N° d'ordre :

N° de série :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE D'EL-OUED
FACULTEDES SCIENCES ET TECHNOLOGIE
Département D'électronique

Mémoire de fin d'étude présenté
Pour l'obtention du diplôme de

Licence ACADEMIQUE

Domaine : **Sciences et techniques**
Filière :**Electronique**
Spécialité :**Télécommunications**

Présenté par :**AMARA SAYAD**

LEKHOUA SALEM

Étude d'un microcontrôleur 16F84

Déposé le 01-06- 2014

Au niveaudu jury composé de :

M.	**HIMA Abd Elkader**	MA	Président
M.	**HETTIRI Messaoud**	MA	Examinateur
M.	**GHENDIR SAID**	MA	Directeur du mémoire

20**13**-20**14**

N° d'ordre :

N° de série :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE D'EL-OUED
FACULTEDES SCIENCES ET TECHNOLOGIE
Département D'électronique

Mémoire de fin d'étude présenté
Pour l'obtention du diplôme de

Licence ACADEMIQUE

Domaine : **Sciences et techniques**
Filière :**Electronique**
Spécialité :**Télécommunications**

Présenté par :**AMARA SAYAD**

LEKHOUA SALEM

Étude d'un microcontrôleur 16F84

Déposé le 01-06- 2014

Au niveaudu jury composé de :

M.	**HIMA Abd Elkader**	MA	Président
M.	**HETTIRI Messaoud**	MA	Examinateur
M.	**GHENDIR SAID**	MA	Directeur du mémoire

20**13**-20**14**

Dédicaces

Je dédie Ce Travail…

✓ *À MES CHERS PARENTS*

Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que vous avez consenti pour mon instruction et mon bien être.
toujours.

✓ *A MES CHERS ET ADORABLE FRERES ET SOEURS*

A mes amis :

Houdaifa Hoggie, Mohamed Moussaoui, Rabie Gar El matred, Oualide Didi, Abd Eladim Bekkouch , Mouize Mehri, Dhia Merad, Abd Latif Mansouri, habita abd allah, Ghanem abd Elouahed.

Remerciment

En premier lieu ,nous remercions DIEU tout puissant qui nous a donné la force et la patience pour accomplir notre travail.

Nous tenons aussi à remercie :

- *Notre professeur encadreur Mr Ghendir Said pour le temps qu'il nous a consacré et pour ses conseils et orientations qui nous ont été d'un grand secours dans sa direction de notre mémoire .*
- *Le président et les membres du jury pour leur temps précieux consacré à la lecture et à l'évaluation de ce mémoire. - Les responsables et tout le personnel du département de l' électronique pour les facilités accordées pour terminer ce travail .*
- *Tous nos professeurs enseignants qui ont contribué à notre formation durant notre notre cursus . A tous ceux qui, nous ont aidés ou encouragés de loin ou de prés à l'élaboration de ce mémoire..*

SOMMAIRE

Sommaire

SOMMAIRE

CHAPITRE II : Programmation de Microcontrôleur 16F84

SOMMAIRE

CHAPITRE III: Projet de Programmation des Feux de circulation

SOMMAIRE

SOMMAIRE

Sommaire

SOMMAIRE

CHAPITRE II : Programmation de Microcontrôleur 16F84

SOMMAIRE

CHAPITRE III: Projet de Programmation des Feux de circulation

SOMMAIRE

Liste de Figure

Liste de Figure

Introduction générale

INTRODUCTION GENERALE

Depuis quelques années, les technologies d'intégration ont eu suffisamment progressé pour permettre la fabrication d'un système auquel, dans un seul boîtier appelé ‘‘microprocesseur’’ (µp) ou ‘‘microcontrôleur’’ (µc).

Cette progression a permis de stocker des centaines de milliers de transistors dans une puce en vertu de semi-conducteur. Cette technique est utilisée aux automatismes industriels, en l'interfaçant avec les différents capteurs et pré-actionneurs d'un processus à commander.

La découverte du microcontrôleur date aujourd'hui de près de quarante ans, en effet la fabrication du premier circuit commence en 1970 , où la société INTEL met au point le premier microprocesseur le 4004. On n'imagine pas à l' époque que cette révolution industrielle donnera naissance à l'ordinateur individuel. Depuis , leur puissance de calcul et l'intégration des transistors les constituant n'ont cessé d'évoluer. On retrouve désormais les microcontrôleurs dans la plupart des applications, comme contrôleur des processus industriels, ou bien encore pour commander les machine électroménagers.

Les microcontrôleurs ne sont jamais employés seuls, des circuits périphériques leur sont toujours associés pour pouvoir être intégrés au sein d'une application. Celui-ci a l'avantage d'être plus intelligent. On est donc passé de la logique câblée qui intègre des dizaines, voir même des centaines de circuits logiques à la logique programmée qui n'utilise qu'un seul élément.

Le sujet de cette étude se base sur le microcontrôleur de type pic 16F84, qui joue un rôle majeur dans le monde industriel, pour cette raison notre objectif dans ce modeste travail, est d'étudier ce dernier en se focalisant la lumière au premier chapitre sur l'architecture interne et externe des microcontrôleurs pic 16F84.

Au deuxième chapitre, nous essayons de donner une idée sur la programmation en Assembleur en se basant sur l'architecture générale du microcontrôleurs pic 16F84.

Enfin, nous allons voir au dernier chapitre un exemple d'un projet d'application révélé du terrain qui est une application de feux de circulation à base d'un µc 16F84. Pour cela, on a adopté dans ce travail tous les outils nécessaires à la mise en œuvre de cette application.

Chapitre

1

ARCHITECTURE DE MICROCONTROLEUR 16F84

I- ARCHITECTURE DU MICROCONTROLEUR 16F84

I.1 Introduction :

En fait, la programmation en assembleur dépend directement de l'architecture des microcontrôleurs. Pour cette raison, avant d'aller à la partie programmation de µc pic 16F84, il faudrait avoir une idée sur l'architecture du microcontrôleur pour que on puise faire adapter la programmation avec le type et la famille du pic concerné.

I.2 Qu'est-ce qu'un PIC 16F84 :

C'est une unité de traitement de l'information, la famille de pic est Mid-Range (qui utilise de mots d'instruction de 14 bit) ont un jeu de 35 instructions, stockent dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en 1 cycle.

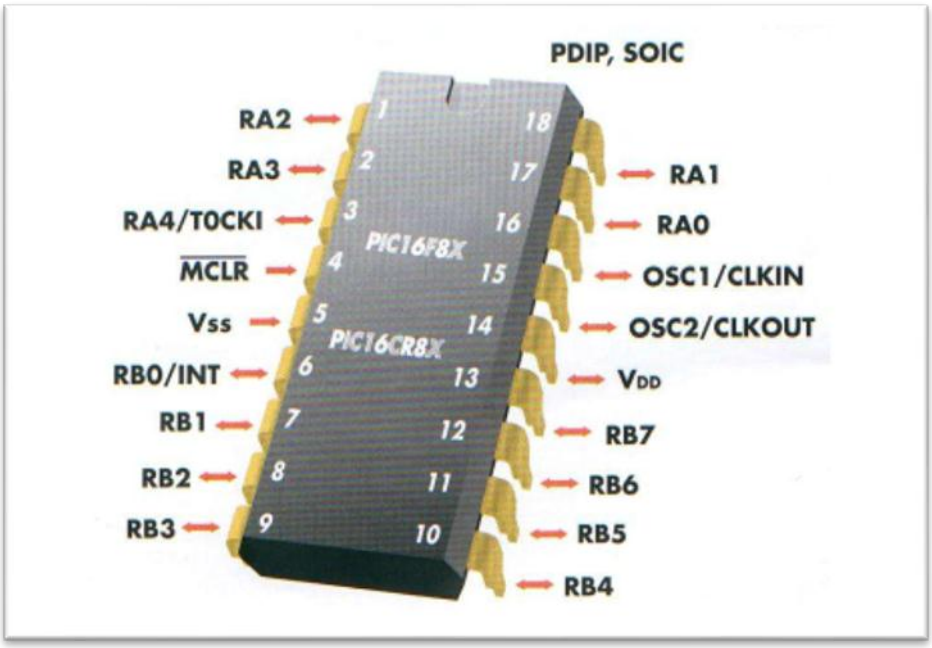
On atteint donc des très grandes vitesses, et les instructions sont de plus très rapidement assimilées. L'exécution en un seul cycle est typique des composants RISC (Reduced Instruction Set Computer).

Le PIC 16F84 est un microcontrôleur 8 bits. Il dispose donc d'un bus de données de huit bits Puisqu'il traite des données de huit bits, il dispose d'une mémoire de donnée dans Laquelle chaque emplacement (défini par une adresse) possède huit cases pouvant contenir chacune un bit.

L'horloge fournie au PIC® est pré divisée par 4au niveau de celui-ci. C'est cette base de temps qui donne la durée d'un cycle. Sion utilise par exemple un quartz de 4MHz, on obtient donc 1000000 de cycles/seconde**. [1]**

I.3 Architecture Externe du PIC 16F84 :

Ce microcontrôleur se présente sous la forme d'un boîtier DIL à 18 broches comme schématisé figure (I.1).



Figure(I.1): Brochage du microcontrôleur PIC 16F84

RA0 à RA4 / TOCKI sont les pattes d'entrées / sorties du port A.

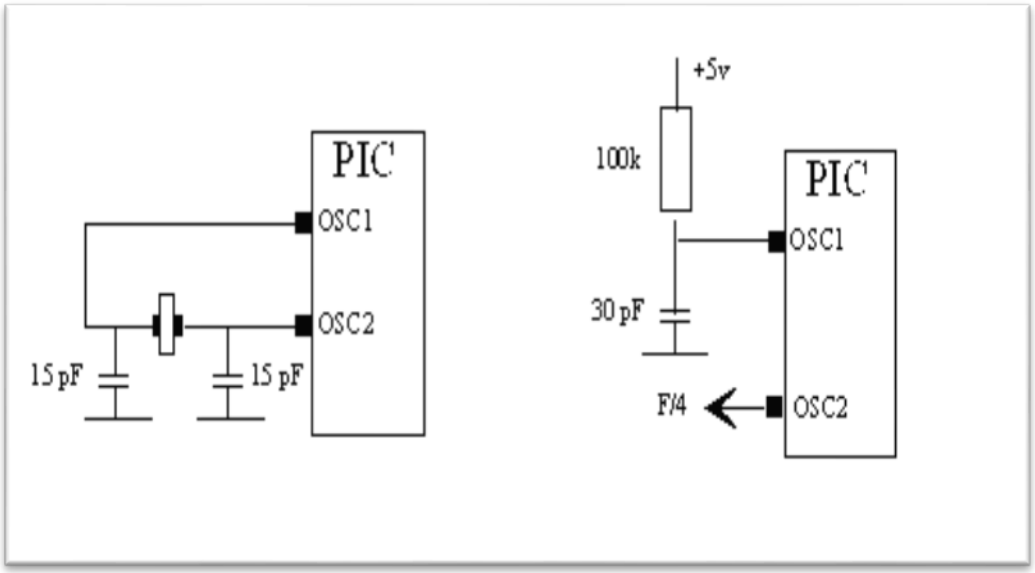
RA4 / TOCKI est commune avec l'entrées d'horloge externe du timer 0.

RB0 / INT à RB7 sont les pattes d'entrées / sorties du port B

RB0 / INT est commune avec l'entrée d'interruption externe.

