
void main(void)
{
 set_tris_b(0xFF);                // port_b en entrée
```

```
 set_tris_c(0x00);              // port_b en entrée

 enable_interrupts(INT_RB);      // Valide l'interruption sur B4-B7
 enable_interrupts(GLOBAL);      // Valide les interruptions
  while(1);                      // ce programme ne fait rien
}
//-----------------------------------------------------//
//  void cligne(int x)            //  Clignotement de la led verte x fois à intervalle de 1s.
//---------------------------------------------------- //
void cligne(int x)
{
int i;
   for (i=0; i<x; ++i)
     {
      output_low(led);

      delay_ms(200);
      output_high(led);
      delay_ms(200);
    }
}
```

2.3 Les autres interruptions

Les autres interruptions seront abordées au moment de l'étude des modules qui les déclenchent.

CHAPITRE 5

LE CONVERTISSEUR ANALOGIQUE NUMERIQUE

1. Description

Ce module est constitué d'un **C**onvertisseur **A**nalogique **N**umérique CAN 10 bits dont l'entrée analogique peut être connectée sur l'une des 8 entrées analogiques externes (RA0-RA5,RE0-RE2). On dit qu'on a un CAN à 8 canaux. Les entrées analogiques doivent être configurées en entrée à l'aide des registres TRISA et/ou TRISE.

Le PIC dispose d'un échantillonneur bloqueur intégré constitué d'un interrupteur S, d'une capacité de maintien C=120 pF et d'un CAN 10 bits. Pendant la conversion, la tension Ve à l'entrée du convertisseur A/N doit être maintenue constante.

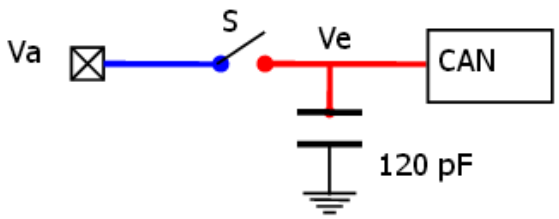


Fig. 5.1 : Constitution du CAN

2. Déroulement d'une conversion

La conversion d'un signal analogique numérique passe par deux phases :

- Echantillonnage blocage (sample and hold). Cette opération consiste à connecter l'entrée à convertir à un condensateur interne, qui va se charger à travers une résistance interne jusqu'à la tension appliquée : c'est le temps d'acquisition.
