République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique

12/03

Ecole Nationale Polytechnique

Département Electronique

Projet de Fin d'Etudes

en vue de l'Obtention du Diplôme d'Ingénieur d'Etat
en Electronique

Thème

Etude et réalisation d'un programmateur horaire à base d'un microcontrôleur PIC 16F84

Proposé et Dirigé par :
M.HADDADI
C.LARBES
Maîtres de conférences

Présenté par :

Mr.BOULEROUAH Aoumeur

Promotion : Septembre 2003

Remerciement

En premier, je tiens à remercie tous les enseignants du département Electronique qui ont contribué à ma formation.

Mes plus grands remerciement vont à mes promoteurs Mr.HADDADI et Mr.LARBAES pour m'avoir guidé et suivi de manière régulière durant toute l'étude, ainsi que les facilités qu'ils m'ont accordé pour mener à bien mon travail.

Nos remerciements à l'ensemble des enseignants et les membres du personnel de notre Département pour leurs aides et leur disponibilité tout au long de notre cursus à l'ENP.

Enfin, nous remercions toute personne ayant contribué de prés ou de loin à l'élaboration de ce travail.

TABLE DES MATIERES

CHAPITRE I

I) INTRODUCTION GENERALE

Les microcontrôleurs envahissent notre environnement sans que nous le sachions. Ces petits composants se retrouvent de plus en plus dans tous les matériels que nous utilisons quotidiennement, machine à laver, ordinateur, téléviseur...etc. Dotés d'une logique programmée, ils sont capables de réagir à l'environnement un peu à la manière d'automates programmables. Mais leurs propriétés ne se limitent pas à offrir un certain nombre d'entrées sorties logiques. Ils sont souvent dotés de fonctions supplémentaires telles que convertisseurs analogiques numériques, horloges temps réel, comptage rapide etc. Ainsi ils permettent de faire l'économie de nombreux circuits périphériques ainsi que la simplicité des montages. Un microcontrôleur peut donc gérer seul une application, sans faire appel forcement à d'autres circuits associés.

I-I) Les programmateurs horaires

Parfois la réalisation automatique de certaines tâches pendant des durées programmées est envisageable quand la présence de l'homme n'est pas nécessaire. Ces tâches sont donc gérées par un dispositif appelé programmateur horaire, ce dernier pouvant être mécanique ou électronique.

Les programmateurs horaires sont utilisés dans différents domaines. On trouve des programmateurs qui permettent le fonctionnement des chauffages pendant la nuit, quand les personnes sont présentes. Cela permet une diminution de la consommation de l'énergie électrique. Ce même système est utilisé aussi pour l'éclairage. Il existe aussi dans le jardinage des appareils pour l'arrosage équipés de ce système qui assure une bonne gestion de l'eau ainsi qu'un fonctionnement automatique. Cela peut être très utile pour les gens qui sont passionnés de jardinage et de voyage. Une autre application intéressante concerne les prises d'électricité programmables destinées aux appareils ménagers.

La composition de ces programmateurs est très simple. Ils sont dotés d'une mémoire qui stocke les informations concernant les différents instants de travail ou d'arrêt, d'une horloge avec laquelle il compare ces informations stockées et d'une interface qui agit sur les systèmes extérieurs.

Plusieurs fabricants proposent ces programmateurs horaires. On peut citer le programmateur "digital cubique" fabriqué par KELKOO. Il assure la programmation quotidienne ou hebdomadaire d'appareils ménagers ou de lampes, de puissance maximum de 3500W, avec 8 programmes par jour ou par semaine, en fonctionnement manuel ou programmé.

L'application que nous allons étudier a pour but la réduction de la consommation d'énergie électrique, principalement pour les équipements alimentés par énergie solaire. Car cette énergie qui provient des cellules photovoltaïques n'est fournie que pendant le jour, donc sur une durée limitée. Pour une bonne gestion de cette énergie, il faut programmer le fonctionnement de chaque appareil ou charge (TV, lampes d'éclairage, ordinateur,...etc) pendant une durée déterminée, au moment où il est nécessaire. C'est à quoi notre programmateur horaire sera destiné.

CHAPITRE II

II) DESCRIPTION ET ETUDE DU CAHIER DES CHARGES

II-1) Le cahier des charges initial

Le travail demandé est la réalisation d'un dispositif qui permet de mettre en service ou d'arrêter à des moments programmés (minute, heure , jour ,mois) 8 appareils, pendant une année, avec ou sans calendrier externe. La programmation de ce dispositif peut être manuelle avec des boutons-poussoirs ou par un ordinateur(liaison RS232).

II-2) Evaluation du cahier des charges

L'étude du cahier des charges initial permet de faire plusieurs remarques concernant notre application :

- Le nombre de données qu'il faut stocker est très grand. On peut calculer la taille de notre mémoire pour une seule programmation par jour et pour un seul appareil de la façon suivante: On a 8 données par jour (minute, heure, jour, mois) pour l'allumage et (minute, heure, jour, mois) pour l'extinction, donc cela nous donne pendant une année 8 x 365 = 2920 données à mémoriser, donc une mémoire de plus de 2Ko. cela nous donne pour les 8 appareils 8 x 2K = 16Ko.
- Notre programmateur horaire peut être programmé manuellement, mais cette programmation est très fastidieuse à cause du nombre élevé de données à stocker.
  Il n'est pas pratique de programmer 32Ko de données manuellement.

On peut en conclure que pour une simplicité de programmation et une réduction de la taille des circuits, il faut réduire le nombre d'appareils à un seul et le nombre de jours à programmer à un mois (30 jours). Ceci permet de réduire le nombre de données à 6 par jour (minute, heure, jour) pour l'allumage et l'extinction. An total on aura 6 x 30 = 180 octets.

Pour la mise en œuvre de ce dispositif on a le choix entre deux méthodes différentes :

- La logique câblée : cette méthode consiste à réaliser toutes les fonctions du système avec des circuits discrets (bascules, compteurs, horloge,....etc.).
  Cette méthode a plusieurs inconvénients :

- Un très grand nombre de circuits intégrés, occasionne un encombrement et exige une très grande surface.
- La difficulté de la réalisation pratique des circuits imprimés
- Un coût très élevé

- La logique programmée: cette méthode consiste à utiliser des circuits programmables (microprocesseur, microcontrôleur). Ces derniers peuvent remplacer plusieurs circuits à la fois par un simple programme introduit dans leur mémoire.

Cette méthode nous offre plusieurs avantages :

- Réduction du nombre de circuits intégrés, donc pas d'encombrement.
- Facilité de la réalisation.
- Intégration de plusieurs fonctions à la fois
- Economie

II-3) Etude et choix du microcontrôleur

II-3-1) Etude du microcontrôleur

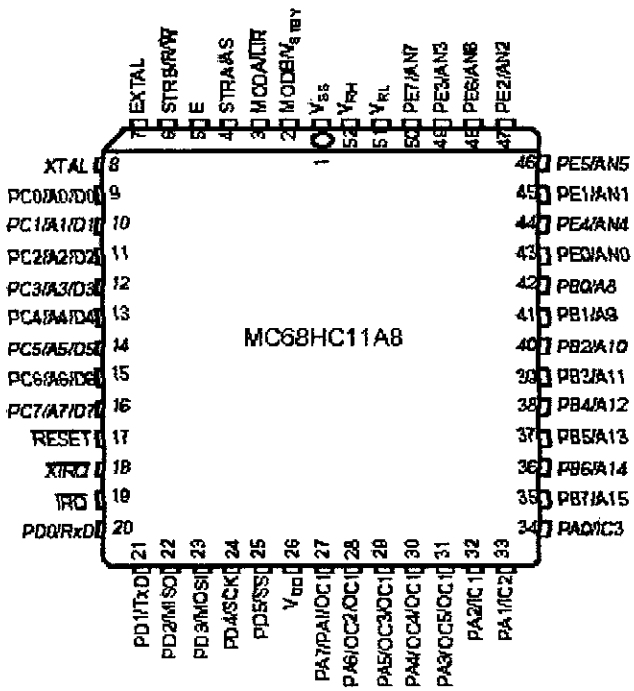
On adoptera donc cette seconde méthode (la logique programmée) pour notre application. Pour cela on va étudier 3 types de microcontrôleurs de deux familles différentes, les pic16F876 et pic16F84 de la famille MICROCHIP et le microcontrôleur 68HC11 de la famille MOTOROLA. Le choix de ces derniers microcontrôleurs parmi d'autres existant sur le marché est dû à leur disponibilité au niveau de notre laboratoire. Cette étude nous permettra de choisir le microcontrôleur qui convient le mieux pour notre application.

II-3-2) Le microcontrôleur 68HC11

A- Présentation générale

Le 68HC11 est un microcontrôleur 8 bits fabriqué par MOTOROLA qui adresse jusqu'à 64Ko de mémoire. Le jeu d'instructions est dérivé de celui de ses ancêtres : 6801, 6805 et autres 6809. Il est décliné en différentes versions comportant chacune des tailles de mémoires différentes (RAM, EPROM, EEPROM). La consommation du circuit est d'environ 15mA en fonctionnement normal (en mode SINGLE) ou 27mA (en mode EXPANDED), de 6mA en mode WAIT et de 50µA en mode STOP.

Figure 1 :Brochage des 68HC11 Ax et Ex en boîtier PLCC 52

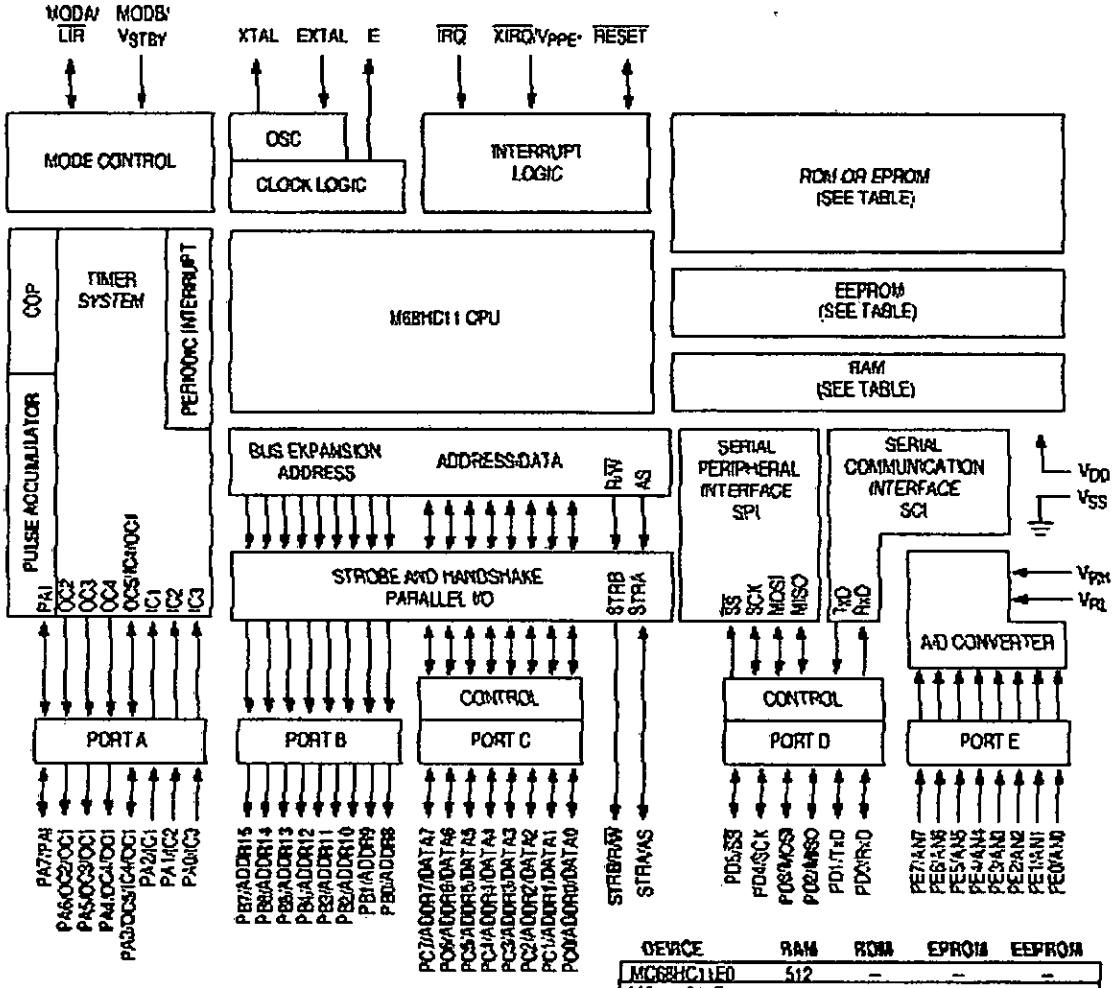


B- Bloc fonctionnel :

Le 68HC11 est caractérisé par les éléments suivants:

- 8Ko de ROM.
- 512 octets de EEPROM.
- 256 octets de RAM.
- Un timer 16 bits (3 entrées de capture, 5 sorties de comparaison).
- 2 accumulateurs 8 bits.
- Une liaison série asynchrone (SCI).
- Une liaison série synchrone (SPI).
- Un convertisseur Analogique/Numérique 8 bits, 8 entrées multiplexées.
- Circuit d'interruption temps réel.
- Circuit oscillant externe (quartz 8 MHz dans notre application).

Figure 2 : Architecture interne de µC 68HC11



DEVICE	RAM	ROM	EPROM	EEPROM
MC68HC11E0	512	–	–	–
MC68HC11E1	512	–	–	512
MC68HC11E8	512	12K	–	–
MC68HC11E9	512	12K	–	512
MC68HC711E9	512	–	12K	512
MC68HC11E20	768	20K	–	512
MC68HC711E20	768	–	20K	512
MC68HC811E2	256	–	–	2048

C- L'unité arithmétique et logique :

Deux accumulateurs 8 bits (A et B) pouvant être concaténés pour obtenir un registre 16 bits (D), 2 registres d'index 16 bits (X et Y) pouvant pointer sur n'importe quelle case mémoire dans les 64Ko. Des instructions sur 16 bits : addition, soustraction, division, décalage, rotation et multiplication 8 * 8.

D- Les versions les plus répandues :

HC11 Ax : 8Ko ROM , ROMless , 256 RAM, 512 EEPROM, SPI, SCI, 3 IC 5 OC, 8 voies 8 bits A/D.

HC11 Dx : Le modèle le plus petit et le moins cher de la famille des HC11. ROMless , 4Ko ROM, 4Ko EPROM - Pas de A/D ni d'EEPROM. Comporte les timers et les liaisons séries SCI et SPI.