Individuellement, chaque broche des ports A et B ne peut débiter plus de 20 mA ou absorber plus de 20 mA. Le total des intensités débitées par le port A ne peut dépasser 50 mA et pour le port B, 100 mA. Le total des intensités absorbées par le port A ne peut dépasser 80 mA et pour le port B, 150 mA.

OSC1 /CLOCKIN et OSC2 / CLOCKOUT sont les pattes d'horloges. Plusieurs types d'horloges peuvent être utilisés : externe, à quartz ou à circuit RC. La figure (I.2) montre les schémas de câblage en version RC et quartz. L'unité centrale de nos montages sera équipée d'un oscillateur à quartz car cette solution présente une meilleure précision que l'oscillateur RC.

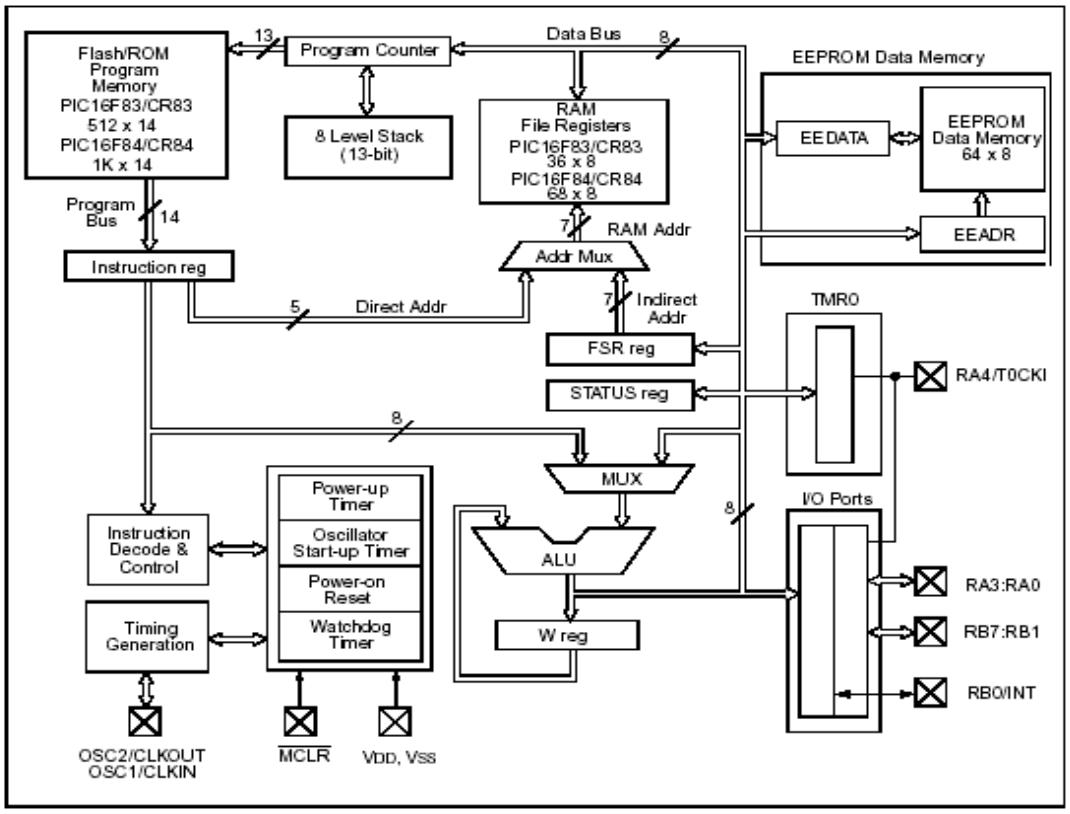


Figure(I.2): Horloges à quartz et à circuit RC

MCLR/VPP est la patte de Reset et d'entrée de la tension de programmation. Les circuits PIC intégrant en interne une circuiterie de Reset automatique à la mise sous tension, cette broche doit être reliée à la VDD en utilisation normale.

V_{SS} et V_{DD} sont les pattes d'alimentation. V_{DD} doit être compris entre 2 et 6 V en utilisation. Lors de la programmation, V_{DD} doit être comprise entre 4,5 V et 5,5 V et V_{SS}comprise entre 12 V et 14 V.

I.4 Architecture Interne du PIC 16F84 :



Figure(I.3): l'architecture interne du pic 16F84

I.4.1 Mémoire Programme et PC :

En haut à gauche de la figure (I.3) se trouve le bloc où est stocké votre programme cette mémoire contient toutes les « instructions » que doit effectuer le microcontrôleur.

Pour le PIC 16F84, chaque instruction est codée sur 14 bits et la taille mémoire est de 1 Ko x 14 bits (soit 1024 instructions à la file ; 1 K=1024 en informatique !).

Cette taille reste suffisante pour les petites applications qui nous intéressent. Le Program Counter ou «PC» (à droite de la mémoire programme sur la figure(I.3)) est un registre qui pointe l'instruction à exécuter et permet donc au programmeur de ce déplacer à volonté dans cette mémoire. A la mise sous tension, ce registre est mis à 0 et pointe donc la première case mémoire. Durant l'exécution du programme, chaque instruction pointée transite dans le « registre d'instruction » afin qu'elle puisse être traitée par le « contrôleur et décodeur d'instructions » (ces deux blocs se trouvent à gauche dans la figure(I.3)). C'est ce

dernier qui gère la suite d'actions internes nécessaires à la bonne exécution de cette instruction.

La mémoire programme peut être de différente nature selon le microcontrôleur utilisé.

Dans notre cas, la mémoire est de type FLASH/ROM, c'est-à-dire réinscriptible à volonté. Mais il existe aussi des mémoires EPROM qui sont effaçables par UV ainsi que des mémoires OTP qui ne peuvent être programmées qu'une seule fois. [2]

I.4.2 Unité de Calcul- ALU :

En bas et au centre, se trouve le cœur du système appelé «ALU» (Unité Arithmétique et Logique). C'est cette partie qui effectue physiquement toutes les actions internes dictées par le Contrôleur de Décodeur d'Instructions. Par exemple, l'ALU peut effectuer une addition, une soustraction ainsi que les opérations logiques telles que « ET » , «OU » , etc.

Pour effectuer toutes ces opérations, l'ALU utilise les données en provenance d'un registre de travail appelé «W» (Work Register). Vous découvrirez que ce registre est largement utilisé par les instructions du programme.

I.4.3 Mémoire RAM :

Pour fonctionner, un programme doit généralement pouvoir stocker temporairement des données. Une zone est spécialement prévue à cet effet : c'est la RAM (Random Access Memory). Contrairement à la Mémoire Programme, cette dernière s'efface lorsque l'on coupe l'alimentation. Vous trouverez ce bloc en haut, au centre de la figure(I.1). On s'aperçoit que la taille de cette mémoire est de 68 x 8 bits.

Comme chaque donnée est codée sur 8 bits, il est donc possible de mémoriser 68 données à la file. De plus, la mémoire RAM contient deux autres zones(appelées Bank 0 et Bank) réservées aux registres de configuration du système. Nous détaillerons la fonction de ces registres à la fin de cet article.

I.4.4 Mémoire de Données EEPROM :

Une particularité (et aussi un grand avantage) du PIC16F84 est de posséder une mémoire de 64 x 8 bits où l'on peut stocker des données qui ne disparaissent pas lors d'une coupure d'alimentation : c'est la mémoire EEPROM (en haut à droite de la figure). Généralement cette zone sert à mémoriser des paramètres d'étalonnage ou de configuration. Mais attention, elle possède un inconvénient : le temps d'accès est relativement long. Elle ne convient donc pas pour une utilisation qui demande de la rapidité (calcul asservissement, etc.).

I.4.5 Le Status Register ou Registre d'état :

Ce registre donne plusieurs indications: le résultat d'une opération effectuée par l'ALU (résultat égal zéro par exemple), l'état de l'initialisation du microcontrôleur (reset), etc. Il permet aussi de définir la zone d'accès en mémoire RAM Bank 0 et Bank 1 pour accéder aux registres de configuration (pas d'inquiétude nous verrons cela plus tard).

I.4.6 Le bloc nommé « 8 Level Stack » ou « Pile » :

Il est utilisé par le microcontrôleur pour gérer le retour des sous programmes et des interruptions.

I.4.7 Le bloc « TMR0»: Il sert au fonctionnement du TIMER

I.4. 8 Le bloc I/O ports: Il permet d'écrire ou de lire sur les ports A et B.

I.4.9 Le bloc Timing Génération:

Associé au bloc présent juste sur sa droite gère tous les signaux d'horloges du système.

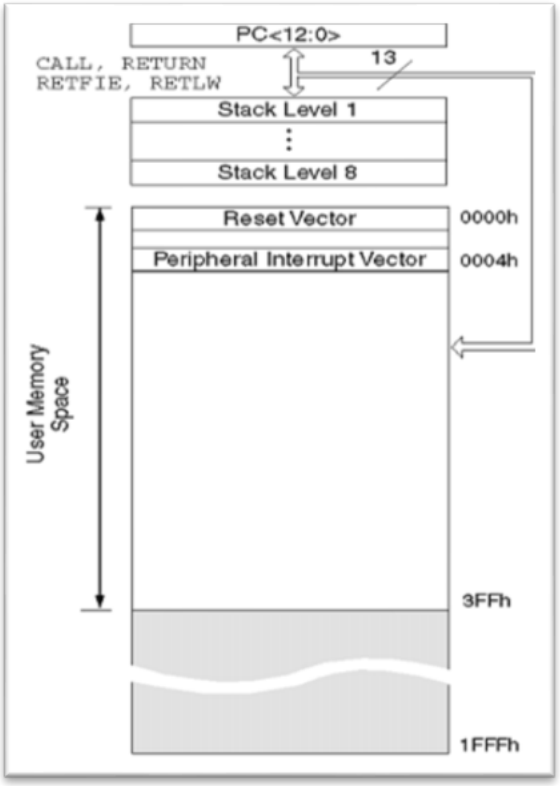
I.4.10 Le registre FSR: Utilisé pour l'adressage indirect de la mémoire RAM.

I.4.11 Le bus de donnée: Il met en liaison les blocs utilisant des données.

I.5 Organisation de Mémoire Programme:

Sur la figure (I.4), nous voyons que l'espace Mémoire Programme commence à l'adresse 0000h et finit à l'adresse FFh. Le « h » indique que le nombre 3 qui précède est écrit en base hexadécimale. Cela permet de loger 1 024 instructions. Enfin, pas tout à fait, la case mémoire 0000h est réservée au vecteur de Reset (dans cette case est logée l'adresse du début du programme) et la case mémoire 0004h au Vecteur d'Interruption des périphériques (utilisée lorsque l'on active les interruptions). Le registre PC, qui pointe en permanence l'instruction à exécuter, se compose de deux registres : l'un, de huit bits, est appelé le PCL(Program Counter Low) et l'autre, de 5 bits, le PCH(Program Counter High), le tout formant 13 bits.

Le PCL est directement accessible en lecture comme en écriture. Par contre, le PCH peut être uniquement accessible en écriture à travers un registre appelé PCLATH.