- Une fois le condensateur est chargé, déconnecté la tension appliquée en ouvrant l'interrupteur S, pour procéder à la phase ce conversion.

Pour résumer, les étapes de conversion sont :

- Configurer la conversion A/N
- Attendre TACQ
- Lancer la conversion
- Attendre conversion finie
- Lire la valeur convertie

➢ Temps d'acquisition

$$Tacq = Tamp + Tc + Tcoff$$

Tamp : temps de réaction des circuits

Tc : temps de charge du condensateur

Tcoff : temps qui dépend du coefficient de température.

Le temps de réaction est fixé à 2µs ; de même Tcoff est limité à une valeur maximale de 1,25µs.

Dans les conditions normale et pour une tension de 5V, Tc = 16,47µs.

Le temps d'acquisition Tacq = 19,72µs |20µs

➢ Temps de conversion

Le temps de conversion est égal à 12 T_{AD}

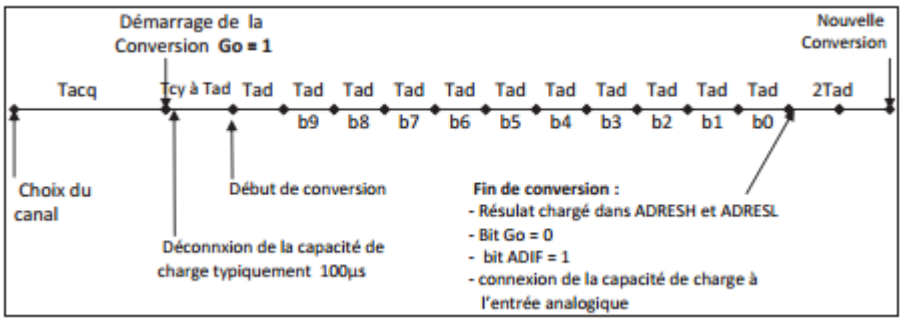


Fig. 5.2 : Cycle du CAN

T_{AD} est le temps de conversion d'un bit, il dépend de la fréquence du quartz et du prédiviseur (div) choisi :

$$T_{AD} = div \times 1/f_{osc}.$$

Le choix de div doit être ajusté pour que T_{AD} soit ≥ à 1,6 µs

Avec un quartz de 4 MHz, il faut choisir div=8 ce qui donne T_{AD}= 2 µs soit un temps de conversion : T_{CONV}= 24 µs

➢ Valeur numérique obtenue

La valeur converti est calculé selon:

N = valeur entière de (Va – Vref-) / Q

Avec :

- **Va:** tension analogique à numériser.
- **N :** valeur numérique sur N bits.
- **Q = pas de quantification = (Vref+- Vref-)/1024**

Exemple :

Vref+ = Vdd = 5V, Vref- = 0, Vin = 4 V

Q = 5V/1024 = 0,0048828125 V

N = 4V / 0,0048828125 = 819

3. Configuration du convertisseur

Avant de réaliser une conversion, il faut définir:

- L'horloge
- Le nombre d'entrées analogiques
- Le type de tension de référence (interne ou externe)

➢ **Choix de l'horloge**

Syntaxe : *setup_adc(XXX); // Utilisation de l'horloge interne(4 MHz)*