HC11 Ex : Identique à la série A sauf la taille de l'EEPROM. Pin compatible, à la différence de la pin PA4 qui peut être utilisée en Input Capture ou en Output Compare.

HC11 Fx : ROMless, les adresses et les données ne sont pas multiplexées. Comporte 4 Chip Select. 1Ko RAM, 512 octets EEPROM.

HC11 Gx : 16Ko ROM/EPROM, les adresses et les données ne sont pas multiplexées. 512 RAM, 4 voies PWM, 10 Bit ADC, 2 timers 16 bits séparés.

HC11 K4 : 24Ko ROM/EPROM, 1Mb d'addressage. les adresses et les données ne sont pas multiplexées. 4 Chip Select, 8 voies 8 bit ADC. 4 voies PWM, 768 octets RAM.

HC11 L6 : Dérivé du 68HCllE9. 16Ko ROM, 1 port d'IO supplémentaire

II-3-3) Les PICs de la famille MICROCHIP

A- Présentation générale

Le PIC est un microcontrôleur, ou un système minimal, c'est à dire une unité de calcul de type microprocesseur à laquelle on a ajouté des périphériques internes permettant de réaliser des montages sans nécessiter l'ajout de composants externes.

Le PICs sont des composants dits RISC (*Reduced Instructions Set Computer*), ou encore composants à jeu d'instructions réduit, par opposition aux composants dits CISC (*Complex Instructions Set Computer*). Ils sont aussi 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 DYNAMIQUES (comme les microprocesseurs de nos ordinateurs), donc la fréquence d'horloge doit rester dans des limites précises.

Les PICs disposent d'un jeu de 35 instructions seulement, permettant un décodage plus facile et plus rapide, et une vitesse de fonctionnement bien supérieure à celle des microprocesseurs CISC.

La famille des PICs (dont le constructeur est *Microchip*) est subdivisée en 3 grandes familles : La famille *Base-Line*, qui utilise des mots d'instructions de 12 bits, la *famille Mid-Range*, qui utilise des mots de 14 bits et dont fait partie le PIC16F84, et la famille *High-End*, qui utilise des mots de 16 bits.

A-1) Identification d'un PIC

Pour identifier un PIC on utilise son numéro, qui contient :

- Deux chiffres pour indiquer la catégorie de PIC.
- Une ou deux lettres qui indiquent le type de la mémoire .
- Un suffixe "-xx" qui donne la vitesse maximale de PIC.

Exemple:

16 F 84 -04

16 : PIC de la catégorie Mid-range

F : mémoire de type FLASH

84 : la référence du circuit.

04 : la vitesse max est de 4MHz.

A-2) Organisation des instructions

Les PICs ont un jeu de 35 instructions décomposées en 4 types :

1-Les instructions orientées octet :

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

- 6 bits pour l'instruction : logique, car comme il y a 35 instructions, il faut 6 bits pour pouvoir les coder toutes
- 1 bit de destination(d) pour indiquer si le résultat obtenu doit être conservé dans le registre de travail de l'unité de calcul (W pour Work) ou sauvé dans l'opérande (F pour File).
- Reste 7 bits pour encoder l'opérande (File).

Exemple : *swap* PORTB,w

2-Les instructions orientées bits :

Ce sont des instructions destinées à manipuler directement des bits d'un registre particulier.

Elles sont codées de la manière suivante :

- 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), et de nouveau :
- 7 bits pour indiquer l'opérande.

Exemple : *bsf* PORTA,1

3- Les instructions générales :

Ce sont les instructions qui manipulent des données qui sont codées dans l'instruction directement. Elles sont codées de la manière suivante :

- L'instruction est codée sur 6 bits
- Elle est suivie d'une valeur IMMEDIATE codée sur 8 bits (donc de 0 à 255).

Exemple : *addlw* 0xA5

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 manières suivante :

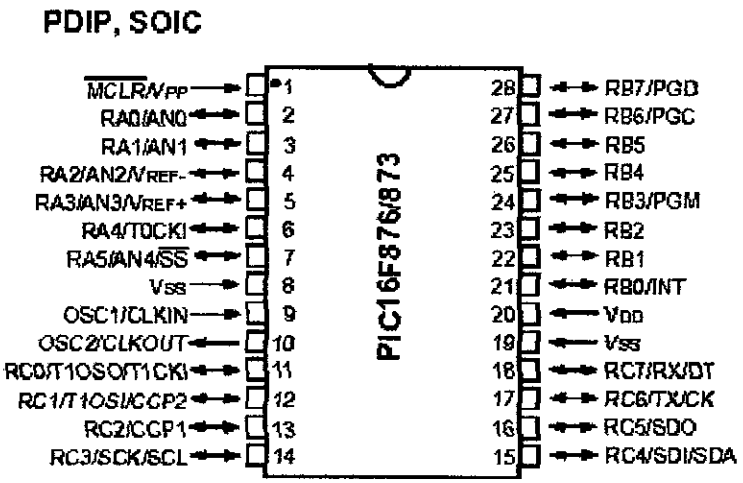
- Les instructions sont codés sur 3 bits
- La destination codée sur 11 bits

Exemples : *goto* etq1
call affichage

B- <u>Le PIC 16F876</u>

Le pic16F876 est un microcontrôleur 8 bits fabriqué par la société américaine Arizona Microchip Technology. Son boîtier est disponible en version standard (PDIP 28 broches) ou en version "composants de surface" (SOIC). Il fait partie de la grande famille des PICs Mid-Range, donc les instructions sont codées sur un mot de 14bits.

Figure 3 : Brochage de µC PIC16F876

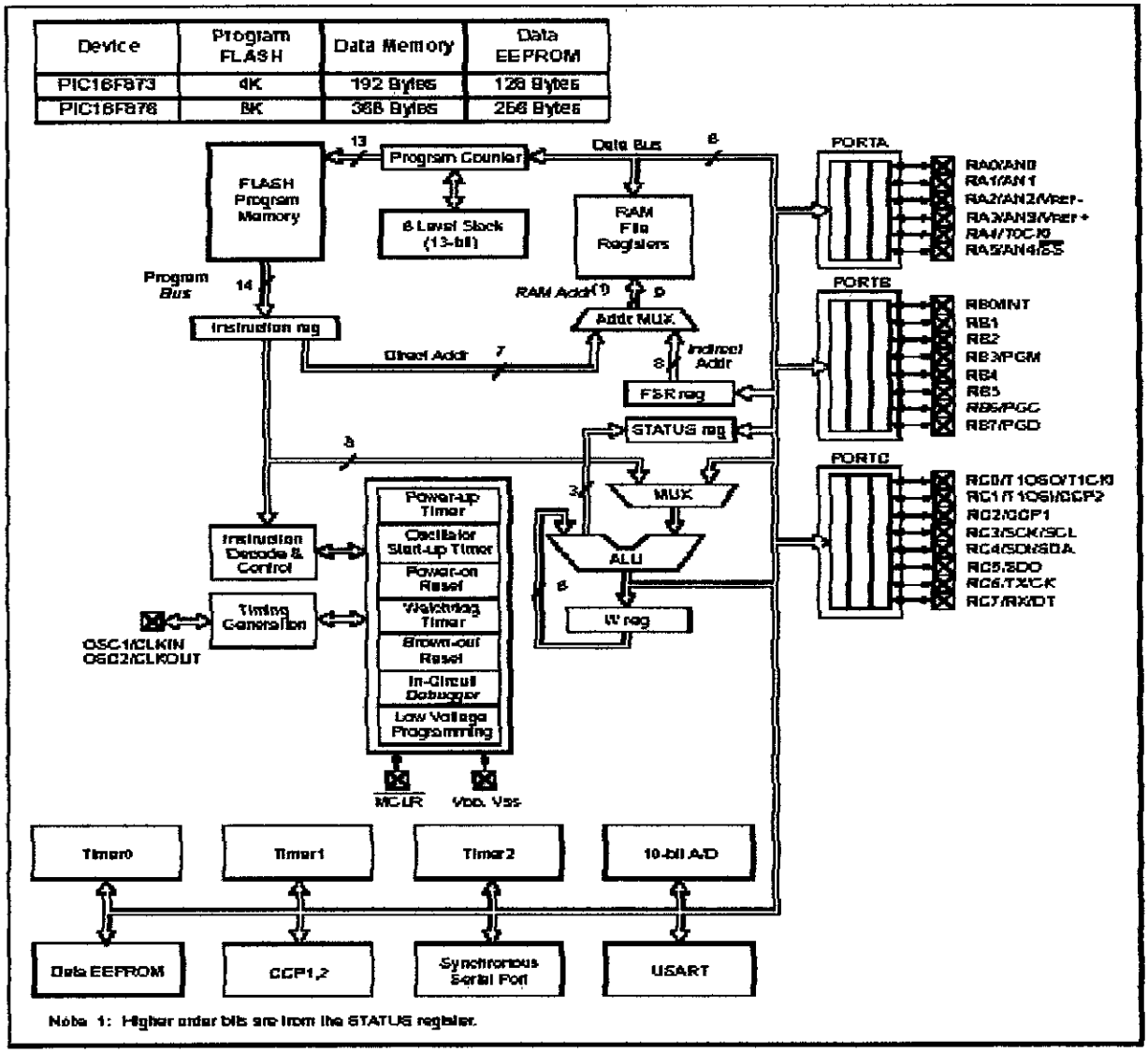


Ces principales caractéristiques sont:

- un jeu de 35 instructions.
- Un CPU de type RISC (Reduced Instructions Set Computer).
- Exécute chaque instruction en 1 seul cycle, sauf les sauts..
- Une vitesse d'horloge de 20MHz.
- Une mémoire de programme de 8K.
- Une mémoire RAM de 368 octets .
- Une mémoire EEPROM de 256 octets.
- Une liaison série asynchrone (SCI).
- Une liaison série synchrone (SPI).
- 3 Timers, Timer0 et Timer2 sur 8 bits, Timer1 sur 16 bits.
- Un Timer Watchdog.
- Un convertisseur analogique/numérique de 10 bits.
- 14 source d'interruptions.

Son architecture interne est représentée dans la figure ci-dessous

Figure 4 : Architecture interne du PIC 16F876



B-1) Organisation de la mémoire.

La mémoire de PIC16F876 est divisée en trois parties. La mémoire programme et la mémoire de données qui ont des bus d'adresses et de données différents. Enfin une mémoire EEPROM.

a) La mémoire programme.

Le PIC16F876 a un Compteur Programme (PC) de 13 bits capable d'adresser une mémoire programme de 8K. C'est dans cette zone de mémoire qu'on stocke nos programmes.

Elle contient un vecteur de reset situé à l'adresse 0000h et un vecteur d'interruption situé à l'adresse 0004h.

b) La mémoire RAM.

La mémoire RAM est celle que nous utilisons pour stocker les variables de nos programmes, c'est une mémoire non rémanente, ses données sont perdues à chaque coupure de courant. Elle contient des registres spéciaux et des registres à usage général. Cette mémoire est d'une capacité de 368 octets répartie de la manière suivante :

1) 80 octets en banque 0, adresses 0x20 à 0x6F
2) 80 octets en banque 1, adresses 0xA0 à 0XEF
3) 96 octets en banque 2, adresses 0x110 à 0x16F
4) 96 octets en banque 3, adresses 0x190 à 0x1EF
5) 16 octets communs aux 4 banques, soit 0x70 à 0x7F = 0xF0 à 0xFF = 0x170 à 0x17F = 0x1F0 à 0x1FF

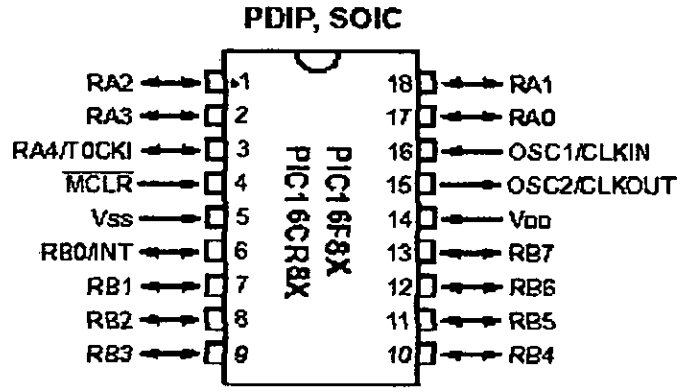
c) La mémoire EEPROM.

La mémoire EEPROM (Electrical Erasable Programmable Read Only Memory) est constituée de 256 octets, accessible en lecture et en écriture. Elle permet la conservation des données en permanence sans que le circuit ne soit alimenté.

C- Le PIC 16F84

Le PIC 16F84 est un microcontrôleur 8bits, doté d'une architecture *HARVARD* qui lui confère une vitesse extraordinaire. Il appartient à la catégorie Mid-range, sa simplicité lui donne un grand champ d'application. Son boîtier est un DIL(Dual In Line) de 18 broches.