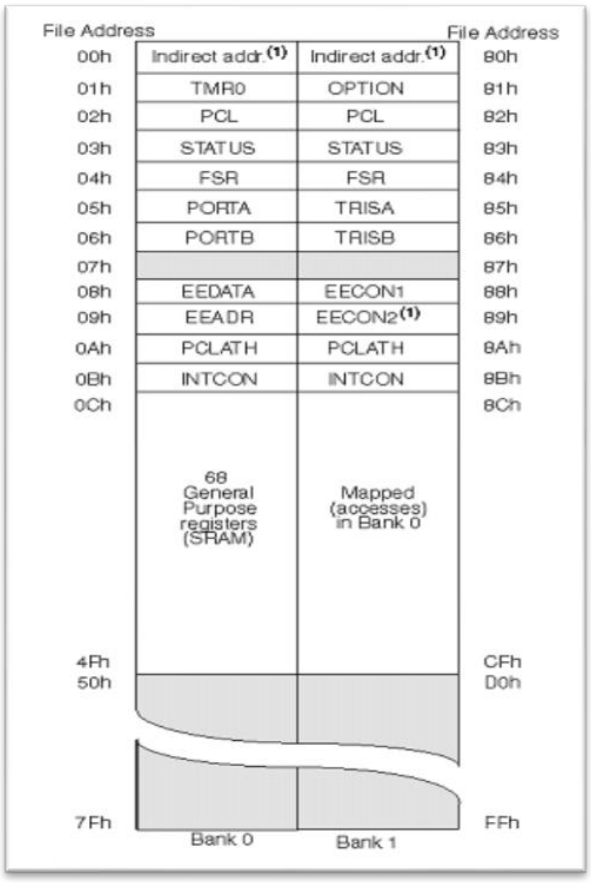


Figure(I.4): Plan de la Mémoire Programme et de la pile.

Le bloc Pile (Stack), à huit niveaux, permet, d'une par t, de mémoriser l'adresse en cours lors d'un « saut » à un sous-programme ou à un programme d'interruption et, d'autre part, de recharger le PC avec cette adresse pour le retour (avec les instructions CALL, RETURN, RETFIE et RETLW). Les huit niveaux de la pile autorisent donc huit sous-programmes imbriqués.

I.6 L'organisation de la Mémoire de Données RAM:

La mémoire RAM est organisée en un bloc de 128 cases mémoires de 8 bits (8 bits étant le format d'une donnée). L'adresse de début étant 00h et l'adresse de fin 7Fh.



Figure(I.5): Plan de la Mémoire de Données RAM

Les cases mémoires comprises entre les adresses 00h et 0Bh sont réservées aux registres de configuration du système (zone SFRs). Partitionnée en deux parties distinctes Bank 0et Bank 1, cette zone possède un double accès. Si l'on active la Bank 0, les registres représentés à gauche sur la figure (I.5) sont accessibles. Si par contre la Bank 1 est activée, les

registres actifs seront ceux de droite. La sélection de la Bank s'effectue à travers les bits de contrôle RP0 et RP1 du registre STATUS (03h).

La zone de mémoire comprise entre les adresses 0Ch et 4Fh (68 cases mémoires) est utilisée pour stocker les données de notre programme. La zone par tant de 8Ch allant jusqu'à CFh de la Bank 1 est une zone « image » de la zone précédente : les données y sont strictement identiques. Par exemple, la valeur inscrite dans la case mémoire 0Ch sera toujours égale à celle de la case mémoire 8Ch.

Maintenant que nous savons accéder à la mémoire de données, voyons le rôle de chacun des registres de configuration.

-TMR0 et OPTION permettent de contrôler le fonctionnement du TIMER interne.

-PCL constitue, comme nous l'avons déjà vu, la partie basse du PC.

- STATUS est un registre (contenant certains bits pouvant être lus et écrits et d'autres pouvant seulement être lus) qui permet de contrôler certaines fonctions du microcontrôleur comme, par exemple, la sélection de la Bank de la mémoire RAM, l'indication sur une opération effectuée par l'ALU, etc.

- FSR est le File Select Registre. Il permet de sélectionner et d'accéder aux données en RAM quand on utilise l'adressage indirect.

- PORTA et PORTB sont deux registres permettant d'accéder aux deux ports du PIC. Lorsqu'un port est utilisé comme sortie, une écriture dans le registre correspondant activera directement les broches de sortie. Inversement, si un port est configuré en entrée, la lecture du registre correspondant donnera l'image binaire des niveaux de tension présents sur les pattes.

- EEDATA, EEADR, EECON1 et EECON2 sont les registres à utiliser pour la mémoire de données EEPROM.

- PCLATH permet d'écrire dans le PCH.

- INTCON est le registre de contrôle des interruptions. Il indique l'apparition d'un événement externe (par exemple le passage d'un niveau haut à un niveau bas sur l'une des broches

d'entrée) ou interne (par exemple la fin de contage du TIMER interne). Il permet aussi de valider les interruptions : lors d'un événement programme en cours peut être arrêté pour laisser la place au programme d'interruption. Une fois ce dernier terminé, le programme principal reprend son déroulement.

I.7 Le registre d'état ou « STATUS »:

Dans cet article, à plusieurs reprises nous avons parlé du registre d'état ou « STATUS ». Nous allons maintenant le décrire en détail. Situés à l'adresse 03h, les 8 bits qui le composent portent les noms suivants : C, DC, Z, PD, RP0, RP1 et IRP. Analysons-les séparément.

- Le bit CARRY(C) indique une retenue lors d'une opération arithmétique. Il est aussi utilisé lors d'une opération de rotation.

- Le bit DIGIT CARRY(DC) indique une retenue sur les quatre premiers bits du résultat d'une opération arithmétique. Ce bit est généralement utilisé pour la conversion « binaire-BCD ».

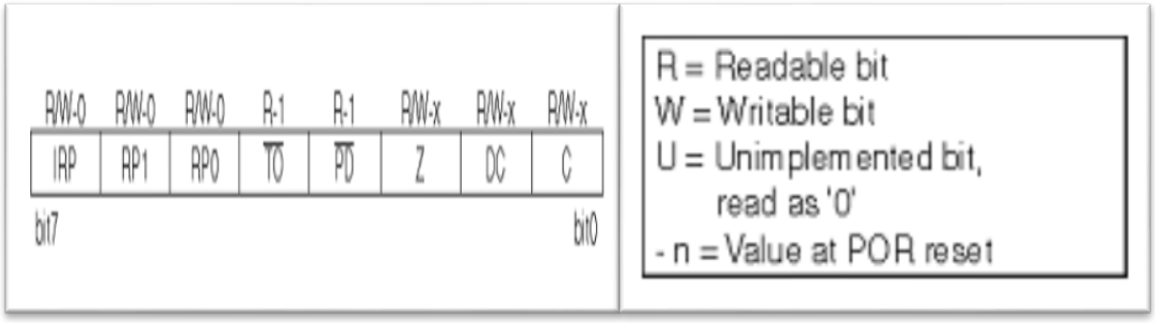
- Le bit ZERO(Z) passe à 1 si le résultat d'une opération est nul.

- Le bit POWER DOWN(PD) est positionné à 0 lorsque le microcontrôleur exécute l'instruction SLEEP qui permet de mettre le PIC en « veille » en arrêtant l'horloge. Au reset, ce bit est positionné à 1.

- Le bit TIME OUT(TO) passe à 0 lorsque le Watch Dog atteint la fin de son comptage. Au reset, ce bit est positionné à 1.

- Les bits RP0et RP1sélectionnent la Bank RAM. Si RP0= 0 et RP1 = 0 la Bank 0 est sélectionnée. Pour accéder à la Bank 1 il faudra RP0 = 1 et RP1 = 0.

- Le bit IRP doit être maintenu à zéro pour le PIC 16F84.



Figure(I.6): Positionnement des bits qui constituent le registre STATUS

I.8 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 :

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(I.7).

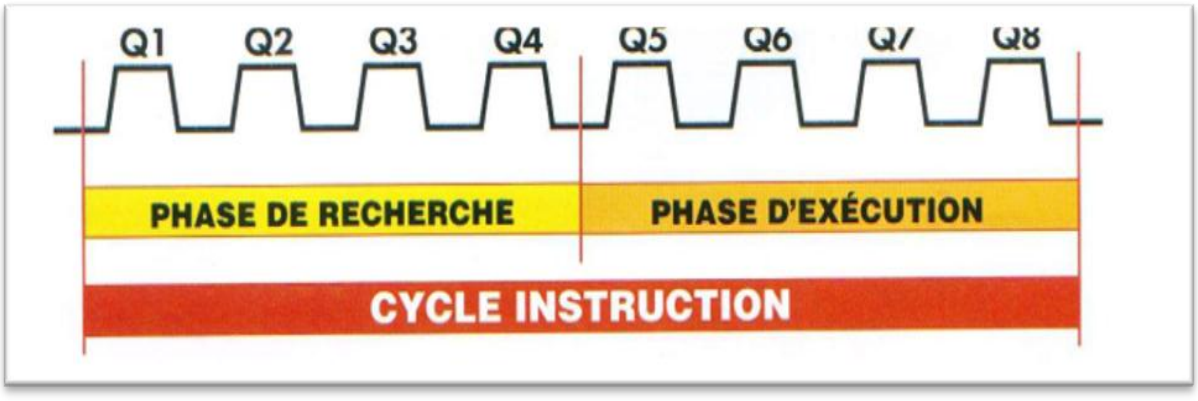
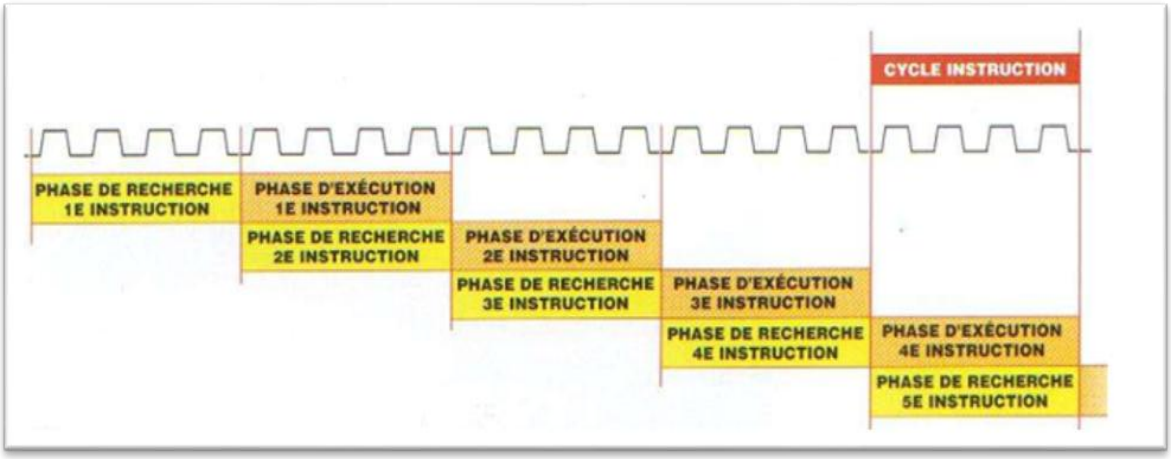


Figure (I.7): la forme générale de l'instruction

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 (I.8) et (I.9)).[3]



Figure(I.8) : La coordination entre l'instruction et signal d'horloge

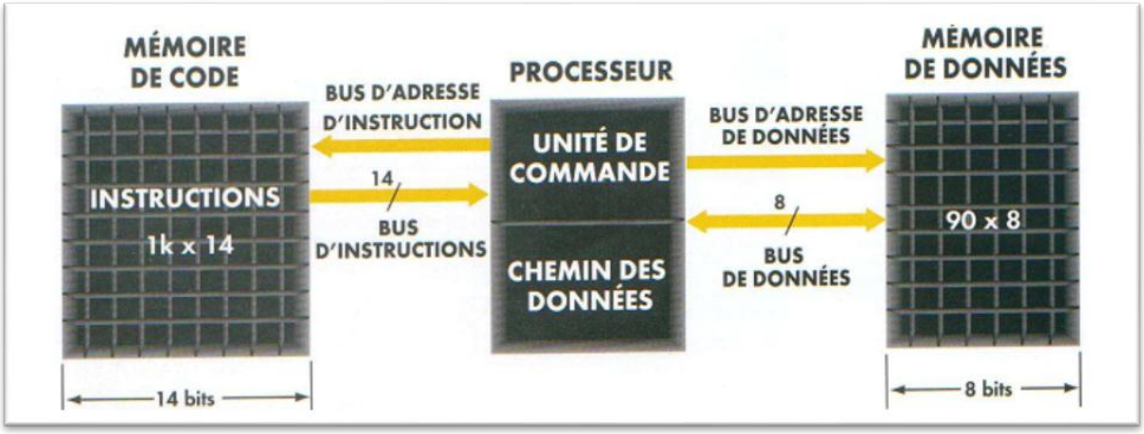


Figure (I .9): la circulation de l'instruction dans la mémoire

I.9 Les Types d'Instructions :

Vous constaterez donc qu'il existe 4 types d'instructions, que nous allons décrire.[1]

I.9 1 Les Instructions « orientées octet » :

Ce sont des instructions qui manipulent les données sous forme d'octets. Elles sont codées de la manière suivant:

- 6 bits pour l'instruction : logique, car comme il y a 35 instructions, il faut 6 bits pour pouvoir les coder toutes.