XXX peut être :

- *ADC_OFF* : CAN désactivé
- *ADC_CLOCK_INTERNAL* : Utilisation de l'horloge interne
- *ADC_CLOCK_DIV_2 :* Utilisation de l'horloge interne divisée par 2
- *ADC_CLOCK_DIV_8 :* Utilisation de l'horloge interne divisée par 8
- *ADC_CLOCK_DIV_32:* Utilisation de l'horloge interne divisée par 32

➢ **Choix des broches en entrée analogique**

Syntaxe : *setup_adc_ports(XXX);*

XXX peut être :

- *NO_ANALOGS* : Tout le port A et E est en mode numérique
- *ALL_ANALOG* : Tout le port A et E est en mode analogique
- *RA0_ANALOG* : Entrée RA0 est en mode analogique
- *RA0_RA1_RA3_ANALOG;* Entrée RA0, RA1 et RA3 est en mode analogique

➢ **Choix des tensions de référence**

Les tensions de références permettant de fixer la dynamique du convertisseur. Elles peuvent être choisies parmi V_{dd}, V_{ss}, V_{ref+} ou V_{ref-} .

Les tensions de référence haute et basse peuvent être choisies par programmation parmi: V_{DD} ou la broche RA3 pour V_{ref+} et V_{SS} ou la broche RA2 pour V_{ref-}

Syntaxe : *setup_adc_ports(XXX);*

XXX peut être :

- *AN0_AN1_VREF_VREF:* RA0 et RA1 en mode analogique, V_{ref+} = RA3 et V_{ref-} = RA2
- *AN0_AN1_VSS_VREF* : RA0 et RA1 en mode analogique et V_{ref+} = RA3
- *AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF :* RA0, RA1, RA4-RA7 en mode analogique, V_{ref+} = RA3 ET V_{ref-} = RA2.

4. Exercices d'application : multimètre

La plupart des capteurs délivrent une tension proportionnelle au phénomène qu'ils mesurent. Pour être exploitable cette tension doit être numérisée à l'aide d'un CAN.

Dans cet exemple, nous allons numériser la tension présente sur l'entrée AN0 et afficher le résultat de cette conversion par la voie série vers le PC et via un afficheur LCD selon la figure 5.3 :

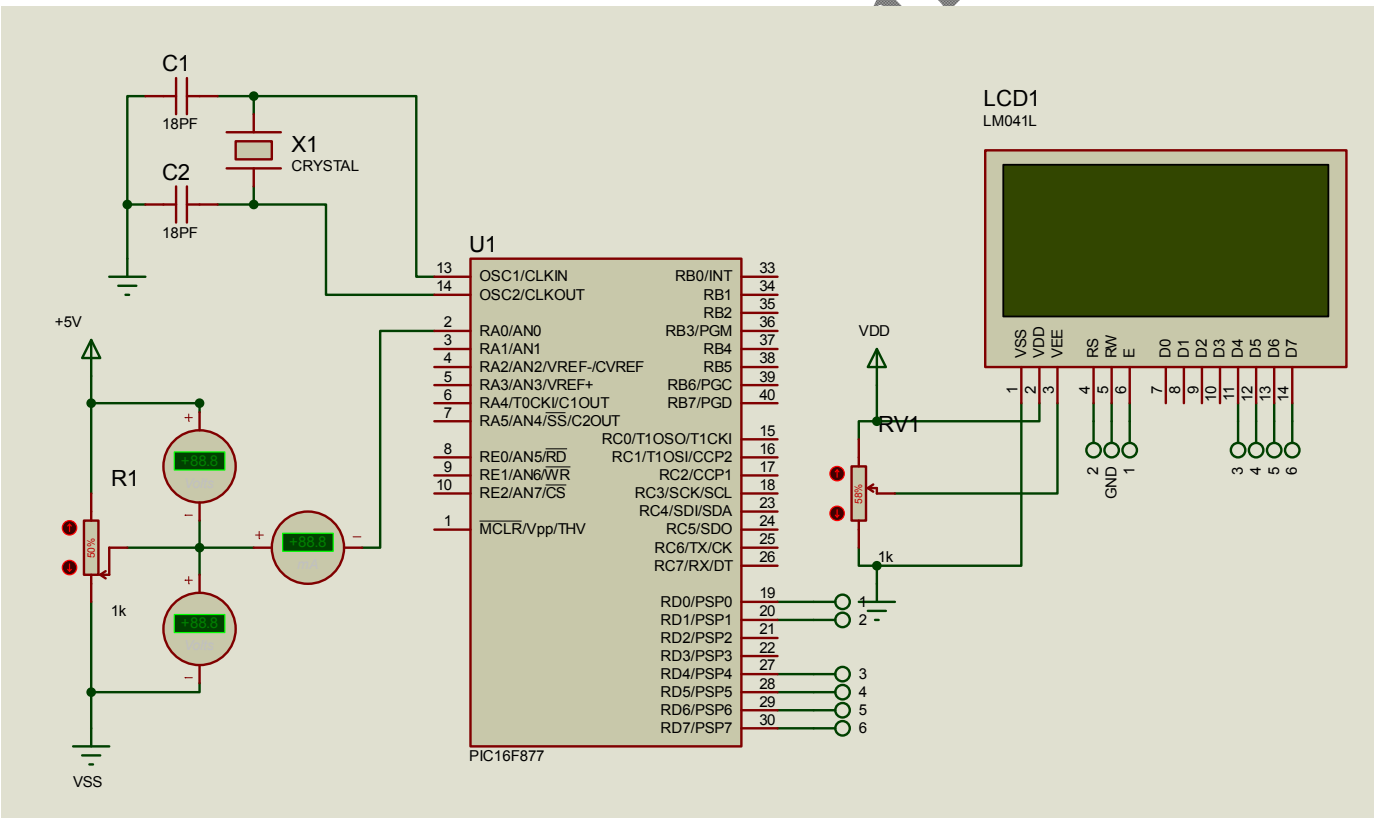


Fig. 5.3 : Application du CAN