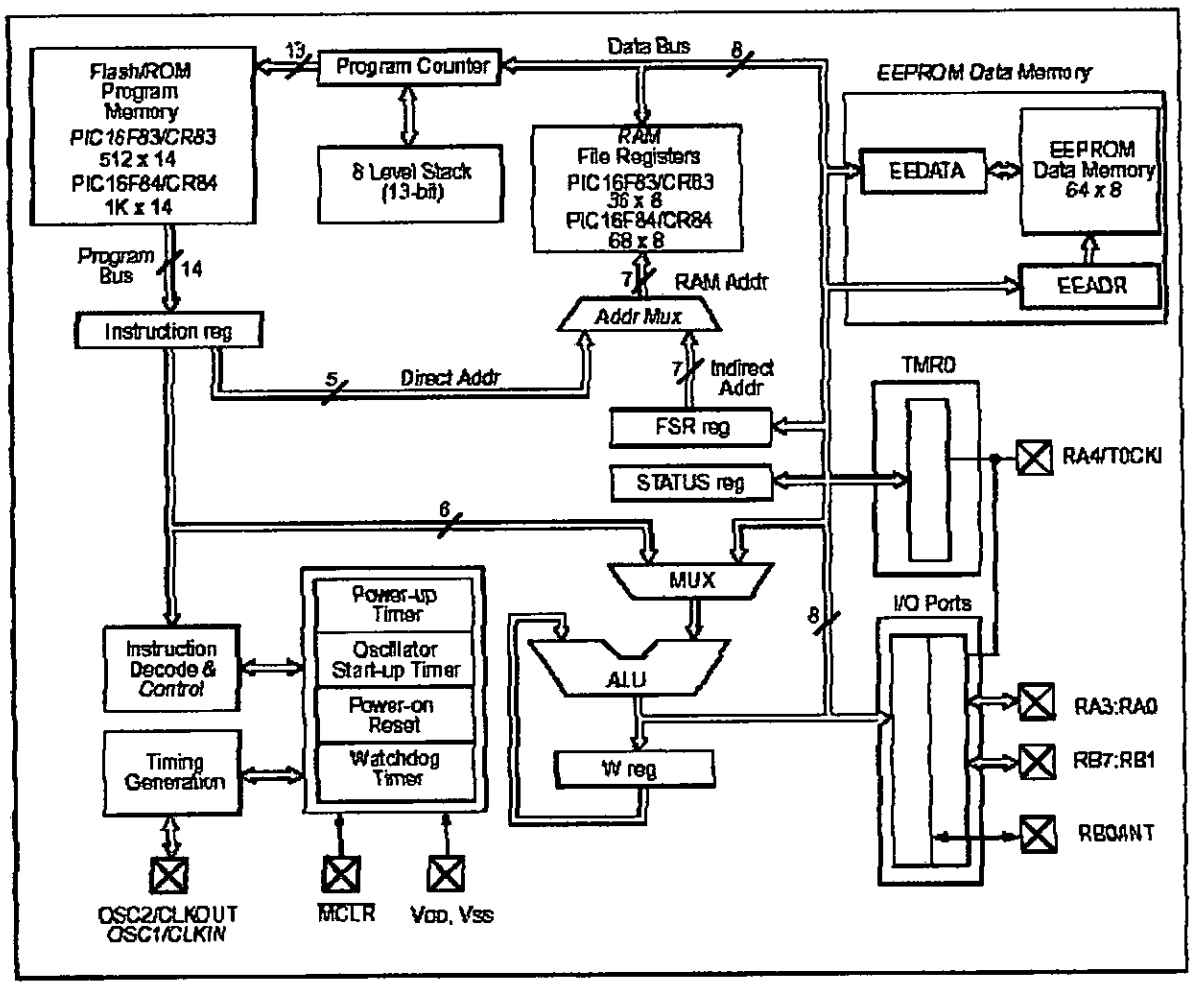
Figure 5 : Brochage de µC PIC16F84



Ces caractéristiques principales sont :

- un jeu de 35 instructions.
- Un CPU de type RISC (Reduced Instructions Set Computer).
- Exécute chaque instruction en 1 seul cycle, sauf les sauts..
- Une vitesse d'horloge de 4MHz.
- Une mémoire de programme de 1K.
- Une mémoire RAM de 68 octets .
- Une mémoire EEPROM de 64 octets.
- Un timer de 8bits.
- Un timer watchdog.

Figure 6 : Architecture interne du PIC 16F84

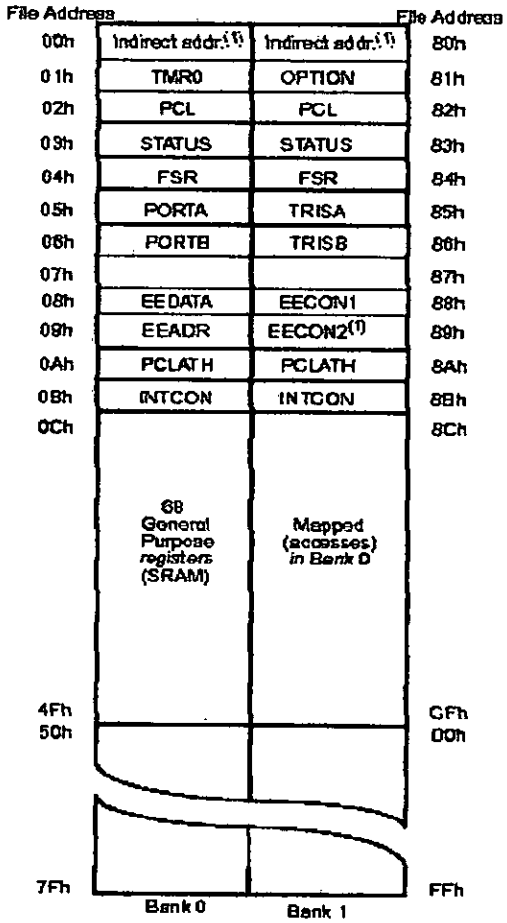


C-1) organisation de la mémoire

La mémoire de PIC16F84 est divisée en 3 partie disjointes qui assurent le stockage de deux types de données; les données non rémanentes par une mémoire RAM qui est constituée de 68 octets, les instructions et les données rémanentes par une mémoire programme d'un capacité de 1K mot de 14 bits et d'une mémoire EEPROM de 64 octets.

La mémoire RAM de PIC16F84 est subdivisée elle aussi en deux banques. Chacune d'elles contient des registres spéciaux qui provoquent un fonctionnement spécial du PIC ou la mise en service d'une fonction particulière.

Figure 7 :Organisation de la RAM



II-3-4) Le choix du microcontrôleur

L'étude de ces 3 microcontrôleurs permet de dire que le µC qui convient le mieux pour notre application et le µC PIC16F84. La raison de ce choix, en est que les deux autres µC (PIC16F876 et 68HC11) ont deux principaux inconvénients :

- Ils disposent de différentes fonctions qui sont inutiles pour notre application, par exemple le convertisseur A/N.
- ils ont un nombre très important de broches (52 pattes pour le µC 68HC11, 28 pattes pour le µC PIC16F876),ce qui encombre notre circuit.

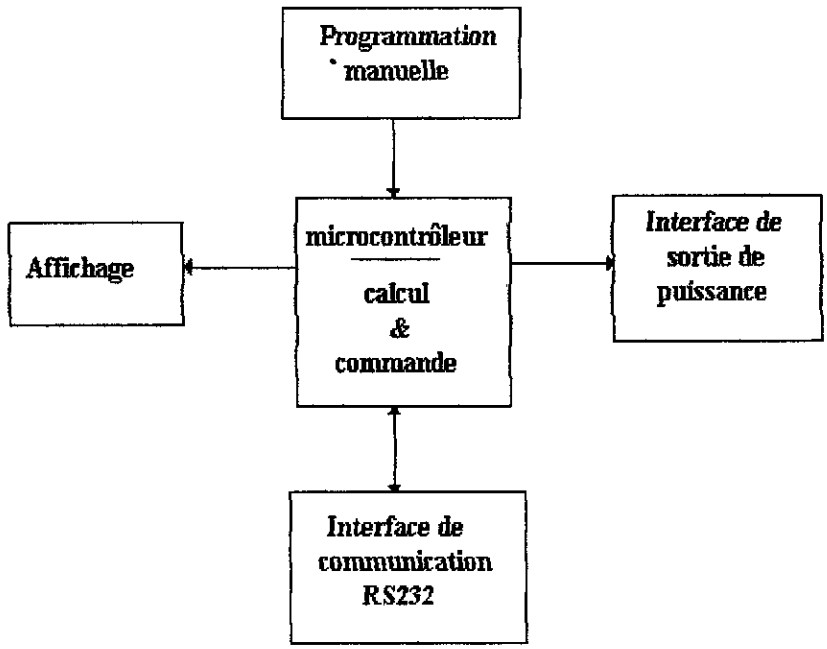
La simplicité du PIC16F84 et l'espace mémoire disponible (68 "RAM" + 64 "EEPROM" =132 octets) nous incitent à le préférer aux deux autres µC.

CHAPITRE III

III) ETUDE DE LA MISE EN OEUVRE (LE HARDWARE)

L'étude précédente nous a permis de choisir le µC adéquat pour notre dispositif. En tenant compte de toutes les fonctions décrites dans notre cahier des charges, notre dispositif peut être divisé en 5 parties différents selon le schéma synoptique suivant :

Figure 8 :Schéma synoptique de la carte



III-1) LE MICROCONTROLEUR PIC16F84:

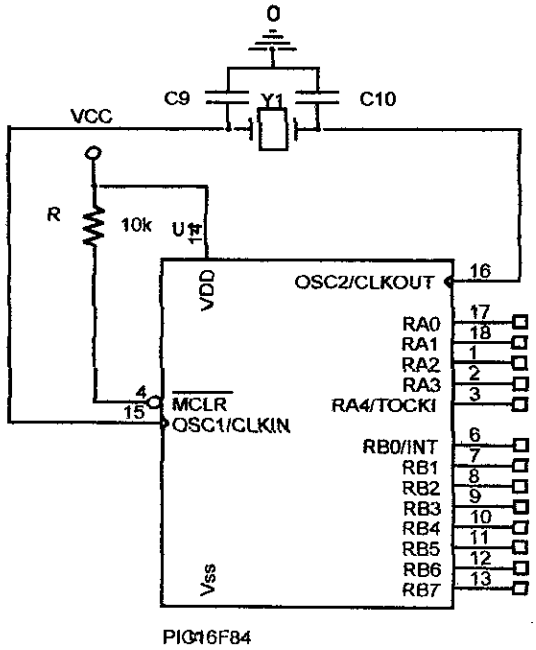
Il représente l'unité principale de notre dispositif et commande toutes les autres parties de ce système (les afficheurs, l'interface de puissance, l'interface de communication) via ses ports de sortie A et B. Il calcule aussi le temps pour avoir le rôle d'une horloge qui cadence le travail de notre dispositif.

III-1-a) LE SCHEMA ELECTRIQUE

Son schéma électrique est composé des éléments suivants:

- Un quartz de 4MHz.
- Deux condensateurs céramiques de 22 pF.
- Une résistance qui maintient MCRL à l'état haut (désactivation du reset).

Figure 9 :Brochage du PIC16F84



III-2) LA PROGRAMMATION MANUELLE

Cette partie nous donne la possibilité de programmer manuellement notre dispositif.

Il existe deux genres de programmation :

- Programmation de l'horloge: qui nous permet le réglage du temps (minutes, heures , jours).
- Programmation des instants de marche et d'arrêt.

Ces deux types de programmation sont activés par des interruptions. Ces dernières sont générées par la mise à la masse des entrées RB0, RA3 ou RA4 du port A du μC à l'aide des boutons poussoirs.

LE SCHEMA ELECTRIQUE

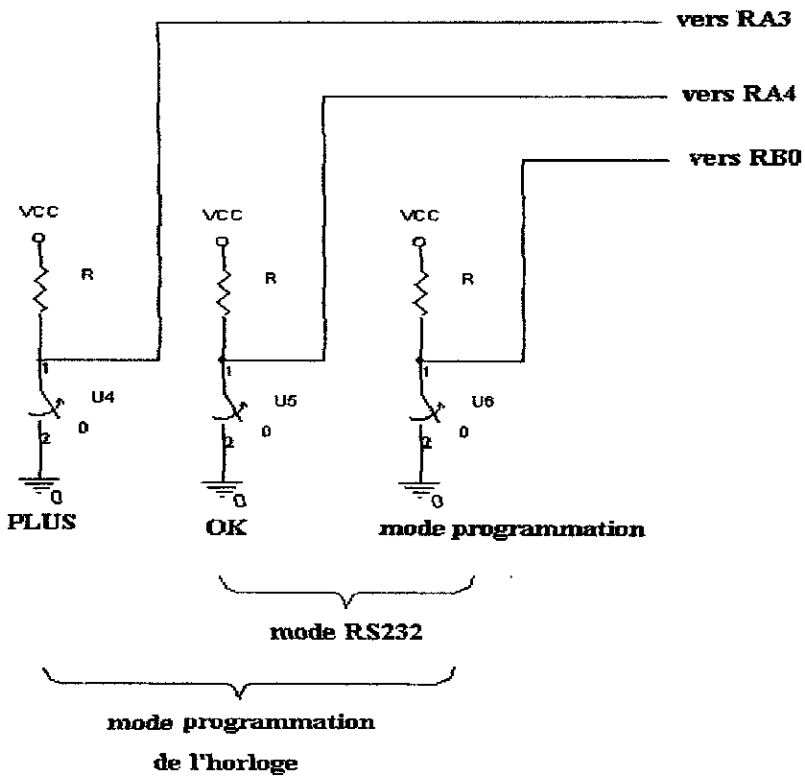
Cette partie est composée de 3 boutons poussoirs (RB0 , OK "RA4", plus "RA3"):

- Le bouton RB0 permet l'entrée en mode interruption.
- Le bouton PLUS permet l'incrémentation de la valeur à programmer.
- Le bouton OK permet la validation de la valeur programmée.

Il existe 3 configurations possibles pour sélectionner les deux types de programmation et la communication RS232,qui sont les suivantes:

- Le mode programmation des instants de marche et d'arrêt: par l'appui sur le bouton RB0.
- Le mode programmation de l'horloge: par l'appui sur les boutons RB0 et PLUS.
- Le mode communication RS232: par l'appui sur les boutons RB0 et OK

Figure 10 :La programmation manuelle



III-3) L'AFFICHAGE

Son rôle est la visualisation du temps (minutes, heures) en mode normal ainsi que toutes les données qui seront stockées dans la mémoire en mode programmation (minutes, heures, jours).

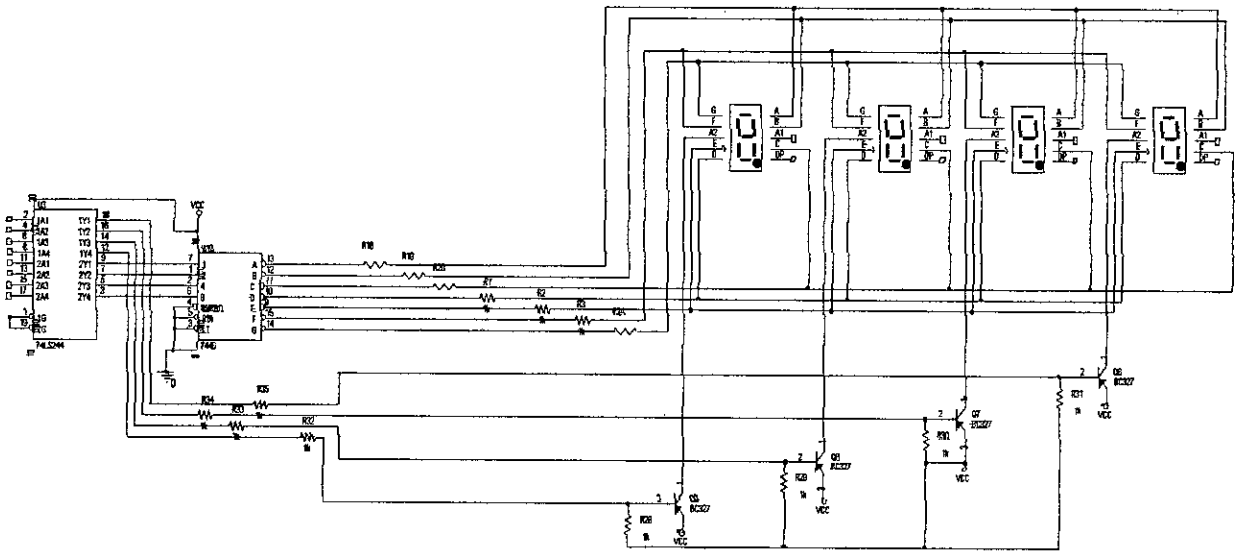
LE SCHEMA ELECTRIQUE

Cette partie est composée des éléments suivants:

- 4 afficheurs 7-segments à anodes communes qui peuvent visualiser toutes les données nécessaires pour la programmation.