-1bit de destination(d) : 0 indique que le résultat de l'opération sera placé dans le de travail W (Work), 1 indique que ce résultat sera palcé dans l'opérande registre précisée dans les 7 bits suivants.

- 7 bits pour encoder l'opérande(File) : Si d vaut 1 (voir ci-dessus), cette opérande renseigne à la fois la donnée à manipuler et également l'endroit où le résultat sera stocké.

Aie, premier problème, 7 bits ne donnent pas accès à la mémoire RAM totale, donc voici ici l'explication de la division de la RAM en deux banques. En effet, il faut 8 bits pour accéder à 256 emplacements différents.

Il faudra bien trouver une solution pour remplacer le bit manquant. Il s'agit en réalité du bit RP0du registre STATUS.

I.9.2 Les instructions « orientées bits » :

Ce sont des instructions qui manipulent les données sous forme de bits. Autrement dit, elles sont destinées à modifier des bits précis d'un registre spécifié. Elles sont codées de la manière suivant:

- 4 bits pour l'instruction (dans l'espace resté libre par les instructions précédentes).
- 3 bits pour indiquer le numéro du bit à manipuler (bit 0 à 7 possible).
- 7 bits pour indiquer l'opérande.

I.9.3 Les instructions générales :

Ce sont les instructions qui manipulent des données qui sont codées dans l'instruction directement. Nous verrons ceci plus en détail lorsque nous parlerons des modes d'adressage. Elles sont codées de la manière suivante:

- 6 bits pour coder l'instruction.
- 8 bits pour coder la valeur concernée (valeur dite « immédiate» parce que se trouvant

immédiatement dans l'instruction). La valeur peut de fait varier de 0 à 255.

I.9.4 Les sauts et appels de sous-routines :

Ce sont les instructions qui provoquent une rupture dans la séquence de déroulement du programme. Elles sont codées de la manière suivante :

- 3 bits pour coder l'instruction.
- 11 bits pour coder l'adresse de la destination

I.10 Conclusion :

Dans ce chapitre, on a pris une vue générale sur l'architecture interne et externe du microcontrôleur 16F84, et le principe de configuration des registres suivant les besoins recommandés au niveau de ses portes et broches. Ainsi, le fonctionnement de l'unité de calcul, les mémoires de Données (RAM et ROM) et comment sont-elles organisées. Finalement, les différents types des instructions utilisées au niveau de microcontrôleur 16F84 ont été présentés.

Chapitre

2

PROGRAMMATION DE MICROCONTROLEUR 16F84

II.1 Introduction

II.2 MPLAB

II.3 Programmation des ports A et B

II.4 Les instructions qui programmer le PIC 16F84

II.5 Les modes d’adressage

II.6 Conclusion

5II. Programmation de Microcontrôleur 16F84

II.1 Introduction :

On va aborder dans ce chapitre, la programmation du pic 16F84 à travers un logiciel qui se nomme « MPLAB ». On aura aussi une idée sur les mnémoniques des instructions en assembleur qui seront touchés en les manipulant dans des exemples.

II.2 MPLAB :

MPLAB® est un logiciel qui est construit sur la notion de projets. Un projet permet de mémoriser tout l'environnement de travail nécessaire à la construction d'un projet. Il nous permettra de rétablir tout notre environnement de travail lorsque nous le sélectionnerons.[1]

II.3 Programmation des ports A et B :

Avant la reconnaissance des instructions qui programmer de pic 16F84, nous avions programmer les ports (entrée, sortie) A et B. Il faut reconnaître le registre qui appelle trise (trise A et trise B) sont configuré les ports A et B. Il définissent, sont les registres de configuration des ports A et B. Ils définissent quels bits seront utilisés en entrée ou en sortie. Un « 1 » configure le bit en entrée et un « 0 » en sortie.[2]

II.4 Les instructions qui programmer le PIC 16F84 :

II.4.1 L'instruction « GOTO » (aller à) :

Cette instruction effectue ce qu'on appelle un saut inconditionnel, encore appelé rupture de séquence synchrone inconditionnelle. « Rupture de séquence » parce que le programme ne se déroule pas dans l'ordre des instructions, « synchrone » parce qu'on sait à quel «instant » du programme se produit la rupture de séquence (l'endroit du « goto »), et «inconditionnelle » parce que le saut se produit dans tous les cas, en l'absence de toute condition.

Syntaxe: goto étiquette

Remarquez que vous pouvez sauter en avant ou en arrière. goto nécessite 2 cycles d'horloge, comme pour tous les sauts.[1]

II.4.2 L'instruction « INCF » (Incrément File) :

Cette instruction provoque l'incrémentation de la valeur contenue à l'emplacement spécifié (encore appelé File).

Syntaxe: Incf f,d

Comme pour toutes les instructions, «f» représente « File », c'est à dire l'emplacement mémoire concerné pour cette opération, « d », quant à lui: représente la destination.« f» vaut donc dans la réalité une adresse, ou le symbole d'une adresse, ou le symbole d'une adresse, comme ceci:

incf variable,f

Ne confondez donc pas le « f» remplacé par « variable » par le f situé après la virgule et qui précise la destination.

Sauf spécification contraire, d vaut toujours, au choix:

- F(la lettre f) ou 1(le chiffre) : dans ce cas le résultat est stocké dans l'emplacement mémoire précisé par f (donc dans « variable » pour notre exemple).

- W(la lettre w) ou 0(le chiffre) : dans ce cas, le résultat est laissé dans le registre de travail et le contenu de l'emplacement mémoire n'est pas modifié. Ce registre de

travail est appelé également « accumulateur».

La formule est donc (f) + 1 -> (d): Les parenthèses signifient « le contenu de ». Soit, en français : Le contenu de l'emplacement spécifié est incrémenté de 1, le résultat est placé dans l'emplacement désigné par « d ». Emplacement qui pourra être soit l'emplacement spécifié par « f », soit l'accumulateur, (f) restant dans ce cas inchangé.

Bit du registre STATUS affecté:

Le seul bit affecté par cette opération est le bit Z.

Rappel : Étant donné que la seule manière en incrémentant un octet d'obtenir 0, c'est de passer de 0xFF à 0x00. Le report n'est pas nécessaire, puisqu'il va de soi. Si, après une incrémentation, vous obtenez Z=1 , c'est que vous avez débordé. Z vaudra donc 1 si (f) avant l'exécution valait 0xFF, il n'y a aucune autre possibilité.

II.4.3 L'instruction «DECF» (Incrément File) :

Décrémente le contenu de l'emplacement spécifié. Le fonctionnement est strictement identique à l'instruction précédente, excepté que cette fois, on décrémente.

Syntaxe: Decf f, d ; (f) – 1 -> (d).

Bit du registre STATUS affecté:

Le seul bit affecté par cette opération est le bit Z .

Si avant l'instruction, (f) vaut 1, Z vaudra 1 après l'exécution (1-1 = 0).

II.4.4 L'instruction «MOVLW» (Move Literal to W) :

Cette instruction charge la valeur spécifiée dans le registre W(registre de travail). La valeur précisée est dite « littérale » ou « immédiate » car ne dépend pas du contenu d'un emplacement mémoire.

Syntaxe: movlw k ; k-> w: k représente une valeur de 0x00 à 0xFF

Bit du registre STATUS affecté:

Aucun(donc même si vous chargez la valeur '0'.).

II.4.5 L'instruction « MOVF » (Move File):

Charge le contenu de l'emplacement spécifié dans la destination.

Syntaxe: movf f, d ; (f) -> (d)

Bit du registre STATUS affecté:

Une fois de plus, seul le bit Z est affecté : si (f) vaut 0, Z vaut 1.

II.4.6 L'instruction « MOVWF » (Move W to File):

Permet de sauvegarder le contenu du registre de travail W dans un emplacement mémoire. C'est l'opération inverse de movf f,d .

Syntaxe: movwf f ; (W) -> (f)

Bit du registre STATUS affecté:

Aucun.

II.4.7 L'instruction « ADDLW » (ADD Literal and W):

Cette opération permet d'ajouter une valeur littérale au contenu du registre de travail W. Nous sommes de nouveau dans le cas de l'adressage immédiat.

Syntaxe: addlw k ; (W) + k -> (W)

Bits du registre STATUS affectés:

Z : Si le résultat de l'opération vaut 0, Z vaudra 1.

C : Si le résultat de l'opération est supérieur à 0xFF (255) , C vaudra 1.

DC: Si le résultat de l'opération entraîne en report du bit 3 vers le bit 4, DC vaudra 1.

II.4.8 L'instruction « ADDWF » (ADD W and F):

Le contenu du registre W est ajouté au contenu du registre F. Ne pas confondre avec l'instruction précédente. Il s'agit maintenant d'un adressage direct.

Syntaxe: Addwf f, d ; (w) + (f) -> (d)

Bits du registre STATUS affectés: C, DC, et Z

II.4.9 L'instruction « SUBLW » (SUBtract W from Literal):

Attention, ici il y a un piège. L'instruction aurait dû s'appeler SUBWL. En effet, on soustrait W de la valeur littérale, et non l'inverse. Comme le nom l'indique, il s'agit d'un adressage littéral ou immédiat.

Syntaxe: sublw k ; k – (W) -> (W)

Bits du registre STATUS affectés:

C, DC, Z

Notez ici que le bit C fonctionne de manière inverse que pour l'addition. Ceci est commun à la plupart des microprocesseurs du marché. Les autres utilisent parfois un bit spécifique pour la soustraction, bit le plus souvent appelé « borrow» (emprunt).

Si le résultat est positif, donc, pas de débordement : C =1. S' il y a débordement, C est forcé à 0.

Ceci est logique, et s'explique en faisant une soustraction manuelle. Le bit C représente le $9^{éme}$ bit ajouté d'office à 1 à la valeur initiale. Si on effectue une soustraction manuelle donnant une valeur <0, on obtient donc une valeur finale sur 8 bits, le report obtenu venant soustraire le bit C. Si le résultat est >0, il n'y a pas de report, le résultat final reste donc sur 9 bits.