```
#include <16F877.h>
#uses delay (clock=4000000)
#include <LCD.C>          // appel du fichier contenant le driver du LCD
```

```
#device ADC=8                // resolution à 8 bits

void main()
{
  float tension,quantum;   //

  setup_adc_ports(RA0_ANALOG);              //RA0 configurée en entrée analogique
  setup_adc(ADC_CLOCK_INTERNAL);    // Utilisation de  l'horloge interne 4Mhz
  set_adc_channel(0);                       // lit sur RA0
  delay_ms(100);

  lcd_init();                               // initialisation du LCD

  lcd_putc("\f*** CAN ***\n");
  delay_ms(1000);
  lcd_putc('\f');

  quantum = 5.0/255.0;                      // calcul du pas de quantification
  while (true)
  {
  tension = read_adc()*quantum;             // lecture de la tension et conversion en numérique
  printf(lcd_putc,"\f Tension: %2.2f v \n",tension);
  printf("\f Tension = %2.2f v\n",tension);
  delay_ms(1000);

  }
}
```

CHAPITRE 6

LES TIMERS

1. Présentation du Timer

Les Timers/compteurs sont des périphériques de gestion de temps. Ils permettent des réaliser les fonctions suivantes :

- comptage des évènements
- synchronisation des signaux
- fixer le débit d'une liaison série synchrone ou asynchrone
- génération des événements périodiques (échantillonnage des signaux analogiques, rafraichissement des afficheurs multiplexés …)
- génération des signaux périodiques (carré, MLI …)
- mesure de temps…

2. Fonctionnement du Timer

Les timers sont des compteurs formés généralement d'un pré-diviseur suivi d'un registre compteur de 8 ou 16 bits. L'entrée d'horloge peut être interne (mode timer) ou externe (mode compteur d'événements). Lorsque le registre compteur atteint sa valeur maximale et repasse à 0, un bit indicateur (flag) sera positionné et une interruption pourra être générée, informant ainsi la CPU du débordement du timer. Il faut bien noter que le programmeur devra remettre à zéro cet indicateur après chaque débordement.

Le microcontrôleur PIC16F877 dispose de trois timers appelés Timer0, Timer1 et Timer2

2.1 Timer 0

C'est un compteur 8 bits qui peut compter (de 0 à 255) :

- soit les impulsions de l'horloge via un prédiviseur : Mode Timer
- soit des impulsions externes, via la broche RA4 : Mode compteur

- **Mode Timer**

Timer 0 est incrémenté à chaque cycle instruction (Fosc/4), en considérant le prédiviseur avec un rapport de 1.

- **Mode Compteur**

Timer 0 est alors incrémenté à chaque front montant ou descendant sur la broche RA4.

- **Prediviseur**

Il est formé d'un pré-diviseur programmable (Programmable Prescaler) suivi d'un registre compteur 8 bits (TMR0). Un prescaler ou prédiviseur permet de diviser la fréquence de comptage.

Il est partagé entre le Watchdog et TMR0.

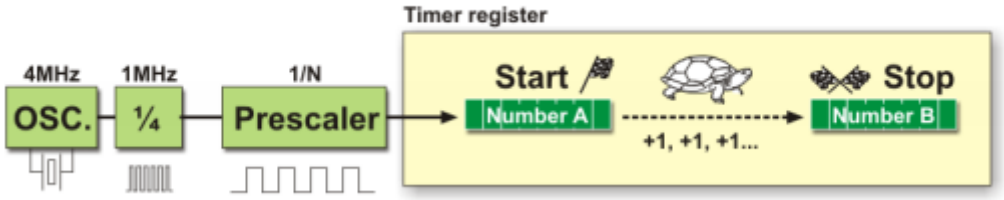


Fig. 6.1 : Structure d'un prédiviseur

2.2 Timer 1

C'est un compteur 16 bits qui peut compter (de 0 à 65535) :

- soit les impulsions de l'horloge
- soit les impulsions externes, et en particulier les impulsions d'un quartz externe.

- **Mode Timer**

Timer 1 est incrémenté à chaque cycle instruction (Fosc/4), en considérant le prédiviseur avec un rapport de 1, 2, 4, ou 8

- **Mode Compteur**

Timer 1 s'incrémente à chaque front montant de l'horloge externe appliquée sur le RC0

L'horloge externe peut également être l'oscillateur interne, dont la fréquence est fixée par un quartz externe branché entre la broche RC0 et la broche RC1.

Il peut être incrémenté en mode veille (Sleep), via l'horloge externe,

2.3 Timer 2

C'est un timer couplé au module dit CCP. Il est utilisé essentiellement pour la génération d'impulsions à période ajustable (PWM).

Le timer 2 comporte un registre compteur 8 bits (TMR2) avec un prédivideur et un postdiviseur. Ce timer admet uniquement une horloge interne (Fosc/4). Le prédiviseur peut être paramétré par l'une de trois valeurs: 1, 4 ou 16 ; tant disque le postdiviseur permet des divisions de 1 à 16 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ou 16.

2.4 Configuration des Timers en CCS

Pour configurer le Timer 0 : ***setup_timer_a(XXX|YYY);***

a= 0,1 ou 2 selon le Timer à utiliser

XXX peut être :

- RTCC_INTERNAL : Horloge intene
- RTCC_EXT_L_TO_H : Horloge externe sur front montant
- RTCC_EXT_H_TO_L : Horloge externe sur front descendant

YYY peut être :

- RTCC_DIV_1
- RTCC_DIV_2
- RTCC_DIV_4
- RTCC_DIV_8
- RTCC_DIV_16
- RTCC_DIV_32
- RTCC_DIV_64
- RTCC_DIV_128
- RTCC_DIV_256

Pour lire la valeur du Timer 0 : *value=**get_timer0();***

Pour régler la valeur du Timer 0 : ***set_timer0(valeur);***

3. Exercice d'application : Interruption du Timer 1

Dans cette application, on se propose de compter les secondes en faisant appel à l'inetrruption du Timer 1 et d'afficher le résultat sur un hyperTerminal

```
#include <16F877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)