- 4 transistors PNP travaillant en commutation sont commandés par le port B du µC pour sélectionner les 4 afficheurs.
- un décodeur DM7447 qui fait la conversion BCD- 7segments, permet l'allumage des afficheurs.
- un buffer 3 états le 74sn244 qui permet la stabilisation des données à l'entrée du décodeur ainsi que la sélection de cette partie d'affichage.
- 7 résistances connectées aux afficheurs pour réduire le courant.

Figure 11 :Le brochage des afficheurs



III-4) L'INTERFACE DE PUISSANCE

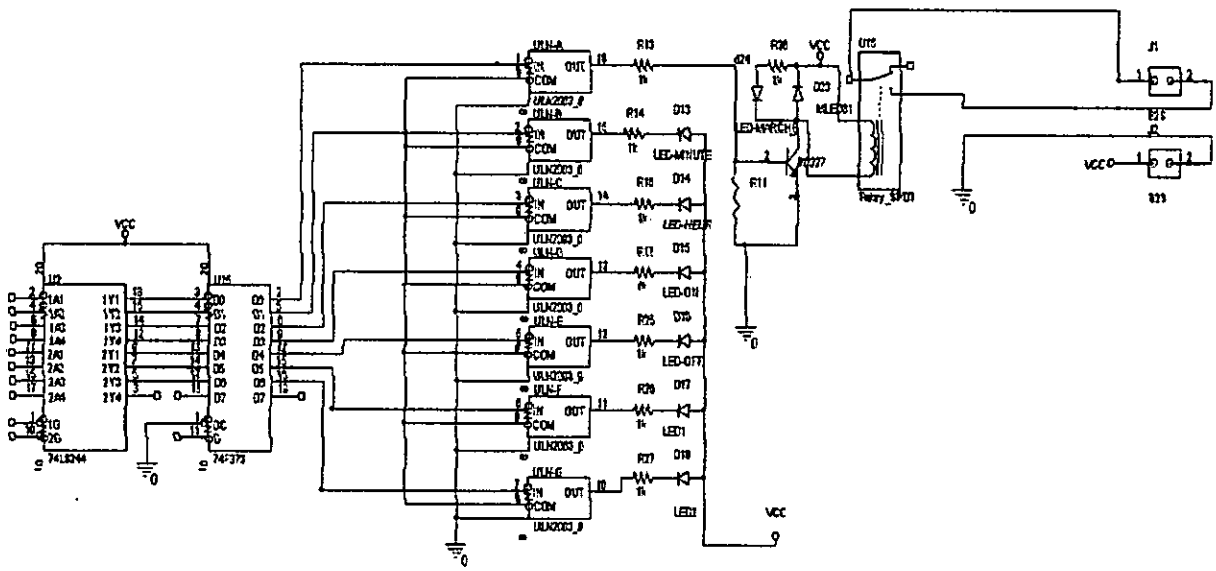
C'est cette interface qui agit sur les charges extérieures (lampes, TV, chauffage...) permettant l'allumage et l'extinction. Il est aussi utilisé pour allumer les LEDs d'indications (LED-minutes, LED-heure, LED-jours,.....) aux moments de la programmation. L'existence de cette interface est dûe à l'incapacité du microcontrôleur de délivrer un courant suffisant pour allumer toutes les LEDs et les charges extérieures.

LE SCHEMA ELECTRIQUE

Cette interface est constituée des composants suivants:

- Un *BUFFER* 3 états "le 74sn244" qui permet la stabilisation des données ainsi que la sélection de cette interface.
- Un *LATCH* à 3états "le 74AC373" qui permet la mémorisation des données de l'interface pour les séparer des données d'affichage, sa sélectionne est faite par le microcontrôleur.
- Un réseau de transistor Darlington " l'ULN2003A ", ce réseau est capable de délivrer un courant max de 500 mA par transistor, cela nous permet l'allumage des LEDs ainsi que l'activation du relais.
- Un relais qui permet l'allumage et l'extinction de charge de 220V par une tension de 5V.

Figure 12 :L'interface de puissance



III-5) L'INTERFACE DE COMMUNICATION RS232

La programmation par PC de notre dispositif peut se faire à l'aide de cette interface qui assure une adaptation entre les signaux générés par la carte qui sont de type TTL (0V, 5V) et les signaux du port de communication du PC qui ont des niveaux de tension de –12V et +12V.

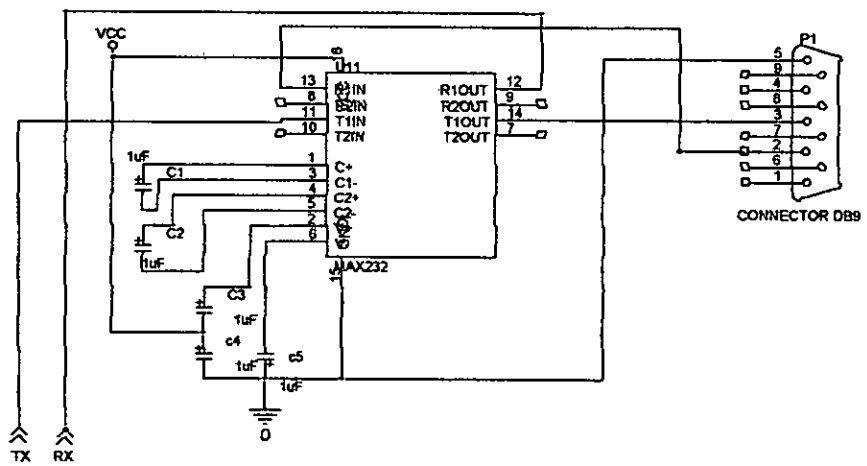
On a le choix entre plusieurs types de communications possible, la communication parallèle, la communication série synchrone et la communication série asynchrone qui est utilisée dans notre application à cause de sa simplicité de connexion et d'utilisation.

LE SCHEMA ELECTRIQUE

L'élément principal de cette interface est le MAX232 qui possède deux drivers en émission et deux en réception et un convertisseur à pompe de charge qui, à partir du +5V, crée du +12V et du -12V. quatre condensateurs de 1µF destinés aux convertisseurs à pompe de charge et un 5ème destiné à filtrer le +5V (Vcc) perturbé par les pointes de courant du convertisseur (commutation).

Pour la mise en œuvre de cette communication série asynchrone il suffit de trois fils (TX: Transmit Data, RX: Receive Data, GND) pour avoir une communication bidirectionnelle entre la carte et le PC.

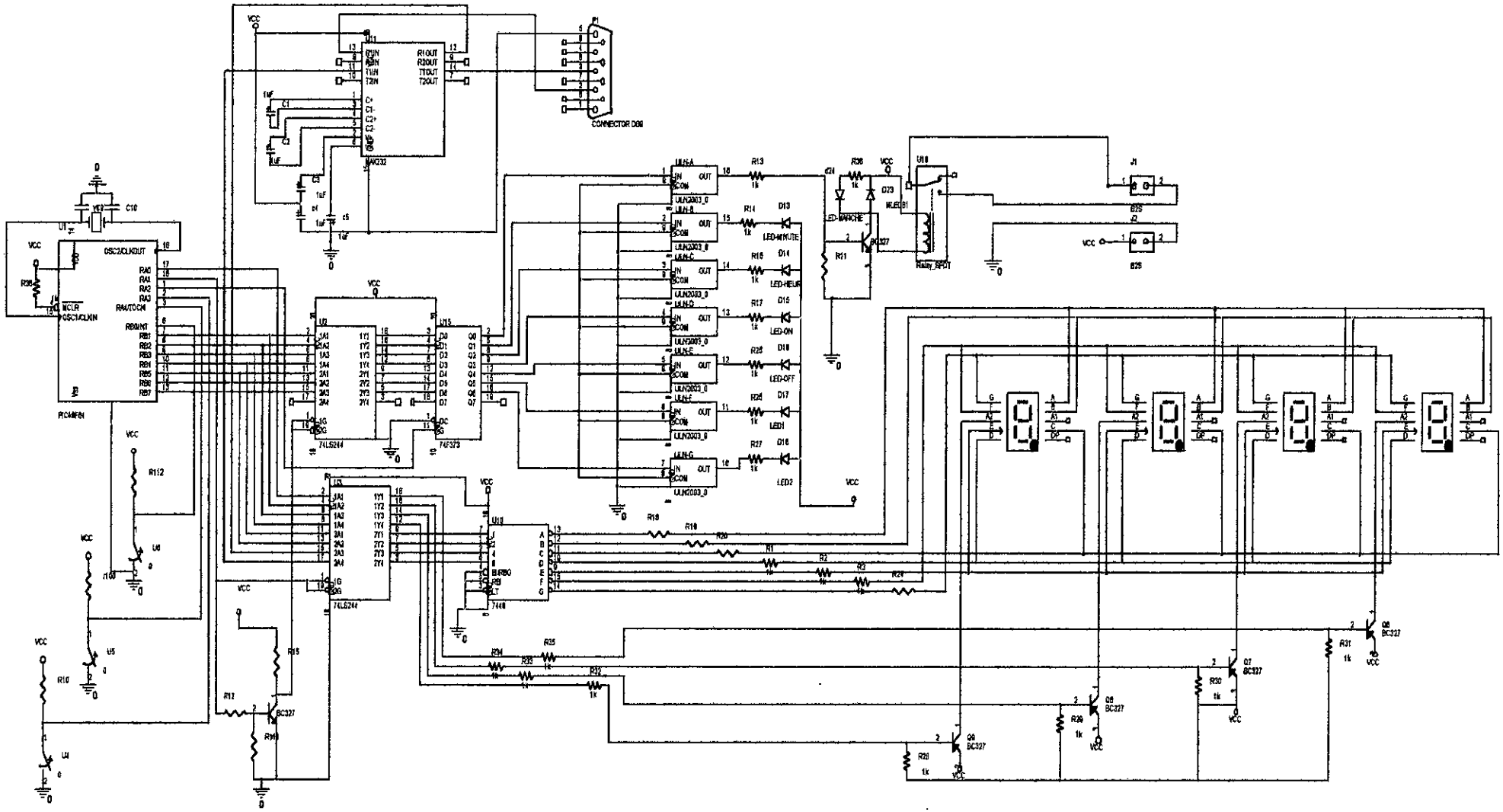
Figure 13 :Le brochage de pilote RS232



III-6) LE CIRCUIT ELECTRIQUE GLOBAL

L'association des 5 parties précédentes donne le circuit électrique global de notre dispositif. Ces parties sont toutes connectées au microcontrôleur *PIC16F84* qui forme le noyau de ce système. La connexion des ces parties figure dans le schéma électrique suivant:

Figure 14 :Le schéma électrique global



III-7) LE CIRCUIT IMPRIME DE LA CARTE

Avant la mise en circuit imprimé, notre schéma électrique était réalisé avec un logiciel d'électronique " *OrCad Capture* " qui permet la simulation des schémas électriques, ce qui n'est pas le cas de notre schéma car il contient un microcontrôleur. Puis on a crée à partir de ce schéma une "NetList" qui est un fichier contenant toutes les informations concernant les liaisons électriques entre composants et leurs caractéristiques physiques (Printed Circuit Board ou PCB footprint) et pour avoir le circuit imprimé on a fair appel au logiciel *OrCad Layout Plus* qui à partir des fichiers N*etList* crée des circuits imprimés, cela est fait après avoir placé les composants de manière à minimiser les intersections des lignes, puis on lance le routage automatique qui nous donne le schéma final de notre circuit imprimé.

Le circuit imprimé de notre dispositif est réalisé sur une carte de double face, cela est dû à la complexité du schéma, la face TOP(composants) où on place les composants et la face bottom (cuivre). La soudure des composants se fait sur les deux face lesquelles sont reliées par des vias.

Les schémas des deux faces sont représentées sur les figures suivantes:

Figure 15 :La face cuivre

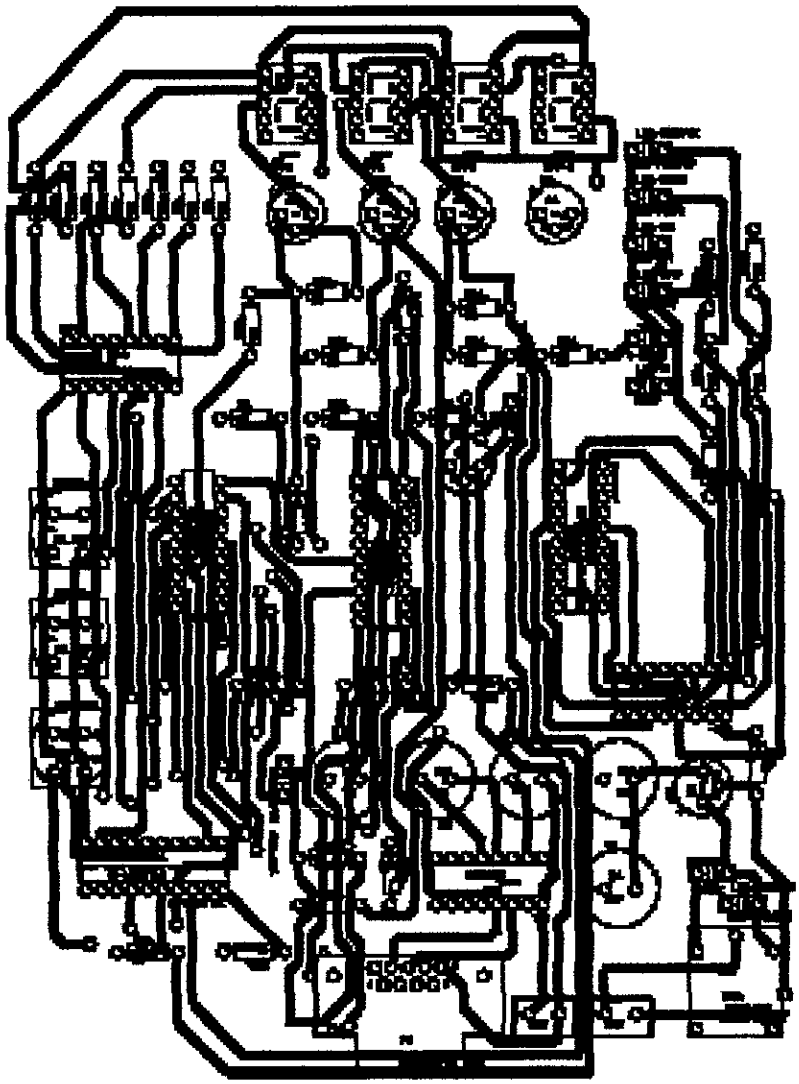
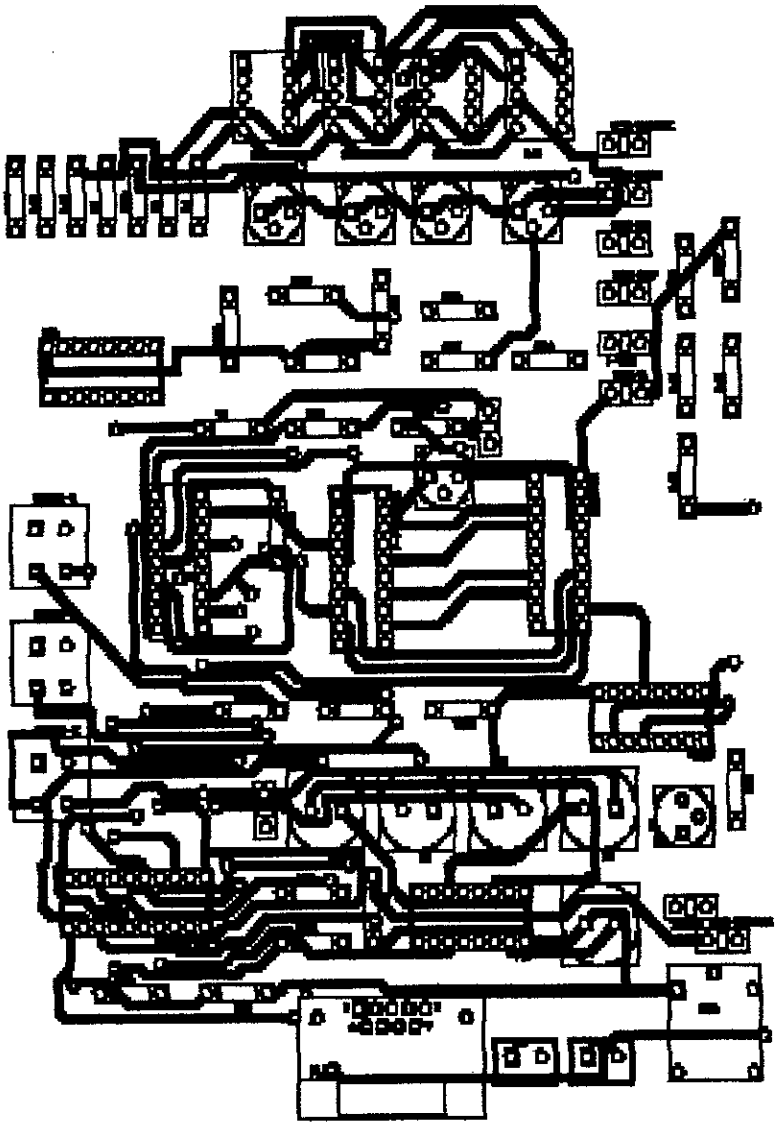


Figure 16 :La face composants



CHAPITRE IV

IV) LA PROGRAMMATION DU MICROCONTROLEUR

Dans cette étape du notre travail on va aborder la partie la plus importante dans la réalisation de notre dispositif. C'est la partie programmation du microcontrôleur PIC16F84 qui va gérer notre dispositif.

La société *Microchip* a mis à la disposition des utilisateurs le logiciel *MPLAB IDE.* Ce logiciel constitue un environnement complet de développement d'applications avec les PICs. Il comprend un éditeur, un assembleur et un simulateur.

IV-1) ORGANISATION DU PROGRAMME

Pour que notre programme soit lisible et bien structuré, on va le diviser en plusieurs parties, chaque partie traitant une fonction bien définie.

IV-1-1) La configuration du microcontrôleur

a) La directive CONFIG

Chaque mode de fonctionnement du Pic nécessite une configuration particulière, celle-ci est appliquée à certains registres du PIC et au fonctionnement du µC lui même.
La directive _CONFIG permet la validation de certaines options, ces dernières sont encodées dans le processeur au moment de la programmation du PIC.

```
__CONFIG  _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
        ; _CP_OFF Code protection désactivée
        ; _PWRTE_ON Timer reset sur power on en service
        ; _WDT_OFF Watch-dog hors service
        ; _HS_OSC Oscillateur quartz grande vitesse.
```

b) Le registre OPTION

Ce registre est un registre de bits, c'est à dire que chaque bit a un rôle particulier. Il se trouve à l'adresse 0x81, donc dans la banque1. Sa valeur peut être modifiée au cours de l'exécution du programme, contrairement aux paramètres de la directive _CONFIG. Ce registre est déclaré avec le nom OPTION_REG. Le tableau suivant représente le contenu de ce registre et sa configuration pour notre programme :

Tableau 1: configuration du registre OPTION

Bit du registre	DEFINTION	Les valeurs de notre programme
B7 : $\overline{RBPU}$	Résistances de rappel du port B validées (=0)	Ne sont pas nécessaires Donc B7= 1
B6 : INTEDG	Sélection de la transition active pour l'interruption RB0	Le sens de l'interruption est de 1vers 0 Donc B6 = 0
B5 : T0CS	Sélection de la source d'horloge pour le timer (RA4 ou cycle d'instruction interne)	Incrémentation en fonction de l'horloge interne Donc B5 =0
B4 : T0SE	Sélection de la transition active pour l'horloge externe	N'est pas utilisé Donc B4 =0
B3 : PSA	Assignation du prédiviseur au timer ou watchdog chien de garde)	Le prédiviseur au timer Donc B3 =0
B2 : PS2	Valeur du prédiviseur	La valeur de prédiviseur est 256 Donc B2,B1,B0 = 111
B1 : PS1		
B0 : PS0		

D'après le tableau précédent la valeur du registre OPTION qu'on va utiliser pour notre programme et qui sera chargée de la valeur OPTIONVAL est:

OPTIONVAL = B'1 0 0 0 0 1 1 1' = H' 87'

C-Le registre INTCON

Ce registre se situe à l'adresse 0x0B, dans les 2 banques. Il est donc toujours accessible. Il contient toutes les informations qui concernent les interruptions. Son contenu et sa configuration sont représentés sur le tableau suivant:

Tableau 2: configuration du registre INTCON

Bit du registre	DEFINTION	Les valeurs de notre programme
B7 : GIE	Validation ou inhibition générale des interruptions.	Les interruptions sont validées Donc B7 = 1
B6 : EEIE	Validation de l'interruption de fin d'écriture en EEPROM.	*invalide* Donc B6 = 0
B5 : T0IE	Validation de l'interruption générée par le débordement du *timer*.	Interruption générée par le débordement du timer validé *Donc B5 = 1*
B4 : INTE	*Validation de l'interruption* générée par modification du niveau sur RB0.	*l'interruption générée par* modification du niveau sur RB0 est validé donc B4 = 1
B3 : RBIE	Validation de l'interruption générée par un changement de niveau sur une des entrées RB4 à RB7.	Invalide *B3 = 0*
B2 : T0IF	Indicateur (flag) de débordement du timer.	Initialement B2 = 0
B1 : INTF	Indicateur de transition active sur *la pin RB0.*	Initialement B1 = 0
B0 : RBIF	Indicateur de changement sur l'une des pin RB4 à RB7.	Invalide donc B0 =0

Donc compte tenu de notre programme la valeur initiale de registre INTCON est:

INTERMASK = B'1 0 1 1 0 0 0 0' = H'b0'

D- Les registres TRISA & TRISB

Ces deux registres se trouvent en banque1, à l'adresse H'85' pour le registre TRISA et à l'adresse H'86' pour le registre TRISB. Ces deux registres sont liés aux fonctionnement des PORTA et PORTB. Chaque bit positionné à 1 configure la pin correspondante en entrée. Chaque bit à 0 configure la pin en sortie. Au reset du PIC, toutes les pins sont mises en entrée,

afin de ne pas envoyer des signaux non désirés sur les pins. Notez également que, comme il n'y a que 5 pins utilisées sur le PORTA, seuls 5 bits (b0/b4) seront utilisés sur TRISA.

Dans notre cas les 7 pins (RB1→RB7) du PORTB seront configurées en sortie, permettant la transmission des données vers les afficheurs et l'activation des LEDs et la charge, ainsi que les 3 pins (RA0→RA2) du PORTA qui permettent la sélection des parties affichage et puissance. Les autres pins sont toutes configurées en entrée, la pin RB0 pour valider les interruptions externes et les pins RA3 et RA4 comme des entrées *PLUS* et *OK* de la partie programmation manuelle. Donc les registres TRISA et TRISB seront chargés des valeurs suivantes:

TRISA = B' 1 1 0 0 0' = H'18'

TRISB = B' 0 0 0 0 0 0 0 1' = H'01'

IV-1-2) Les assignations

Les assignations se comportent comme une simple substitution. Au moment de l'assemblage, chaque fois que l'assembleur va trouver une assignation, il la remplacera automatiquement par sa valeur. Cela facilitera la maintenance du programme. Car si on change la valeur d'une assignation, le changement sera effectué pour tout le programme. Donc on ne risque pas d'oublier des valeurs en chemin. Elles sont utilisées dans notre programme et sont représentées par la syntaxe EQU.

```
OPTIONVAL       EQU   H'0087'      ; Valeur de  registre option
INTERMASK       EQU   H'00b0'      ; Masque d'interruption, registre intcon
```

IV-1-3) Les définitions

Elles fonctionnent comme des assignations, mais pour remplacer des textes plus complexes. Elles sont construites de la manière suivante: la directive #DEFINE, suivie par le nom que l'on désire utiliser, puis la chaîne à substituer.

Exemple du notre programme:

```
#DEFINE ledmin     PORTB,2      ; LED de  témoin des minutes
#DEFINE ledheur    PORTB,3      ; LED de témoin des heurs
#DEFINE ledon      PORTB,4      ; LED indicant l'allumage
#DEFINE ledoff     PORTB,5      ; LED indicant l'éteint
#DEFINE charge     PORTB,1      ; la charge extérieur
#DEFINE plus       PORTA,3      ; bouton poussoir plus
#DEFINE OK         PORTA,4      ; bouton poussoir OK
```

IV-1-4) les macros

Pour simplifier l'écriture du programme, certains morceaux de code qui sont souvent utilisés sont remplacés par des macros. La macro se compose d'un nom écrit en première colonne, suivi par la directive « macro ». Commence alors à la ligne suivante la portion de code qui constitue la macro. La fin de la macro est définie par la directive « endm) (end of macro). Dans notre programme on a utilisé quatre macros, deux pour le changement de banque et deux pour la lecture et l'écriture dans la mémoire *EEPROM.*

- Macros de changement de banque:

```
BANK0     macro
          bcf     STATUS , RP0        ; passer banque0
endm

BANK1     macro
          bsf     STATUS , RP0        ; passer banque1
endm
```

- Macro de lecture de la mémoire *EEPROM*:

```
READEE    macro  adeeprom     ; macro avec parametres
          movf    jourvar,w   ; charger adresse EEPROM
          movwf   EEADR       ; adresse à lire dans registre EEADR
          BANK1               ; passer en banque1
          bsf     EECON1 , RD ; lancer la lecture EEPROM
          BANK0               ; repasser en banque0
          movf    EEDATA , w  ; charger la valeur lue dans W
endm                          ; fin de la macro
```

- Macro d'écriture dans la mémoire *EEPROM*:

```
WRITEE    macro  addwrite     ; la donnée se trouve dans W
          movwf  EEDATA       ; placer data dans registre
          movf   jourvar,w    ; charger adresse d'écriture
          movwf  EEADR        ; placer dans registre
          BANK1               ; passer en banque1
          bcf    EECON1 , EEIF ; effacer flag de fin d'écriture
          bsf    EECON1 , WREN ; autoriser accès écriture
```

```
        movlw  0x55                ; charger 0x55
        movwf  EECON2              ; envoyer commande
        movlw  0xAA                ; charger 0xAA
        movwf  EECON2              ; envoyer commande
        bsf    EECON1 , WR         ; lancer cycle d'écriture
        bcf    EECON1 , WREN       ; verrouiller prochaine écriture
        BANK0                      ; repasser en banque0
endm
```

IV-1-5) Déclaration des variables

Dans cette zone on va définir toutes les variables, qui sont des emplacements mémoires auxquels on a donné un nom, utilisées dans notre programme. Cette zone utilisable débute à l'adresse 0x0C dans la mémoire RAM, elle commence par la directive *CBLOCK*. Chaque déclaration repond à la syntaxe suivante : nom de la variable suivi du signe « : » suivi de la taille utilisée.

```
CBLOCK 0x0C                ; début de la zone variables
        w_temp :1          ; Sauvegarde du registre W
        status_temp : 1    ; Sauvegarde du registre status
        indmarche :1       ; indice de marche ou d'arrêt
        indmarche_temp :1  ; Sauvegarde d'indice de marche ou d'arrêt
        secondes :1        ; registre des secondes
        minutes :1         ; registre des minutes
        heures :1          ; registre de heures
        jours :1           ; registre de jours
        jourvar :1         ; une copie de registre jours
        jourvar_temp :1    ; sauvegarde de registre jourvar
        jourp :1           ; une autre copie de registre jours
        jourp_temp :1      ; sauvegarde de registre jourp
        minheur :1         ; valeur codée de minute et heure
        minheur_temp :1    ; sauvegarde minheur
        decmin :1          ; valeur décodée de minute
        decmin_temp :1     ; sauvegarde de decmin
        decheur :1         ; valeur décodée de l'heure
```

```
        decheur_temp        ; sauvegarde de decheur
        etatmh :1           ; registre contient les indice de codage
        etatmh_temp         ; sauvegarde de registre etatmh
        cmpt :1             ; compteur
        afvar :1            ; valeur à afficher
        afficheur1 :1       ; la valeur à afficher sur l'afficheur1
        afficheur2 :1       ; la valeur à afficher sur l'afficheur2
        afficheur3 :1       ; la valeur à afficher sur l'afficheur3
        afficheur4 :1       ; la valeur à afficher sur l'afficheur4
        valeur :1           ; constante égale à dix
        cont :1             ; compteur
        cmpt1 :1            ; compteur1
        cmpt2 :1            ; compteur2
        etaPB :1            ; la donnée qui se trouve sur le portB
endc                        ; Fin de la zone
```

IV-1-6) Initialisation de programme

En cas de mise sous tension ou d'un redémarrage du PIC, le processeur commence l'exécution du programme à l'adresse 0x00 qui contient un saut inconditionnel vers la routine d'initialisation. Cette routine contient toutes les valeurs de départ de nos variables déclarées auparavant ainsi que toutes les configurations des registres et des ports.