La formule de la soustraction est donc : k précédé d'un neuvième bit à 1 – contenu de W = résultat sur 8 bits dans W avec $9^{éme}$ bit dans C.

II.4.10 L'instruction « SUBWF » (SUBtract W from F):

Cette opération soustrait le contenu de W du contenu de F. Nous restons dans les soustractions, mais, cette fois, au lieu d'un adressage immédiat, nous avons un adressage direct.

subwf f, d ; (f) – (W) -> (d) Syntaxe:

Bits du registre STATUS affectés: C ,DC, Z

II.4.11 L'instruction « ANDLW » (AND Literal with W):

Cette instruction effectue un « ET logique » bit à bit entre le contenu de W et la valeur littérale qui suit. Il s'agit donc d'un adressage littéral ou immédiat.

Syntaxe: andlw k ; (w) AND k -> (w)

Bit du registre STATUS affecté: Z

II.4.12 L'instruction « ANDWF » (AND W with F):

Réalise un « ET logique » bit à bit entre le contenu du registre W et le contenu de F.

Maintenant, vous devriez avoir bien compris, il s'agit de la même opération que précédemment, mais réalisée en adressage direct. Je vais donc accélérer les explications.

Syntaxe: andwf f, d ; (f) AND (w) -> (d)

Bit du registre STATUS affecté: Z

II.4.13 L'instruction « IORLW » (Inclusive OR Literal with W):

Réalise un "OU inclusif logique" bit à bit entre le contenu du registre w et la valeur mmédiate précisée. La notion d'inclusif, nous en avons déjà parlé, précise que le résultat de l'opération pour un rang de bit donné vaudra 1 si un des 2 bits de ce rang vaut 1, incluant le cas où les deux bits valent simultanément 1. Nous avons ici affaire à un adressage littéral, ou immédiat .

Syntaxe: iorlw k ; (w) OR k -> (w)

Bit du registre STATUS affecté: Z

II.4.14 L'instruction « IORWF » (Inclusive OR W with File):

Effectue un « OU inclusif logique » bit à bit entre le contenu de W et le contenu de F. Il s'agit d'une instruction utilisant l'adressage direct. Je ne donnerai pas d'exemple, vous devriez maintenant avoir compris .

Syntaxe: iorwf f, d ; (w) OR (f) -> (d)

Bit du registre STATUS affecté: Z

II.4.15 L'instruction « XORLW » (eXclusive OR Literal with W):

Réalise un « OU exclusif logique » bit à bit entre le contenu du registre W et la valeur littérale précisée. La notion « exclusif» se rapport au fait que, pour un rang donné, le bit de résultat vaudra 1 si le bit de ce rang pour le premier opérateur vaut 1 ou si celui du second opérateur vaut 1, à l'exclusion du cas où les deux bits valent 1 simultanément. Nous avons affaire ici à de l'adressage littéral, ou immédiat.

Syntaxe: xorlw k ; (w) xor k -> (w)

Bit du registre STATUS affecté: Z

II.4.16 L'instruction « XORWF » (eXclusive OR W with F):

Effectue un « OU exclusif logique » bit à bit entre le contenu du registre W et le contenu de F. Il s'agit exactement la même opération que XORLW, mais en adressage direct.

Syntaxe: xorwf f, d ; (w) xor (f) -> (d)

Bit du registre STATUS affecté: Z

II.4.17 L'instruction « BSF » (Bit Set F):

Force le bit de rang spécifié du contenu de l'emplacement F à 1. Dit autrement, force un numéro de bit d'un registre précisé à 1. Il s'agit d'un adressage direct .

Syntaxe:

```
bsf  f, b                ;(f).b = 1

                         ; le bit n° b est positionné dans la case mémoire (f)

                         ;b est évidemment compris entre 0 et 7
```

Bit du registre STATUS affecté: Aucun

II.4.18 L'instruction « BCF » (Bit Clear F):

Force le bit de rang spécifié du contenu de l'emplacement F à 0. Dit autrement, force un numéro de bit d'un registre précisé à 0. Il s'agit d'un adressage direct.

Syntaxe:

```
bsf f , b                ;(f).b = 0

                         ; le bit n° b est positionné dans la case mémoire (f)

                         ;b est évidemment compris entre 0 et 7
```

Bit du registre STATUS affecté: Aucun

II.4.19 L'instruction « RLF » (Rotate Left through Carry):

Rotation vers la gauche du registre F en utilisant le carry. Il s'agit ici d'une instruction utilisant l'adressage direct.

Les opérations de décalage sont des opérations très souvent utilisées. Les PIC16F® ont la particularité de ne disposer que d'instructions de rotation sur 9 bits. Vous allez voir qu'avec ces instructions, on peut très facilement réaliser des décalages. Le mot qui va subir les rotations est constitué des 8 bits du registre spécifié, complété par le bit Carry du registre STATUS.

L'opération de rotation effectue l'opération suivante : Le bit de carry C est mémorisé Ensuite chaque bit de l'octet est déplacé vers la gauche. L'ancien bit 7 sort de l'octet par la gauche, et devient le nouveau carry. Le nouveau bit 0 devient l'ancien carry. Il s'agit donc bien d'une rotation sur 9 bits .

Syntaxe: rlf f, d ; (f) rotation gauche avec carry-> (d)

Bit du registre STATUS affecté: C

II.4.20 L'instruction « RRF » (Rotate Right through Carry):

Rotation vers la droite du registre F en utilisant le carry. Il s'agit ici d'une instruction utilisant l'adressage direct.

L'opération de rotation vers la droite effectue l'opération suivante : Le bit de carry « C» est mémorisé. Ensuite chaque bit de l'octet est déplacé vers la droite. L'ancien bit 0 sort de l'octet par la droite, et devient le nouveau carry. L'ancien carry devient le nouveau bit7. Il s'agit donc également d'une rotation sur 9 bits.

Syntaxe: rrf f, d ; (f) rotation droite avec carry-> (d)

Bit du registre STATUS affecté: C.

II.4.21 L'instruction « BTFSC » (Bit Test F, Skip if Clear):

Teste le bit précisé du contenu de l'emplacement F et saute s'il vaut 0. Il s'agit ici de votre premier saut conditionnel (lié à une condition), ou rupture de séquence synchrone conditionnelle.

En effet, il n'y aura saut que si la condition est remplie. Les instructions conditionnelles sont la base de tout ce qui est « prise de décision » dans un programme.

Notez que dans ce cas l'instruction prendra 2 cycles, sinon, elle n'utilisera qu'un cycle. De plus, il faut retenir que pour tous ces types de saut, on ne saute que l'instruction suivante(skip et non goto). En effet, la syntaxe ne contient pas d'adresse de saut, comme nous allons le voir.

Syntaxe:

```
btfsc f, b  ; on teste le bit b de la mémoire (f)
            ;si ce bit vaut 0, on saute l'instruction suivante, sinon
            ;on exécute l'instruction suivante.
Instruction x  ; exécutée seulement si f.b vaut 1
Instruction y  ; exécutée directement si b vaut 0
```

Notez que pour passer de l'instruction btfsc à l'instruction y il vous faudra en générale 2 cycles. En effet:

- Soit la condition f.b est remplie et donc au saute l'instruction x, un saut prenant 2 cycles.

- Soit la condition n'est pas remplie et on exécute l'instruction x. Il n'y a pas de saut et donc btfsc prend un seul cycle, mais il faut un cycle également pour exécuter l'instruction x. Notez que ceci ne vaut que si l'instruction x n'est pas elle-même une instruction de saut, selon la syntaxe suivante, par exemple:

```
btfsc f, b          ; on teste le bit b de la mémoire (f).
                    ;si ce bit vaut 0, on saute l'instruction suivante, sinon
                    ;on exécute l'instruction suivante
goto plus loin      ; si b vaut 1 on traite plus loin ce cas
Instruction y  ; on traite ici le cas où b valait 0
…
…
…
```

Plus loin : ; on traite ici le cas où b valait 1

instruction z

Cette façon de procéder est très souvent utilisée lorsqu'il y a plusieurs instructions à exécuter dans un cas et dans l'autre. Le « goto » peut éventuellement être remplacé par un «call » si le déroulement du programme doit reprendre à l'instruction y dans tous les cas.

Ces façons de faire sont applicable aux autres instructions impliquant un « skip », je n'en reparlerai donc pas

Bit du registre STATUS affecté: Aucun

II.4.22 L'instruction « BTFSS » (Bit Test F, Skip if Set):

Teste le bit précisé du contenu de l'emplacement F et saute s'il vaut 1.

Le fonctionnement est strictement identique à celui de l'instruction précédente.

Syntaxe: btfss f, b ; on teste le bit b de la mémoire (f).

;si ce bit vaut 1, on saute l'instruction

;suivante, sinon

;on exécute l'instruction suivante

xxxx ; si le bit vaut 1, ne sera pas exécutée (skip)

xxxx ; Le programme continue ici

Bit du registre STATUS affecté: Aucun

II.4.23 L'instruction « DECFSZ » (DECrement F, Skip if Z):

Décrémente le contenu de l'emplacement F et saute l'instruction suivante si le résultat vaut 0. Cette instruction est très utilisée pour créer des boucles.

Syntaxe: decfsz f, d ; (f) –1 -> (d). Saut si (d) = 0

Bit du registre STATUS affecté: Aucun.

II.4.24 L'instruction « INCFSZ » (INCrement F, Skip if Zero):

Incrémente le contenu de l'emplacement F et saute l'instruction suivante si le résultat vaut 0. Je ne vais pas détailler cette instruction, car elle est strictement identique à la précédente, hormis le fait qu'on incrémente la variable au lieu de la décrémenter

Syntaxe: incfsz f, d ; (f) + 1 -> (d) : saut si (d) = 0

Bit du registre STATUS affecté: Aucun

II.4.25 L'instruction « SWAPF » (SWAP nibbles in F):

Inverse les 2 quartets du contenu de l'emplacement F.. Cette opération inverse simplement le quartet (demi-octet) de poids faible avec celui de poids fort. Ce genre d'opération est utile lorsque vous manipulez des quartet, l'exemple typique étant le codage BC.D

Syntaxe: swapf f, d ; inversion des b0/b3 de (f) avec b4/b7 -> (d)

Bit du registre STATUS affecté:

Aucun: cette particularité nous sera très utile lorsque nous verrons les interruptions.

II.4.26 L'instruction « CALL » (CALL subroutine):

Appelle une sous-routine. Cette opération effectue un saut inconditionnel vers un sous-programme. Il s'agit d'une rupture de séquence inconditionnelle synchrone avec mémorisation de l'adresse de retour.

Qu'est un sous-programme ? Et bien, il s'agit tout simplement d'une partie de programme qui peut être appelé depuis plusieurs endroits du programme dit « principal » et

qui présente la particularité de renvoyer en fin de son exécution à l'instruction suivant l'endroit où il avait été appelé

Syntaxe: call sous routine ; (pc)->(pile), puis sous routine + PCLATH -> (pc)

Bit du registre STATUS affecté: Aucun

II.4.27 L'instruction « RETURN » (RETURN from subroutine):

Retour de sous-routine. Va toujours de pair avec une instruction call. Cette instruction indique la fin de la portion de programme considérée comme sous-routine (SR). Rappelez-vous que pour chaque instruction « call» rencontrée, votre programme devra rencontrer une instruction « return» .