#define INTS_PAR_SECONDE 19        // (20000000/(4*4*65536))

int8 secondes;                     // Compteur pour secondes
int8 int_count;                    // Nombres d'interruptions survenues pendant une seconde
```

```
#INT_TIMER1                          // Interruption Timer 1
void clock_isr()                     // débordement timer1 (65535->0), qui est environ 19 fois en
{                                    // une seconde pour ce programme
if(--int_count==0) {
    ++secondes;
    int_count = INTS_PAR_SECONDE;
   }
}

void main() {
  int_count = INTS_PAR_SECONDE;
  setup_timer_1(T1_INTERNAL | T1_DIV_BY_4);  // Démarrer le Timer 1
  set_timer1(0);
  enable_interrupts(INT_TIMER1);
  enable_interrupts(GLOBAL);

  while(TRUE) {
    printf("Appuyer sur une touche pour commencer.\n\r");
    getc();
    int_count = INTS_PAR_SECONDE;
    secondes = 0;
    printf("Appuyer sur une touche pour arrêter.\n\r");
    getc();
    printf("%u Secondes.\n\r", secondes);
  }
}
```

Direction **G**énérale des **E**tudes **T**echnologiques, **I**nstitut **S**upérieur des **E**tudes **T**echnologiques de **N**abeul

Examen

Microprocesseurs et Microcontrôleurs

Filière : 2ème Année Licence Appliquée en Génie Electrique – Année universitaire 2013-2014 – Semestre 2
Classes : AII2
Durée : 1H30
Nombre de pages : 02
Documents : Non autorisés
Enseignants : Mme Yosra RKHISSI KAMMOUN et M. Nizar TOUJENI

Exercice 1 : (5 points)

A la présence d'une carte magnétique (**p**), une barrière automatique se lève jusqu'au contact (**b**), elle reste levée pendant 10s ensuite elle s'abaisse pour retoucher le contact (**a**). Les mouvements de la barrière sont possibles grâce à un moteur à 2 sens, commandé par deux contacteurs. Ce cycle est décrit par le GRAFCET de point de vue PC suivant :

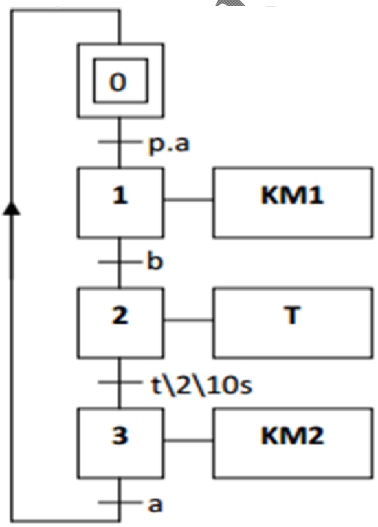


Figure 1

1. Donner un schéma de la carte de commande à base d'un microcontrôleur de votre choix.
2. Ecrire par la suite le programme en C à implanter dans le microcontrôleur.

Exercice 2 : (7,5 points)

Un codeur optique à dix pas est installé sur l'arbre d'une machine tournante (M) afin de détecter la position angulaire. Le code binaire de la position est fourni sur quatre bits (a3, a2,