Avant l'initialisation des variables notre routine fait un effacement général de tout l'espace mémoire de la RAM, ceci permet d'éliminer toutes valeurs aléatoires des registres après le redémarrage.

Sous-programme d'initialisation

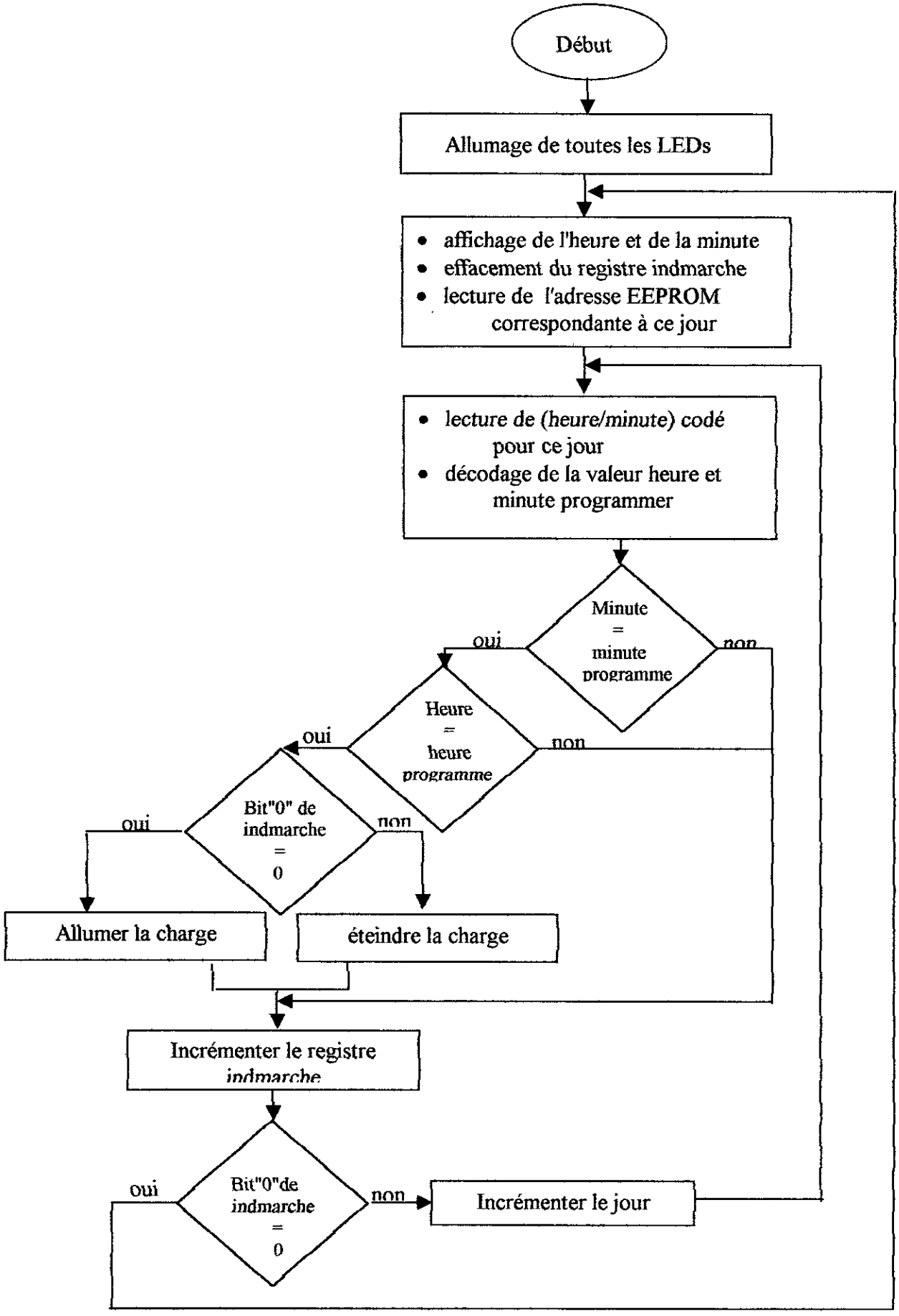
```
init
        clrf    PORTA           ; Sorties portA à 0
        clrf    PORTB           ; sorties portB à 0
        bsf     STATUS,RP0      ; sélectionner banque 1
        clrf    EEADR           ; permet de diminuer la consommation
        movlw   OPTIONVAL       ; charger masque
        movwf   OPTION_REG      ; initialiser registre option
```

```
; Effacer la RAM
; -----------
        movlw   0x0c            ; initialisation pointeur
        movwf   FSR             ; pointeur d'adressage indirect
        init1
        clrf    INDF            ; effacer ram
        incf    FSR,f           ; pointer sur suivant
        btfss   FSR,6           ; tester si fin zone atteinte (>=40)
        goto    init1           ; non, boucler
        btfss   FSR,4           ; tester si fin zone atteinte (>=50)
        goto    init1           ; non, boucler
        BANK0                   ; Sélectionner banque 0
        movlw   15              ; initialisation de compteur
        movwf   cmpt            ; par la valeur 15
        movlw   1               ;
        movwf   jours           ; initialisation de registre jours par 1
        BANK1                   ; changement de banque
        movlw   0x01            ;
        movwf   TRISB           ; la configuration du portB
        movlw    24             ;
        movwf   TRISA           ; la configuration du portA
        movlw   INTERMASK       ; masque interruption
        movwf    INTCON         ; charger interrupt control
        BANK0                   ; changement de banque
        goto    start           ; saut au programme principal
```

IV-1-7) Le programme principal

Après l'initialisation de toutes les variables et la configuration des ports A et B, le processeur exécute le programme principal. Dans notre cas le programme principal travaille comme une horloge, il affiche l'heure et la minute et teste si ceci correspond à l'heure et à la minute d'allumage ou d'extinction programmées pour le même jour. Si cela est vrai le µC allume ou éteint la charge, sinon il continue l'affichage du temps. Au moment d'une interruption le processeur sort du programme principal et exécute le programme de l'interruption correspondante.

Figure 17 :Organigramme du programme principal



Le programme principal

```
start
        clrf    PORTB           ; effacer le contenu du portB
        movlw   0xff            ;
        iorwf   PORTA,f         ; allumer toutes les LEDs d'indication
        bcf     controle        ; validation de l'allumage
etq1
        movf    heurs,w         ; lecture de la valeur de l'heure
        movwf   afvar           ;
        call    BCD             ; appel de la routine de conversion binaire-bcd
        movf    afficheur1,W    ;       affichage de
        movwf   afficheur3      ;       la valeur
        movf    afficheur2,w    ;       de l'heure dans
        movwf   afficheur4      ;       les afficheurs 3 et 4
        movf    minutes,w       ; lecture de la valeur de minute
        movwf   afvar           ;
        call    BCD             ; appel de la routine de conversion binaire-bcd
        call    affichage       ; affichage de l'heure et de minute
        clrf    indmarche       ; effacer l'indice de marche ou d'arrêt
        movf    jours,w         ; lecture de la valeur de jour
        movwf   jourp           ;
        call    tablejours      ;  lecture de l'adresse EEPROM correspondante
                                ;               à ce jour
        clrf    PCLATH          ;
        movwf   jourvar         ;
etq2
        READEE  jourvar         ;  lecture de la valeur EEPROM (heure, minute )
        Movwf   minheur         ;       programmer auparavant
        call    DECODE          ; décodage de la valeur lu (heure , minute)
        movf    decmin,w        ;       tester si la valeur minute de l'horloge
        subwf   minutes,w       ;       est égale à la valeur minute lu
        btfss   STATUS,Z        ;
        goto    changemin       ; si non aller vers le test de la  minute suivante
        movf    decheur,w       ;  si oui tester si la valeur de l'heure de l'horloge
```

```
        subwf   heurs,w         ; si  est égale à la valeur de l'heure programmée
        btfss   STATUS,Z        ;
        goto    changemin       ; si non aller vers le test de la minute suivante
        btfss   indmarche,0     ;si oui tester le bit 0 de registre indmarche
                                ; si égal à zéro
        goto    allume          ;  si oui sauter vers allumage
        clrf    etaPB           ;  si non éteindre la charge
        clrf    PORTB           ;       sélection
        bsf     controle        ;    de la partie puissance
        bcf     controle        ; validation de l'extinction
        goto    changemin       ;  aller vers le test de la minute suivante
allume
        clrf    PORTB           ;  allumage de la charge
        bsf     charge          ; sélection
        bsf     controle        ; de la partie puissance
        bcf     controle        ; validation de l'allumage
        movlw   2               ;       chargement du registre
        movwf   etaPB           ;       etaPB par la valeur 2
changemin
        incf    indmarche,f     ;  incrémentation de l'indmarche
        btfss   indmarche,0     ;tester si  le bit0 d'indmarche égale à zéro
        goto    etq1            ; si oui aller vers etq1
        incf    jourvar,f       ; si non incrémenter la valeur du jour
        goto    etq2            ; saut vers etq2
repere
        ORG (repere+5 )         ; repère de programme
tablejours
        incf    PCLATH,f        ;       incrémentation de
        incf    PCLATH,f        ;  registre  PCLATH par 2 (changement de page )
        addwf   PCL , f         ; ajouter w à PCL
        nop
        retlw   00              ; adresse EEPROM du 1 jour
        retlw   02              ; adresse EEPROM du 2 jour
        retlw   04              ; adresse EEPROM du 3 jour
```

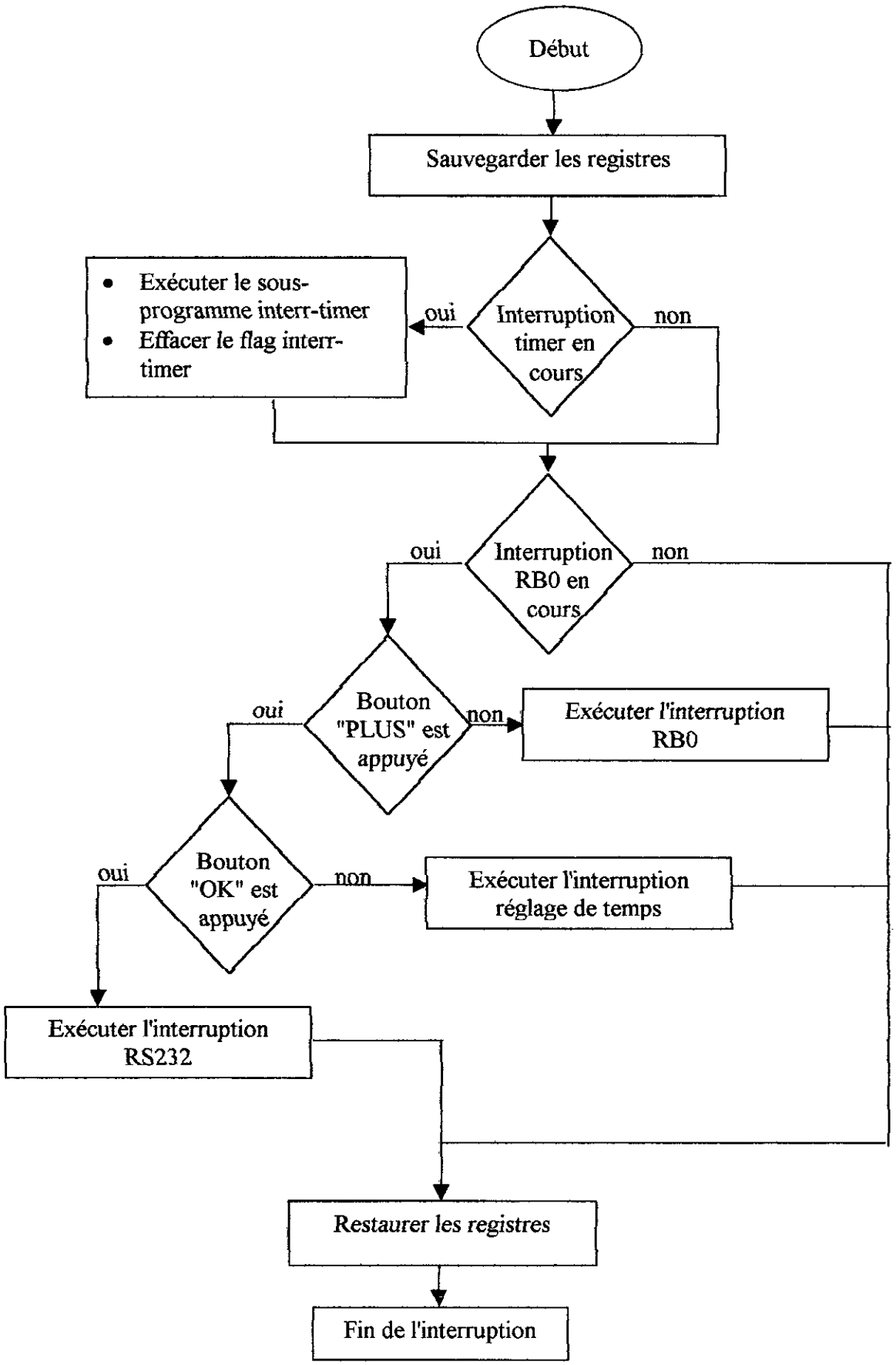
```
        retlw   06      ; adresse EEPROM du 4 jour
        retlw   08      ; adresse EEPROM du 5 jour
        retlw   10      ; adresse EEPROM du 6 jour
        retlw   12      ; adresse EEPROM du 7 jour
        retlw   14      ; adresse EEPROM du 8 jour
        retlw   16      ; adresse EEPROM du 9 jour
        retlw   18      ; adresse EEPROM du 10 jour
        retlw   20      ; adresse EEPROM du 11 jour
        retlw   22      ; adresse EEPROM du 12 jour
        retlw   24      ; adresse EEPROM du 13 jour
        retlw   26      ; adresse EEPROM du 14 jour
        retlw   28      ; adresse EEPROM du 15 jour
        retlw   30      ; adresse EEPROM du 16 jour
        retlw   32      ; adresse EEPROM du 17 jour
        retlw   34      ; adresse EEPROM du 18 jour
        retlw   36      ; adresse EEPROM du 19 jour
        retlw   38      ; adresse EEPROM du 20 jour
        retlw   40      ; adresse EEPROM du 21 jour
        retlw   42      ; adresse EEPROM du 22 jour
        retlw   44      ; adresse EEPROM du 23 jour
        retlw   46      ; adresse EEPROM du 24 jour
        retlw   48      ; adresse EEPROM du 25 jour
        retlw   50      ; adresse EEPROM du 26 jour
        retlw   52      ; adresse EEPROM du 27 jour
        retlw   54      ; adresse EEPROM du 28 jour
        retlw   56      ; adresse EEPROM du 29 jour
        retlw   58      ; adresse EEPROM du 30 jour

END                     ; directive fin de programme
```

IV-1-8) Les routines d'interruption

La routine d'interruption est un sous-programme particulier, déclenché par l'apparition d'un événement spécifique. Notre programme dispose de quatre routines d'interruptions permettant la programmation des moments d'allumage et d'extinction ainsi que le réglage du temps. Avant l'exécution de ces routines il faut sauvegarder certains registres pour permettre au programme principal de fonctionner dans l'état où il était au moment de l'interruption.