Syntaxe: return ; (pile) -> PC

;Le programme poursuit à l'adresse qui suit la ligne call

Bit du registre STATUS affecté: Aucun

II.4.28 L'instruction « RETLW » (RETurn with Literal in W):

Retour de sous-routine avec valeur littérale dans W. C'est une instruction très simple: elle équivaut à l'instruction return, mais permet de sortir d'une sous-routine avec une valeur spécifiée dans W.

Syntaxe: retlw k ; k -> (w) puis (pile)->(pc)

Bit du registre STATUS affecté: Aucun

II.4.29 L'instruction « RETFIE » (RETurn From IntErrupt):

Retour d'interruption. Nous verrons dans un chapitre séparé ce que sont les interruptions, il serait idiot de caser tout un chapitre dans l'explication d'une instruction. Sachez cependant que cette instruction effectue un retour d'interruption, exactement comme

return effectue un retour de sous-programme, à ceci près que REFTIE relance automatiquement les interruptions, il y a donc une opération supplémentaire réalisée.

Syntaxe: retfie ; retour d'interruption

Bit du registre STATUS affecté: Aucun

II.4.30 L'instruction « CLRF » (CLeaR F):

Efface le contenu de l'emplacement F. Effacer signifie simplement « placer la valeur 0» à l'emplacement précisé.

Syntaxe: clrf f ; (f) = 0

Bit du registre STATUS affecté:

Z: Vaut donc toujours 1 après cette opération, l'intérêt est donc pour le moins limité.

II.4.31 L'instruction « CLRW » (CLear W):

Efface le registre W. Dit autrement, charge la valeur 0 dans le registre de travail.

Syntaxe: clrw ; (w) = 0

C'est une instruction qui n'est pas vraiment indispensable, car on pourrait utiliser l'instruction « movlw 0». Cependant, à la différence de movlw 0, clrw positionne le bit Z.

Bit du registre STATUS affecté: Z: Vaut donc toujours1 après cette opération.

II.4.32 L'instruction « CLRWDT » (CLeaR WatchDog):

Réarme le chien de garde(watchdog) de votre programme. Nous aborderons la mise en œuvre du watchdog ultérieurement. Sachez cependant que c'est un mécanisme très pratique qui permet de provoquer un reset automatique de votre PIC® en cas de plantage du programme (parasite par exemple).

Le mécanisme est de toutes façons très simple à comprendre : il s'agit pour votre programme d'envoyer cette instruction à intervalles réguliers. Si la commande n'est pas reçue dans le délai imparti (programmable), le PIC® redémarre à l'adresse 0x00. C'est exactement le mécanisme, dit « de l'homme mort » utilisé par les conducteurs de train qui doivent presser un bouton à intervalle régulier. Si le bouton n'est pas pressé, le train s'arrête. On détecte ainsi si le conducteur est toujours dans l'état d'attention requis (et vivant).

Syntaxe: clrwdt ; (wdt) = 0

Bit du registre STATUS affecté: Aucun.

II.4.33 L'instruction « COMF » (COMplement F) :

Effectue le complément à 1 du contenu de l'emplacement mémoire spécifié. Dit de façon plus évidente : inverse tous les bits de l'emplacement.

Syntaxe: comf f , d ; NOT (f) -> (d)

Bit du registre STATUS affecté: Z

II.4.34 L'instruction « SLEEP » (Mise en sommeil) :

Place le PIC® en mode sommeil. Arrêt de l'exécution du programme et diminution de la consommation en sont les résultats espérés. Il ne se réveillera que sous certaines conditions que nous verrons plus tard.

Syntaxe: sleep ; Silence, je dors!

Bit du registre STATUS affecté:

T0, PD : ces bits seront décrits dans l'étude des modes de reset du PIC.

II.4.35 L'instruction « NOP » (No Opération):

Aucune opération. Comme vous devez être fatigué, et moi aussi, je vous présente l'instruction qui ne fait rien, quine positionne rien, et qui ne modifie rien. On pourrait croire

qu'elle ne sert à rien. En fait elle est sur tout utilisée pour perdre du temps, par exemple pour attendre une ou deux instructions, le temps qu'une acquisition ai pu se faire, par exemple. Nous l'utiliserons donc à l'occasion

Syntaxe: nop ; Faire semblant de travailler, c'est déjà travailler!

Ceci termine l'analyse des 35 instructions utilisées normalement dans les PIC® mid-range.Ceci peut vous paraître ardu, mais en pratiquant quelque peu, vous connaîtrez très vite toutes ces instructions par cœur. Pensez pour vous consoler que certains processeurs CISC disposent de plusieurs centaines d'instructions.

II.5 Les modes d'adressage:

II.5.1 L'adressage littéral ou immédiat :

Avec l'adressage immédiat, ou littéral, vous pouvez dire:

« j'empoche 100€ »

La valeur fait immédiatement partie de la phrase elle-même. J'ai donné littéralement la valeur concernée. Je n'ai besoin d'aucune autre information située ailleurs pour savoir la somme que j'ai empochée

Exemple:

```
movlw  0x55 ; charger la valeur0x55 dans W
```

II.5.2 L'adressage direct:

Avec l'adressage direct, vous pouvez dire en allant à votre banque:

«Je vais empocher le contenu du compte n° 123456. »

Ici, le renseignement m'indiquant où se trouve l'argent est indiqué directement dans la phrase. Cependant, il m'est nécessaire d'aller voir dans l'emplacement en question(le compte 123456) pour savoir ce que je vais empocher.

En effet, je vais mettre le contenu du compte numéro 123456 et non un montant de 123456€ (dommage). On ne met donc pas en poche le numéro du compte, mais ce que contient le compte qui possède ce numéro. Pour faire l'analogie avec les syntaxes expliquées dans les instructions, je mets (compte 123456) dans ma poche.

Exemple:

```
movf  0x10, W     ; charger le contenu de l'emplacement 0x10 dans W
```

II.5.3 L'adressage indirect:

Avec l'adressage indirect, vous pouvez imaginer que vous voulez empocher par procuration l'argent du compte de votre ami. Or, lui connaît son numéro de compte mais pas vous. Vous pouvez donc dire:

«Mon ami va me fournir le numéro de compte dont je vais empocher le contenu »

Vous avez donc plusieurs opérations à réaliser :

- Tout d'abord, demander le numéro du compte à votre ami (sympa, votre ami).

- Ensuite, vous devez regarder ce que contient ce compte.

- Et seulement maintenant vous savez combien vous allez empocher (et lui rendre, évidemment, vu que vous êtes honnête).

Ceci permet de dire que vous allez obtenir le numéro du compte à utiliser indirectement par l'intermédiaire de votre ami.

Vous n'empochez donc ni votre ami (le pauvre),ni le numéro du compte, mais le contenu du compte dont le numéro est fourni par votre ami. Le chemin d'accès est donc beaucoup plus indirect . (ami)->(compte)->poche

Je résume donc:

Adressage littéral : montant -> poche

Adressage direct : (compte) -> poche

Adressage indirect : (ami)->(compte)->poche

Si vous avez compris ceci, vous avez tout compris. Comme il faut un élément d'indirection (votre ami), ceci se traduit dans le PIC par la présence de registres particuliers. Examinons-les donc

II.5.4 Les registres FSR et INDF:

INDF signifie INDirect File. Vous le voyez maintenant ? Et oui, c'est le fameux registre de l'adresse 0x00. En fait, ce registre n'existe pas vraiment, ce n'est qu'un procédé d'accès particulier à FSR utilisé par le PIC® pour des raisons de facilité de construction électronique interne.

Sur certains micros, le mode d'adressage indirect est spécifié dans l'instruction. Dans les PIC16F, le mode d'adressage indirect utilise un pseudo-registre pour y accéder. De nouveau un mode présent non répertorié dans la liste des instructions explicites.

Le registre FSR quant à lui se trouve à l'adresse 0x04 dans les 2 banques. Il n'est donc pas nécessaire de changer de banque pour y accéder, quelle que soit la banque en cours d'utilisation. Il est, lui, un véritable registre.

Dans l'exemple schématique précédent, votre ami est représenté par ce registre FSR. L'adressage indirect est un peu particulier sur les PIC®, puisque c'est toujours à la même adresse que se trouvera l'adresse de destination. En somme, on peut dire que vous n'avez qu'un seul ami (sniff). Heureusement ce dernier dispose de plusieurs comptes. Comment tout cela se passe-t-il en pratique?

Premièrement, nous devons écrire l'adresse pointée (le numéro du compte) dans le registre FSR. Ensuite, nous accédons à cette adresse pointée par le registre INDF.

On peut donc dire que INDF est en fait le registre FSR utilisé pour accéder à la case mémoire. Donc, quand on veut modifier la case mémoire pointée, on modifie FSR, quand on

veut connaître l'adresse de la case pointée, on accède également à FSR. Si on veut accéder au contenu de la case pointée, on accède via INDF. Nous allons voir tout ceci par un petit.

II.6 Conclusion:

On a étudié dans ce chapitre les principales instructions de programmation de pic 16F84 avec quelques exemples afin de clarifier leurs fonctionnement pendant un programme réel.

En fait, il n'y pas que le software « MPLAB » comme outil de programmation. Il existe plusieurs logiciel pour cet objet, par exemple le langage C à travers le logiciel « micro C », où les instructions de programmation sont déférentes. L'inconvénient de tout langage différent de celui de l'assembleur se réside au niveau de la taille de programme à la RAM qui est toujours plus grande que celle de l'assembleur.

C'est pour cette raison, la programmation directe en assembleur offre la bonne gestion de mémoire et fournit un programme plus optimisé que d'autres langages de programmation.

Chapitre

3

PROJET DE PROGRAMMATION DES FEUX DE CIRCULATION

III. Projet de Programmation des Feux de circulation

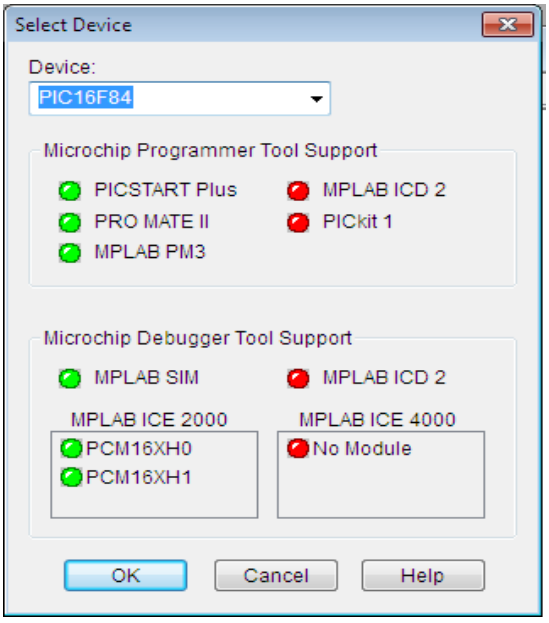
III.1Introduction:

A la fin de cette étude, on verra dans chapitre un projet à base d'un microcontrôleur pic 16F84, dans lequel on veut faire un programme d'un Feu de circulation. Cette application est formée par des LEDs (rouge, orange et vert) qui s'allume conjointement avec les commandes d'un microcontrôleur pic. Avant d'aborder ce projet, on va voir comment faire sélectionner un nouveau projet dans MPLAB, puis en écrivant l'organigramme qui fait l'objet de ce travail suivi par le programme en assembleur alloué.

III.2Comment faire sélectionner un nouveau projet dans MPLAB ?