a1, a0). On cherche à mettre en œuvre un système de transcodage fournissant la position angulaire en code **binaire naturel** sur les sorties (S3, S2, S1 et S0).

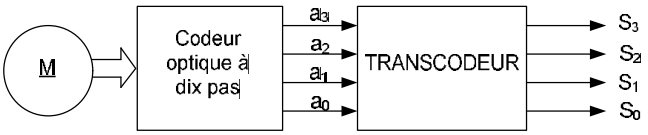


Figure 2

Le tableau suivant donne le code binaire de la position du disque incrémental :

Pas	a3	a2	a1	a0
0	0	0	1	0
1	0	1	1	0
2	0	1	1	1
3	0	1	0	1
4	0	1	0	0
5	1	1	0	0
6	1	1	0	1
7	1	1	1	1
8	1	1	1	0
9	1	0	1	0

1. Dresser la table de vérité donnant les sorties (S3 S2 S1 S0) en fonction des sorties du codeur optique (a3, a2, a1, a0).

2. Etablir un programme en C permettant de réaliser un codeur optique à dix pas à base d'un PIC 16F877 en respectant la table des mnémoniques suivante :

Désignations	Les entrées /Sorties
a3, a2, a1, a0	PORTD.3, PORTD.2, PORTD.1, PORTD.0
S3, S2, S1, S0	PORTB.3, PORTB.2, PORTB.1, PORTB.0

Exercice 3 : (7.5 points)

On veut réaliser un système de comptage selon les valeurs de deux tensions analogiques VA et VB. L'affichage des résultats se fera sur des afficheurs 7 segments comme indiqué sur la figure ci-dessous:

Etablir un programme en C permettant de réaliser les opérations suivantes :

- Comptage modulo 20 si VA < VB.
- Décomptage modulo 20 si VA ≥ VB.

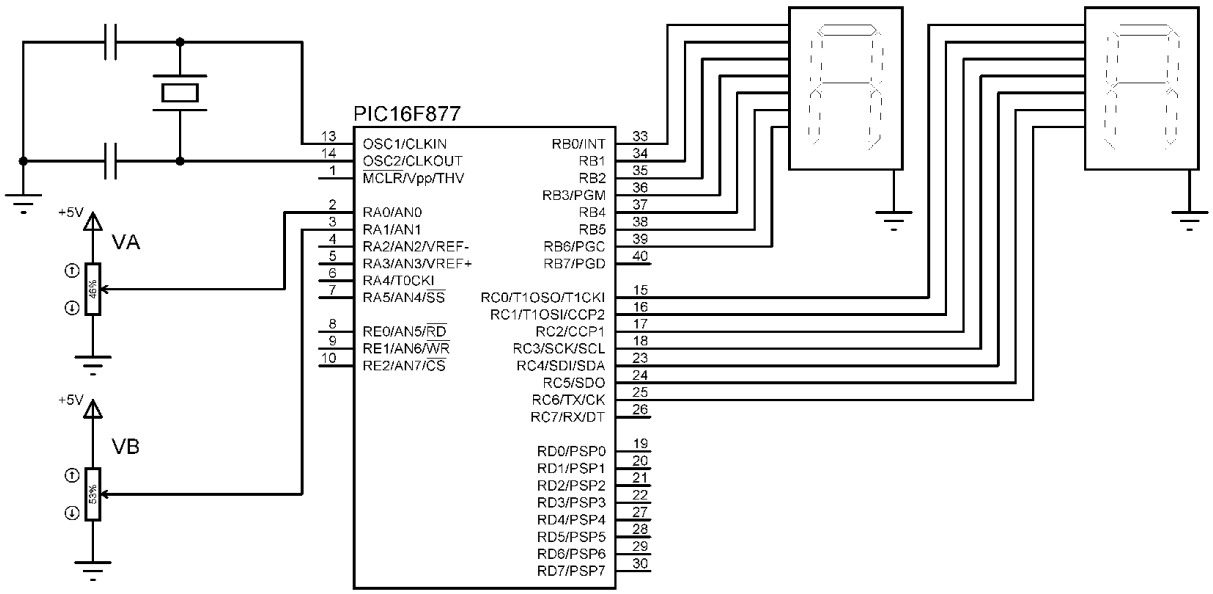


Figure 3

BIBLIOGRAPHIE

[1] J-M Bernard, J Hugon, De la *logique câblée* aux *microprocesseurs* Tome 3: *Méthodes* de *conception* de *systèmes*. *Edité* par *Eyrolles* (1984)

[2] R. Zakset A.Wolfe. Du composant au système – Introduction aux microprocesseurs. Sybex, Paris, 1988.

[3] C. Tavernier, Les microcontrôleurs PIC Recueil d'applications. Dunod , Paris, 2005.

[4] C. Tavernier, Programmation en C des PIC. Dunod , Paris, 2005

[5] C. Tavernier, Les microcontrôleurs PIC Description et mise en œuvre. Dunod, Paris, 2000

[6] C. Tavernier, Les microcontrôleurs PIC 10, 12, 16 Description et mise en œuvre. Dunod , Paris, 2007

[7] B.Van Dam, 50 nouvelles applications des microcontrôleurs PIC. Elector, 2010

[8] P. Letenneur, Langage C (CCS Info pour les PICs), 2003

WEBORGRAPHIE

[1] BIGONOFF (2002). La gomme MID-RANGE par l'étude des16F87X (16F876-16F877), http://www.abcelectronique.com/bigonoff/

[2] Datasheet PIC16F87X , "1999 Microchip Technology Inc. DS30292B. http://www.microchip.com

[3] Compilateur C pour PIC (PCWH). Custom Computer Services, inc. http://www.ccsinfo.com/