A chaque interruption, un sous-programme cherche l'origine de cette interruption et lui associe sa routine. Avant de sortir de l'interruption, on doit restaurer tous les registres sauvegardés auparavant.

Figure 18: Organigramme d'interruptions



Sous-programme d'interruption

;le sauvegarde des registres

```
org 0x004                                   ; adresse d'interruption
        movwf    w_temp                     ; sauvegarder le registre W dans w_temp
        swapf    STATUS,w                   ; swap status avec résultat dans w
        movwf    status_temp                ; sauvegarder le registre status swappé
        movf     indmarche,w                ; sauvegarder le registre indmarche
        movwf    indmarche_temp             ;       dans le registre indmarche_temp
        movf     jourp,w                    ; sauvegarder le registre jourp
        movwf    jourp_temp                 ;       dans le registre jourp_temp
        movf     jourvar,w                  ; sauvegarder le registre jourvar
        movwf    jourvar_temp               ;       dans le registre jourvar_temp
        movf     minheur,w                  ; sauvegarder le registre minheur
        movwf    minheur_temp               ;       dans le registre minheur_temp
        movf     decmin,w                   ; sauvegarder le registre decmin
        movwf    decmin_temp                ;       dans le registre decmin_temp
        movf     decheur,w                  ; sauvegarder le registre decheur
        movwf    decheur_temp               ;       dans le registre decheur_temp
        movf     etatmh,w                   ; sauvegarder le registre etatmh
        movwf    etatmh_temp                ;       dans le registre etatmh_temp
```

; switch vers différentes interruptions

```
        btfsc    INTCON,T0IE                ; tester si interrupt timer autorisée
        btfss    INTCON,T0IF                ; oui, tester si interrupt timer en cours
        goto     intsw1                     ; non test suivant
        call     inttimer                   ; oui, traiter interrupt timer
        bcf      INTCON,T0IF                ; effacer flag interrupt timer
intsw1
        btfsc    INTCON,INTE                ; tester si interrupt RB0 autorisée
        btfss    INTCON,INTF                ; oui, tester si interrupt RB0 en cours
        goto     restorereg                 ; aller à la restauration des registres
```

```
        btfsc   plus            ; tester si le bouton plus est appuyé
        goto    intprog         ; non, aller vers la routine de réglage de temps
        btfsc   OK              ; oui, tester le si le boutons OK est appuyé
        goto    intrt           ; non, aller à la routine d'interruption réglage de
                                ;temps
        call    intrs232        ; oui, appel de la routine RS232
        goto    restorereg      ; aller à la sauvegarde des registres
intprog
        call    intprogramme    ; appel de la routine de programmation
        goto    restorereg      ; aller à la sauvegarde des registres
intrt
        call    intrtime        ; appel de la routine de réglage de temps
        goto    restorereg      ; aller à la sauvegarde des registres

                        ;restauration des registres
restorereg
        bcf     INTCON,INTF     ; effacer flag interupt RB0
        movf    indmarche_temp,w ;     restauration
        movwf   indmarche       ;        des registres
        movf    jourp_temp,w    ;   sauvegardés auparavant
        movwf   jourp           ;
        movf    jourvar_temp,w  ;
        movwf   jourvar         ;
        movf    minheur_temp,w  ;
        movwf   minheur         ;
        movf    decmin_temp,w   ;
        movwf   decmin          ;
        movf    decheur_temp,w  ;
        movwf   decheur         ;
        movf    etatmh_temp,w   ;
        movwf   etatmh          ;
        swapf   status_temp,w   ; swap ancien status, résultat dans w
        movwf   STATUS          ; restaurer status
        swapf   w_temp,f        ; Inversion L et H de l'ancien W
```

```
                                    ; sans modifier Z
        swapf       w_temp,w        ; Réinversion de L et H dans W
                                    ; W restauré sans modifier status
retfie                              ; sortir de l'interruption
```

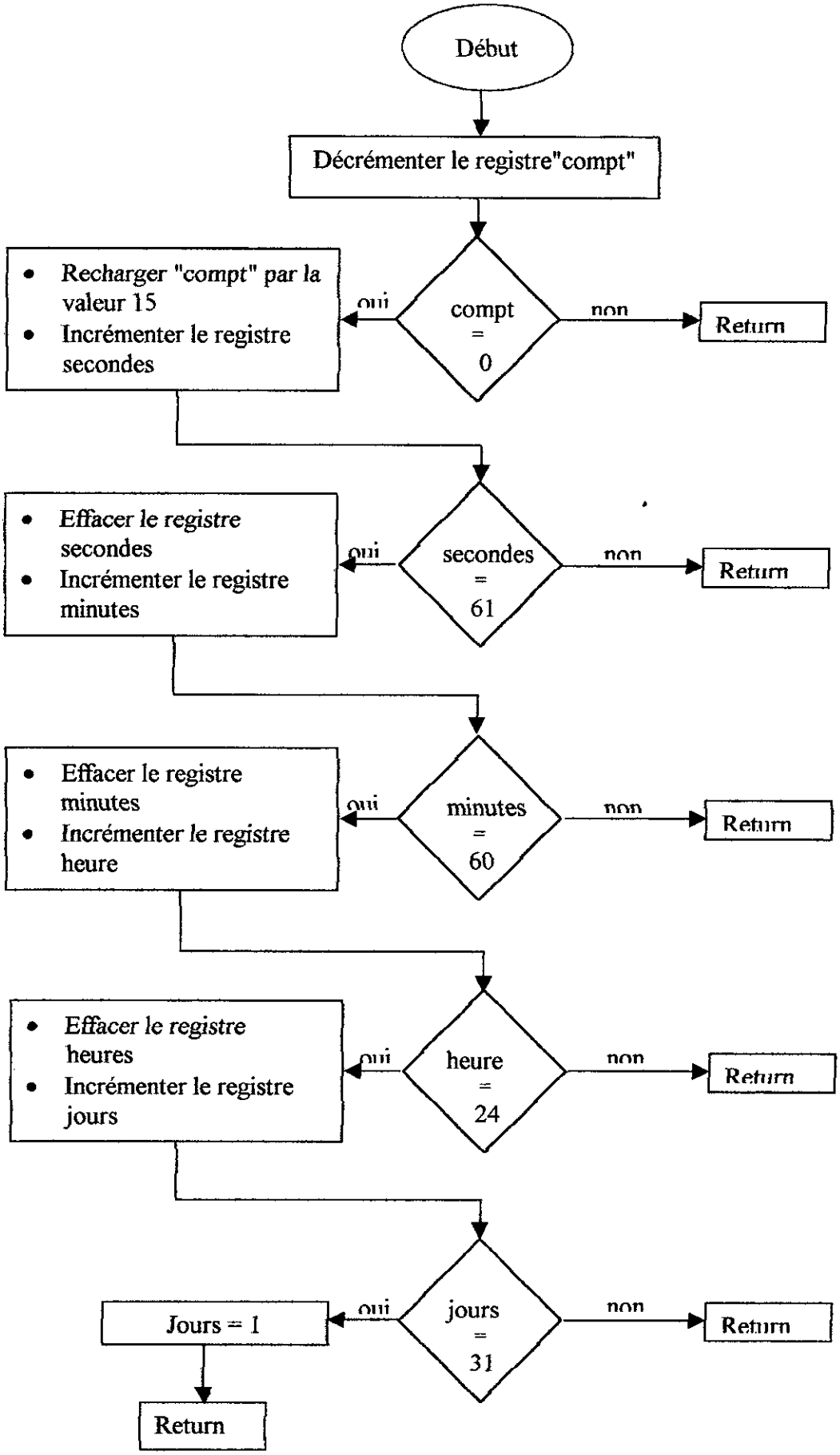
a) Routine d'interruption timer

Cette routine calcule le temps nécessaire pour incrémenter les registres représentant la base de temps (secondes, minutes, heures, jours). Le débordement du registre *timer* génère une interruption qui engendre l'exécution de cette routine, la configuration du registre option nous permet de choisir entre plusieurs durées de débordement. On a choisi une durée de 65536 µs correspondant à une valeur de prédiviseur du *timer* égale à 256.

Pour avoir une incrémentation du registre des secondes chaque 1s, il faut multiplier cette durée par 15,258 (attendre 15,258 débordement). Puisque on ne peut pas charger un registre avec une telle valeur, on fait une multiplication par 15, on aura donc :

65536 µs x 15 = 0,98304 s. donc l'erreur est de 1 – 0,98304 = 0,01696 s chaque 1s et de 1,017 s chaque une minute. Pour améliorer la précision pendant une minute on multiplie par 61 la durée de la seconde, on aura donc une erreur de 60 – (65536 µs x 15 x 61) = 0,03456 s par minute et 2s par heure.