Le téléchargement étant effectué (on peut aussi se procurer le logiciel sur CD) son installation sur l'ordinateur n'offre pas de difficultés particulières. Sous Windows une icône apparaît sur le bureau . En cliquant sur elle on obtient (si aucun projet n'a été préalablement entré) l'écran ci-dessous.**[4]**

Figure(III.1): L'écran vide avec menu et barres d'outils

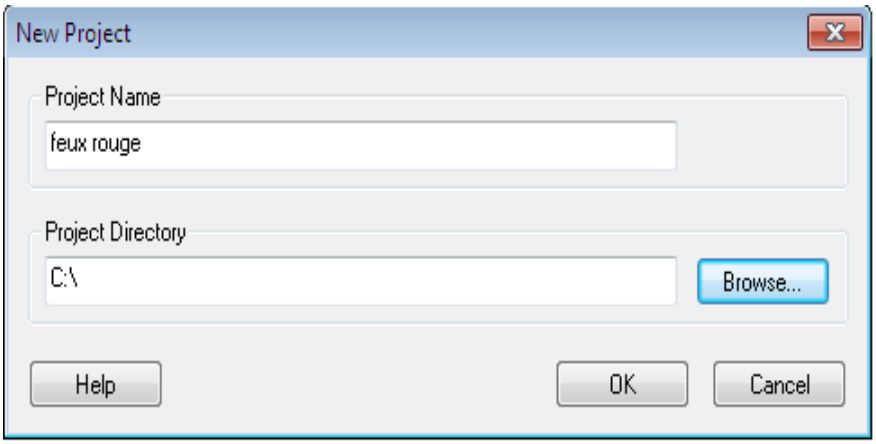


Figure(III.2): une fenêtre paraître nous
Proposer le chois de PIC

Pour faire fonctionner votre système nous serons amenés à développer plusieurs fichiers (Source , objet etc..) dont l'ensemble est appelé un PROJET MPLAB vous demande d'attribuer àce PROJET un nom , il ouvrira alors un répertoire spécial à ce nom pour stocker tous les fichiers nécessaires.

Dans la barre d'outils on commence donc par cliquer sur PROJECT puis NEW: Une fenêtre s'ouvre dans laquelle nous entrons le nom du projet ;par exemple feux rouge ainsi que le répertoire dans lequel il sera créé sur le disque ; par exemple C:\TRAVAIL.

Entrée du nom du projet et localisation



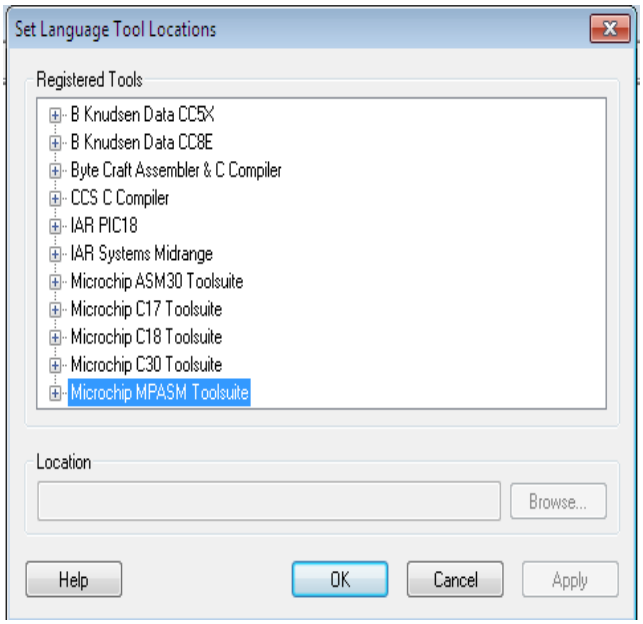
Figure(III.3): La fenêtre s'ouvre nous permet d'introduit le nom
Du projet et répertoire de travail du dit projet

Dans le cadre en haut à gauche apparaît le noms du projet qui va être créé feux rouge.mcp ainsi que les outils qui seront utilisés.

Il faut maintenant configurer l'outil pour cela on utile encore la barre d'outils CONFIGURE > Setting Project, plusieurs cadres s'ouvrent .Dans un premier temps ne rien changer aux options par défaut proposées.

Puis CONFIGURE > Select Device

Il s'agit de définir le type de microcontrôleur qu' on utilise. Dans la fenêtre qui s'ouvre vous pouvez choisir le boîtier .Les boutons rouges ou verts vous indiquent quels sont les outils de développement commerciaux de MICROCHIP sont compatibles avec ce boîtier.



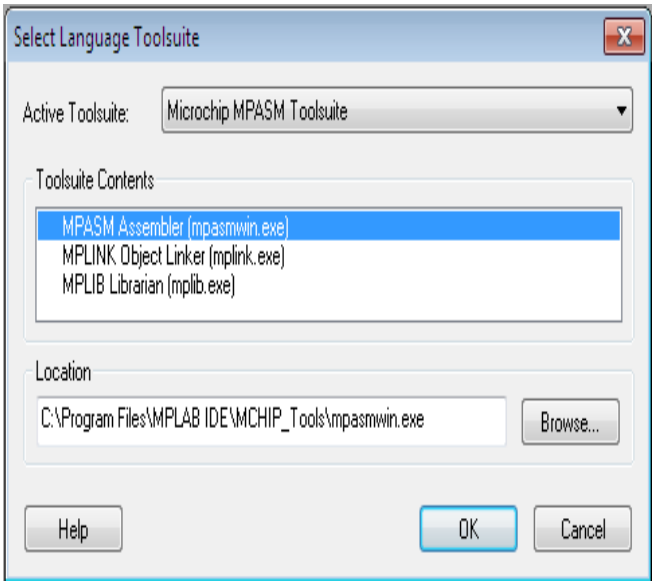
Figure(III.4): contient tous les outils nécessaires à l'introduction du fichier assembleur et son test logiciel.

Il faut maintenant introduire le texte assembleur proprement dit. MPLAB contient tous les outils nécessaires à l'introduction du fichier assembleur et son test logiciel.

Il faut d'abord préciser quel est l'outil utiliser .

Pour cela Project > Set Language Tool Location.

Apparaît la fenêtre ci-dessous :



Figure(III.5): la fenêtre qui s'ouvre sélectionnez

Dans le menu déroulant

Valider MPASM Assembler puis on sélectionne le langage Project > Set Language ToolSuite

Dans la fenêtre supérieure du cadre activer (si ce n'est pas déjà le cas) Microchip MPASM TooolSuite Puis OK.

Maintenant, on peut créer le code source. Sélectionner Files > New.

III.3Principe de fonctionnement de ce travaille:

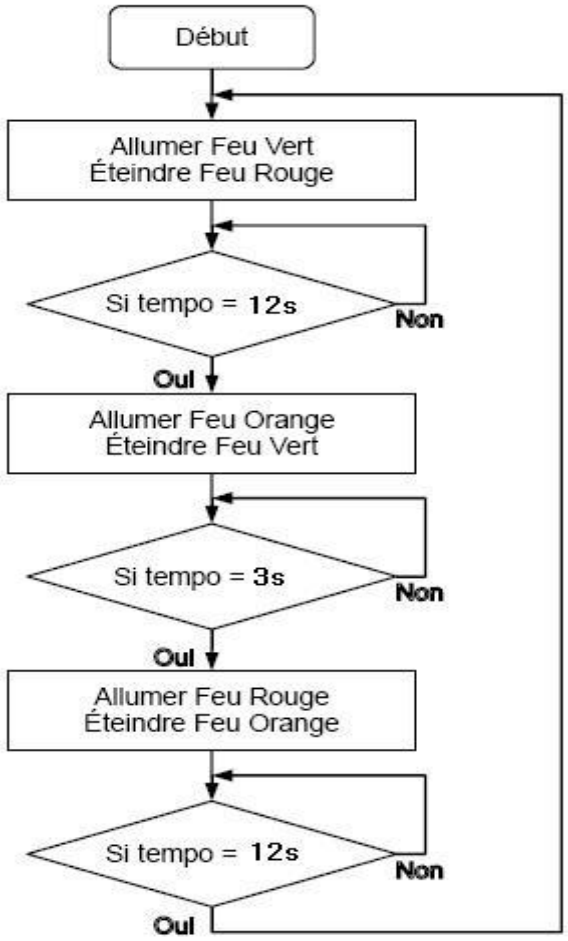
Les feux rouges dans un carrefour sont compose par des LEDs rouge, orange et vert.et sont supposés comme suit :

- Rouge : dans la première lignes marche 12sec, où « Tout conducteur doit marquer l'arrêt absolu devant un feu de signalisation rouge, fixe ou clignotant. »

- Vert : dans la deuxième lignes marche 12sec, où« Les feux de signalisation verts autorisent le passage des véhicules […]. »

- Orange : dans la troisième lignes marcher 3s, où« Tout conducteur doit marquer l'arrêt devant un feu de signalisation jaune fixe, sauf dans le cas où, lors de l'allumage dudit feu, le conducteur ne peut plus arrêter son véhicule dans des conditions de sécurité suffisantes. »

III.4 l'organigramme de ce projet :



Figure(III.6): l'organigramme de ce projet

III.5La programme de cette projet :

```
;----------  Exemple d'application avec un PIC : Les feux tricolores  ----------
; Titre : Feux rouge
; Date : 1 Mai 2014
 ; Auteur : Lekhoua salem et Amara sayad
 ; PIC utilisé : PIC 16 F 84
 ; On réalise des feux rouge sur les broches RB0 à RB5  d' un PIC 16 F 84
; le quartz est de 4 Mhz , on effectue une tempo longue  environ égale à 4 secondes et
 ; une tempo courte environ égale à 1.5 secondes.
```

```
 ; un bouton marche sur le port A permet de lancer l' application
 ; RB0=rouge1 RB1=orange1 RB2=vert1
 ; RB3=rouge2 RB4=orange2 RB5=vert2
 ;------------  Directive d' assemblage pour PLAB  ---------------
       list      p=16f84A
       #include p16f84A.inc
       __config H'3FF9'
 ;------------  Définition des constantes  ---------------
 #define inter0 0        ; bouton marche
#define inter1 1         ; bouton clignotement orange
;------------  Définition des registres temporaires  ---------------
 retard1     EQU      0x0C     ; le registre temporaire retard1 se trouve à l' adresse 0C
retard2     EQU      0x0F     ; le registre temporaire retard2 se trouve à l' adresse 0F
 retard3     EQU      0x10     ; le registre temporaire retard3 se trouve à l' adresse 10
 ;------------  Init des ports A et B  ---------------
ORG 0
 bsf STATUS,5                  ; on met à 1 le 5eme bit du registre status pour accéder
                               ; à la 2eme page mémoire ( pour trisa et trisb )
 MOVLW 0x00                    ; on met 00 dans le registre W
MOVWF TRISB                    ; on met 00 dans le port B il est programmé en sortie
MOVLW 0x1F          ; on met 1F dans le registre W
MOVWF TRISA                    ; on met 1F dans le port A il est programmé en entrée
 bcf STATUS,5                  ; on remet à 0 le 5eme bit du registre status pour accéder
                               ; à la 1eme page mémoire
 ;------------  Init des feux ROUGE1 et ROUGE2  ---------------
MOVLW  B'00001001'     ; on met 09 dans le registre W ( Rouge1 et Rouge2 )
```

```
 MOVWF  PORTB        ; on met W sur le port B ( led )
;--------------------  Programme principal  ----------------------
debut
 MOVLW  B'00001001'        ; on met 09 dans le registre W ( Rouge1 et Rouge2 )
 MOVWF  PORTB              ; on met W sur le port B ( led )
btfss  PORTA,inter0   ; interrupteur 0 ( marche ) appuyé ? si oui on continu sinon
                                  ;va à debut
 goto debut
 ret_cli
 btfsc PORTA,inter1           ; interrupteur 1 ( clignotant ) appuyé ? si oui on
                                  ;va à clignote
 goto clignote
 MOVLW  B'00001001'        ; on met 09 dans le registre W ( Rouge1 et Rouge2 )
MOVWF  PORTB              ; on met W sur le port B ( led )
;---------------- Chargement de la temporisation ---------------------
CALL  tempo                 ; on appel la temporisation 1 ( longue )
MOVLW  B'00001100'        ; on met 0C dans le registre W ( Vert1 et Rouge2 )
MOVWF  PORTB              ; on met W sur le port B ( led )
CALL  tempo                 ; on appel la temporisation 1 ( longue )
MOVLW  B'00001010'        ; on met 0A dans le registre W ( Orange1 et Rouge2 )
MOVWF  PORTB              ; on met W sur le port B ( led )
CALL  tempo2                ; on appel la temporisation courte
MOVLW  B'00001001'        ; on met 09 dans le registre W ( Rouge1 et Rouge2 )
MOVWF  PORTB              ; on met W sur le port B ( led )
CALL  tempo2                ; on appel la temporisation courte
MOVLW  B'00100001'        ; on met 21 dans le registre W ( Rouge1 et Vert2 )
```

```
MOVWF  PORTB            ; on met W sur le port B ( led )
CALL  tempo             ; on appel la temporisation longue
MOVLW  B'00010001'      ; on met 11 dans le registre W ( Rouge1 et Orange2 )
MOVWF  PORTB            ; on met W sur le port B ( led )
CALL  tempo2            ; on appel la temporisation courte
GOTO  debut             ; retour au début du programme
;----------  Programme de temporisation longue  ----------
tempo
MOVLW 0xFF          ; on met ff dans le registre W
MOVWF retard1       ; on met W dans le registre retard1
MOVWF retard2       ; on met W dans le registre retard2
MOVLW 0x12          ; on met 12 dans le registre W
MOVWF retard3       ; on met W dans le registre retard3
attente
DECFSZ retard1,F    ; on décrémente retard1 et on saute la prochaine instruction si
GOTO  attente       ; le registre retard1 = 0 sinon retour à attente
movlw 0xFF          ; on recharge retard1
movwf retard1
DECFSZ retard2,F    ; on décrémente retard2 et on saute la prochaine instruction si
GOTO  attente       ; le registre retard2 = 0 sinon retour à attente
movlw 0xFF          ; on recharge retard2
movwf retard2
DECFSZ retard3,F    ; on décrémente retard3 et on saute la prochaine instruction si
GOTO  attente       ; le registre retard3 = 0 sinon retour à attente
RETURN              ; retour au programme principal après l 'instruction CALL
;----------  Programme de temporisation courte  ----------
```

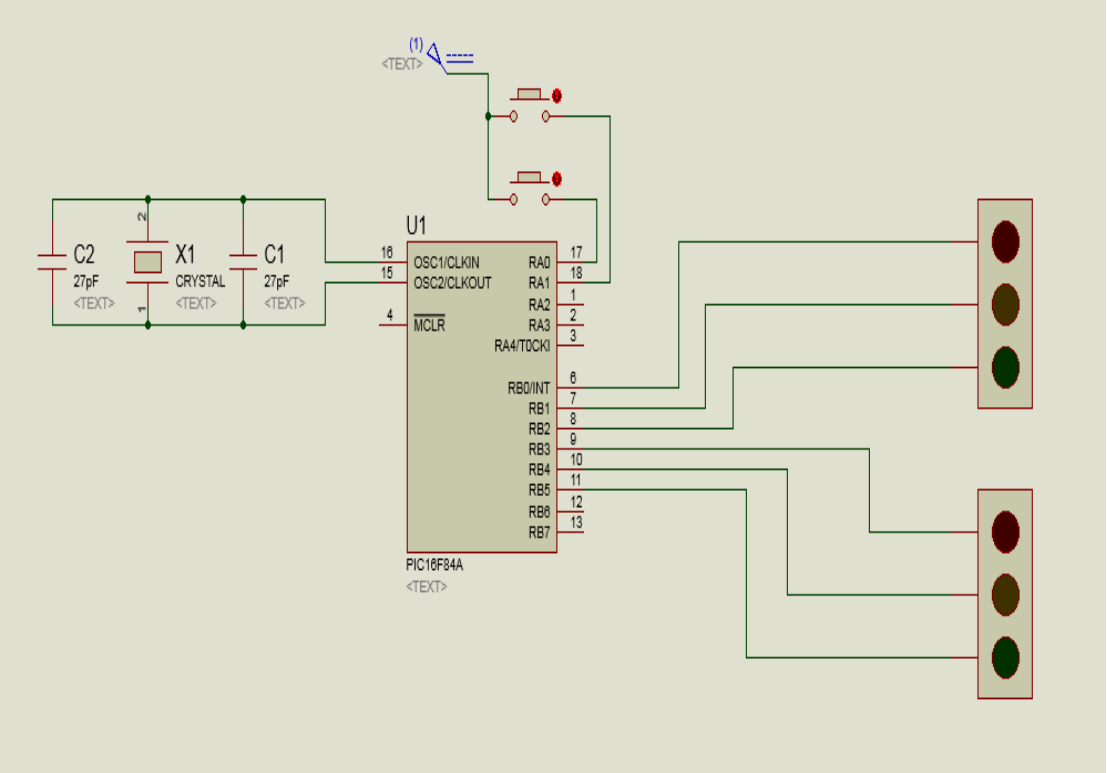
```
tempo2
MOVLW 0xFF          ; on met ff dans le registre W
MOVWF retard1       ; on met W dans le registre retard1
MOVWF retard2       ; on met W dans le registre retard2
MOVLW 0x07          ; on met 7 dans le registre W
MOVWF retard3       ; on met W dans le registre retard3
attente2
DECFSZ retard1,F    ; on décrémente retard1 et on saute la prochaine instruction si
GOTO  attente2      ; le registre retard1 = 0 sinon retour à attente2
movlw 0xFF          ; on recharge retard1
movwf retard1
DECFSZ retard2,F    ; on décrémente retard2 et on saute la prochaine instruction si
GOTO  attente2      ; le registre retard2 = 0 sinon retour à attente2
movlw 0xFF          ; on recharge retard2
movwf retard2
DECFSZ retard3,F    ; on décrémente retard3 et on saute la prochaine instruction si
GOTO  attente2      ; le registre retard3 = 0 sinon retour à attente2
RETURN
clignote
MOVLW  B'00010010'  ; on met 12 dans le registre W ( Orange1 et Orange2 )
MOVWF  PORTB        ; on met W sur le port B ( led )
CALL  tempo2        ; on appel la temporisation courte
MOVLW  B'00000000'  ; on met 00 dans le registre W ( aucune led )
MOVWF  PORTB        ; on met W sur le port B ( led )
CALL  tempo2        ; on appel la temporisation courte
goto ret_cli
```

END

III.6 câblage du Circuit :

Le schéma ci-dessous présente le circuit des feux rouges dont le microcontrôleur pic 16F84 à programmer. En tout cas ce circuit comprend un pic 16F84, LEDs de lumière, bouton poussoir, oscillateur 4MHz composant par quartz de 4MHz et deux capacité de 27pF, générateur de tension continu.

En effet, notre ici réside à la programmation du pic 16F84 pas la mise en place de ce circuit dans le laboratoire.



Figure(III.7): la circuit de cette projet

III.7Conclusion:

Dans ce dernier chapitre on essayé de faire exploiter tout ce qu'on a appris au premier et au deuxième chapitre dans un projet de feux rouges à l'aide d'un logiciel de simulation MPLAB. Sachant que un organigramme synoptique qui atteint notre objective a été présenté, suivi par notre programme que l'on a fait en assembleur pour cet objet. Finalement, et comme travail à suivre, on voudrait prochainement concrétisez ce projet au niveau de laboratoire, cela reste comme perspectif à toucher prochainement si Allah le veut.

Conclusion générale

CONCLUSION GENERALE

Dans cette étude, notre objectif était la mise en œuvre et l'apprentissage de la programmation du microcontrôleur pic 16F84 suivant les contraintes et les besoins désirés.

Nous avons étudié le pic 16F84 car l'architecture de ce microcontrôleur est plus simple, où on a essayé de focaliser la lumière sur son architecture générale afin de faciliter la compréhension de son programmation, comme la composition intérieur, et les différentes types de mémoires, et la composition extérieur et nombre de broches, les entrées et les sorties…etc. Après, on a vu les différentes instructions qui programment PIC 16F84 et les types d'adressage .

En suite nous avons manipulé tout ce qu'on a appris à la programmation dans un projet d'un Feux de circulation à base du pic.

Afin de contribuer à ce projet, notre premier but est essayer de faire programmer un microcontrôleur 16F84 qui est utilisé comme gérant de feux de rouges. Sachant que afin de déboguer notre programme on a utilisé logiciel MPLAB.

Notre perspective réside dans l'utilisation et la concrétisation de ce projet réellement ce projet au niveau du laboratoire.

Références bibliographiques

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] http://www.abcelectronique.com/bigonoff , Première partie – Révision 34 démarrer les PIC® avec le PIC16F84.

[2] http://www.electronique-magazine.com, l'électronique par la pratique n°5, Microcontrôleurs Pic de la théorie aux applications $4^{éme}$ partie l'architecture interne du PIC. 16F84.

[3] http://fr.scribd.com/doc/144514285/Etude-Et-Realisation-d-Un-Robot-Suiveur-de-Ligne-Part2 vue 07/03/2014, 17 :00h

[4] avrj.cours.pagesperso-orange.fr...16F84_2004_3.pdf , vue 23/05/2014 ,10:00h

Résumé

Le domaine de l'utilisation du microcontrôleur est très vaste, c'est pour cela nous l'avons étudié à travers sa construction intérieur et extérieure qui représente la base principal. Nous avons également pu déboguer notre programme à travers un logiciel qui s'appelle MPLAB qui est capable de détecter les erreurs qui se trouvent dans un programme.

Finalement, nous avons programmé le pic 16F84 pour un projet de feux de circulation en faisant le débogage du programme par le logiciel MPLAB.

Mots clés: pic16F84, assembleur, µc, MPLAB

Abstract

The domain of the microcontroller is very large, for this reason we have studied it through its internal and external construction and which represents the main base. We were also able to debug our program through a program called MPLAB which is able to detect errors that are in a program.

Finally, we programmed the pic 16F84 for a project of traffic lights by debugging the program by the MPLAB software.

Keywords: pic16F84, assembler, µc, MPLAB

الملخص

ان مجال استعمال المتحكمات جد واسع لهذا السبب قمنا بدراسة من خلال تركيبته الداخلية و الخارجية والتي تمثل القاعدة الاساسية كما تمكنا ايضا من تنفيذ بعض التعليمات بواسطة برنامجMPLAB وهو المسؤول على كشف الاخطاء الموجودة في البرنامج.

اخيرا بإجراء تجربة بسيطة المتمثلة في برمجة المتحكم في اطار مشروع التحكم في اشارات المرور عن طريق تنفيذ التعليمات عن برنامج MPLAB .

الكلمات المفتاحية:,pic 16 f84, لغة التجميع,µc , MPLAB