Figure 19: organigramme d'interruption Timer



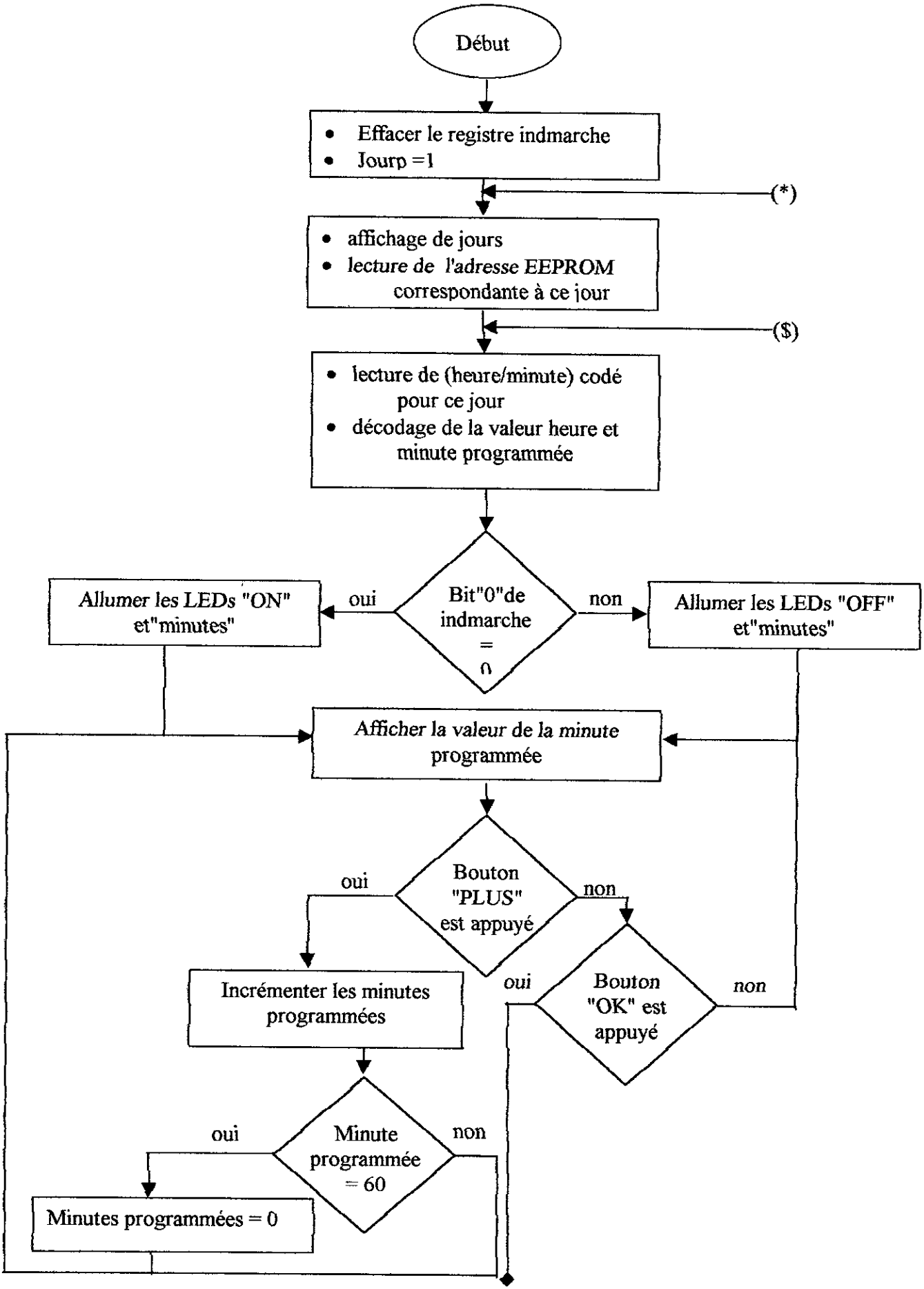
Sous-programme d'interruption timer

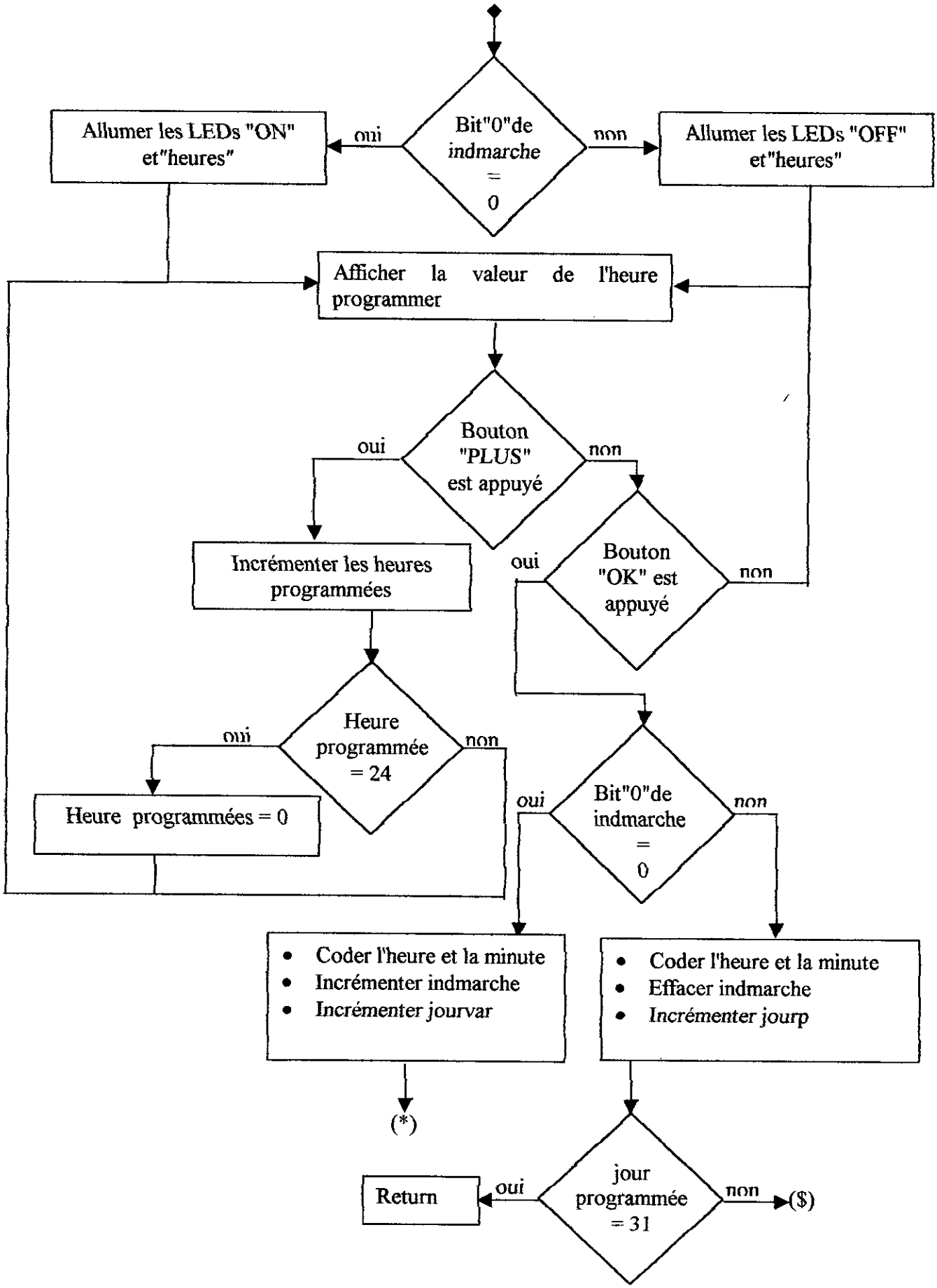
```
inttimer
        decfsz      cmpt,f          ; décrémentation de "cmpt",tester si = 0
        return                      ; si non, fin d'interruption timer
        movlw       0x0f            ; si oui, charger "cmpt" par la valeur 15
        movwf       cmpt            ;
        incf        secondes,f      ; incrémentation de registre "secondes"
        movlw       0x3d            ;
        subwf       secondes,w      ;
        btfss       STATUS,Z        ; tester si "secondes" = 61
        return                      ; si non, fin d'interruption timer
        clrf        secondes        ; si oui, effacer le registre "secondes"
        incf        minutes,f       ; incrémentation de registre "minutes"
        movlw       0x3c            ;
        subwf       minutes,w       ;
        btfss       STATUS,Z        ; tester si "minutes" = 60
        return                      ; si non, fin d'interruption timer

        clrf        minutes         ; si oui, effacer le registre "minutes"
        incf        heurs,f         ; incrémentation de registre "heurs"
        movlw       0x18            ;
        subwf       heurs,w         ;
        btfss       STATUS,Z        ; tester si "heurs" = 24
        return                      ; si non, fin d'interruption timer
        clrf        heurs           ; si oui, effacer le registre "heurs"
        incf        jours,f         ; incrémentation de registre "jours
        movlw       0x1f            ;
        subwf       jours,w         ;
        btfss       STATUS,Z        ; tester si "jours" = 30
        return                      ; si non, fin d'interruption timer
        movlw       0x01            ; si oui, charger le registre jours par la valeur 1
        movwf       jours           ;
        return                      ; fin d'interruption timer
```

b) routine de programmation RB0

Cette routine a pour rôle la programmation manuelle, avec les deux boutons (plus, OK), des moments d'allumage et d'extinction de 30 jours. Cette routine est déclenchée après l'appui sur le bouton *RB0*. le processeur affiche 01 sur les afficheurs 3et 4 indiquant le premier jour et affiche l'ancienne valeur de minute dans les afficheurs 1et 2, ceci en allumant les LEDs minute et ON, indiquant la minute d'allumage du premier jour. Il suffit d'appuyer sur le bouton "plus", pour incrémenter la valeur de la minute, et sur le bouton "OK" pour enregistrer la modification et de passer au réglage de l'heure d'allumage pour le premier jour. En appuyant de nouveau sur le bouton"OK" on passe au réglage de la minute et de l'heure de l'extinction du premier jour. L'appui sur le bouton "OK" après chaque programmation de l'heure d'extinction nous mène à la programmation du jour suivant et ainsi de suite jusqu'au dernier jour.

Figure 20: Organigramme de l'interruption RB0




Bit"0"de indmarche = 0
oui
Allumer les LEDs "ON" et"heures"
non
Allumer les LEDs "OFF" et"heures"
Afficher la valeur de l'heure programmer
Bouton "PLUS" est appuyé
oui
non
Incrémenter les heures programmées
Bouton "OK" est appuyé
oui
non
Heure programmée = 24
oui
non
Heure programmées = 0
Bit"0"de indmarche = 0
oui
non
• Coder l'heure et la minute
• Incrémenter indmarche
• Incrémenter jourvar
• Coder l'heure et la minute
• Effacer indmarche
• Incrémenter jourp
(*)
jour programmée = 31
oui
non
Return
($)

sous-programme d'interruption RB0

```
intprogramme
        clrf    indmarche       ; effacer le registre indmarche
        movlw   1               ; charger le registre jourp
        movwf   jourp           ;       par la valeur 1
etq10
        movwf   afvar           ;
        call    BCD             ; convertir la valeur de jour au bcd
        movf    afficheur1,w    ; afficher la valeur de jour
        movwf   afficheur3      ;  dans les afficheurs 3et 4
        movf    afficheur2,w    ;
        movwf   afficheur4      ;
        movf    jourp,w         ;
        call    tablejours      ; lecture de l'adresse EEPROM correspondante
                                ;                       à ce jour
        clrf    PCLATH          ;
        movwf   jourvar         ;
etq100

        READEE  jourvar         ; lecture de la valeur EEPROM (heure, minute )
        movwf   minheur         ;       programmée auparavant
        call    DECODE          ; décodage de la valeur lu (heure, minute)
        btfsc   indmarche,0     ; tester si le bit "0" de "indmarche" = 0
        goto    pmhoff          ;si non, saut à l'étiquette pmhoff
        movf    etaPB,w         ;si oui, allumer la LED "ON" et la LED "minute"
        movwf   PORTB           ;       sans affecter la charge
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bsf     ledmin          ;
        bsf     ledon           ;
        bcf     controle        ; sélectionner la partie affichage
        goto    affiche         ; afficher le jour et la valeur de la minute
```

```
pmhoff
        movf    etaPB,w         ;
        movwf   PORTB           ;
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bsf     ledmin          ; allumer la LED "OFF" et la LED "minute"
        bsf     ledoff          ;      sans affecter la charge
        bcf     controle        ; sélectionner la partie affichage
affiche
        movf    decmin,w        ;
        movw    afvar           ;  convertir la valeur de
        call    BCD             ;        la minute décodée en bcd
reaffiche
        call    affichage       ; appel de la routine d'affichage
        btfsc   plus            ; tester si le bouton "plus" est appuyé
        goto    tok             ;si non, sauter vers l'étiquette tok
        incf    decmin,f        ;si oui, incrémenter la valeur de la minute
        movlw   60              ;
        subwf   decmin,w        ;
        btfss   STATUS,Z        ; tester si minutes = 60
        goto    affiche         ; si non, sauter vers l'étiquette affiche
        clrf    decmin          ;si oui, effacer le registre decmin
        goto    affiche         ; sauter vers l'étiquette affiche
tok
        btfsc   OK              ; tester si le bouton "OK" est appuyé
        goto    reaffiche       ; si non, sauter vers l'étiquette reaffiche
        btfsc   indmarche,0     ; si oui, tester si le bit "0" de indmarche = 0
        goto    pmhoff1         ; si non, sauter vers l'étiquette pmhoff1
        movf    etaPB,w         ; si oui, allumer la LED "ON" et la LED "heure"
        movwf   PORTB           ;        sans affecter la charge
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bsf     ledheur         ;
        bsf     ledon           ;
```

```
        bcf     controle        ; sélectionner la partie affichage
        goto    affiche2        ; sauter vers l'étiquette affiche2
pmhoff1
        movf    etaPB,w         ;
        movwf   PORTB           ;
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bsf     ledheur         ; si oui, allumer la LED "OFF" et la LED "heure"
        bsf     ledoff          ;       sans affecter la charge
        bcf     controle        ; sélectionner la partie affichage
affiche2
        movf    decheur,w       ;
        movwf   afvar           ; convertire la valeur
        call    BCD             ;  de l'heure au bcd
reaffiche2
        call    affichage       ; appel de la routine d'affichage
        btfsc   plus            ; tester si le bouton "plus" est appuyé
        goto    tok1            ; si non, sauter vers l'étiquette tok1
        incf    decheur,f       ; si oui, incrémenter la valeur de l'heure
        movlw   24              ;
        subwf   decheur,w       ;
        btfss   STATUS,Z        ; tester si la valeur heure = 24
        goto    affiche2        ; si non, sauter vers l'étiquette affiche2
        clrf    decheur         ; si oui, effacer le registre decheur
        goto    affiche2        ; sauter vers l'étiquette affiche2
tok1
        btfsc   OK              ; tester si le bouton "OK" est appuyé
        goto    reaffiche2      ; si non, sauter vers l'étiquette affiche2
        btfsc   indmarche,0     ; si oui, tester si le bit "0" de indmarche = 0
        goto    jsuivant        ; si non, sauter vers l'étiquette jsuivant
        call    CODAGE          ; si oui, codé la valeur de la minute et de l'heure
        call    tempo           ; appeler la routine de temporisation
        incf    indmarche,f     ; incrémenter la valeur de registre indmarche
        incf    jourvar,f       ; incrémenter la valeur de registre jourvar
```

```
        goto        etq100          ; sauter vers l'étiquette etq100
jsuivant
        call        CODAGE          ; coder la valeur de la minute et de l'heure
        call        tempo           ; appeler la routine de temporisation
        clrf        indmarche       ; effacer la registre indmarche
        incf        jourp,f         ; incrémenter le registre jourp
        movlw       31              ;
        subwf       jourp,w         ;
        btfsc       STATUS,Z        ; tester si jourp = 31
        goto        regtemps        ; si non, sauter vers l'étiquette regtemps
        movf        jourp,w         ;si oui, charger jourp dans le registre w
        goto        etq10           ; sauter vers l'étiquette etq10
```

c) routine de réglage du temps

Au moment de l'exécution de la routine de programmation, toute autre interruption est inhibée, cela pose un autre problème. Les interruptions générées par le débordement du timer ne sont pas prises en compte, cela veut dire que la durée de programmation ne sera pas prise en compte dans le calcul du temps. Pour remédier à ce problème on a introduit cette routine de réglage après la routine de programmation pour que l'utilisateur puisse corriger ce retard. Cette routine nous permet aussi de faire le réglage sans passer par la routine de programmation RB0, simplement par l'appui simultanément sur les boutons "RB0" et "plus", ceci allume la LED "minute" indiquant le réglage de la minute, l'appui sur le bouton "plus" incrément la valeur de la minute et l'appui sur "OK" l'enregistre et allume la LED "heure" pour passer au réglage de l'heure, après le réglage de l'heure on passe au réglage du jour. Pour sortir de cette interruption il suffit d'appuyer à nouveau sur la bouton "OK".

Sous-programme de réglage du temps

```
regtemps
        movlw       15              ; effacer les afficheurs 3 et 4
        movwf       afficheur3      ;
        movwf       afficheur4      ;
        movf        etaPB,w         ;
        movwf       PORTB           ;
        movlw       0xfe            ;
```

```
        iorwf   PORTA,f         ; allumer la LED "minute"
        bsf     ledmin          ;       sans affecter la charge
        bcf     controle        ; sélectionner la partie affichage
        movf    minutes,w       ;
        movwf   afvar           ; convertir la valeur
        call    BCD             ;       minute au bcd
reaffiche3
        call    affichage       ; appeler la routine d'affichage
        btfsc   plus            ; tester si le bouton "plus" est appuyé
        goto    tok2            ; si non, sauter vers l'étiquette tok2
        incf    minutes,f       ; si oui, incrémenter les minutes
        movlw   0x3c            ;
        subwf   minutes,w       ;
        btfss   STATUS,Z        ; tester si minutes = 60
        goto    intrtime        ; si non, sauter vers l'étiquette intrtime
        clrf    minutes         ; si oui, effacer le registre minutes
        goto    intrtime        ; sauter vers l'étiquette intrtime
tok2
        btfsc   OK              ; tester si le bouton "OK" est appuyé
        goto    reaffiche3      ; si non, sauter vers l'étiquette reaffiche3
affiche4
        movf    etaPB,w         ; si oui, allumer la LED "heure"
        movwf   PORTB           ;       sans affecter la charge
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bsf     ledheur         ;
        bcf     controle        ; sélectionner la partie affichage
        movf    heurs,w         ;
        movwf   afvar           ;
        call    BCD             ;  convertir  la valeur heure en bcd
reaffiche4
        call    affichage       ; appeler la routine d'affichage
        btfsc   plus            ; tester si le bouton "plus" est appuyé
        goto    tok3            ; si non, sauter vers l'étiquette tok3
```

```
        incf    heurs,f         ; si oui, incrémenter la valeur heure
        movlw   0x18            ;
        subwf   heurs,w         ;
        btfss   STATUS,Z        ; tester si la valeur heure = 24
        goto    affiche4        ; si non, sauter vers l'étiquette affiche4
        clrf    heurs           ; si oui, effacer le registre heure
        goto    affiche4        ; sauter vers l'étiquette affiche4
tok3
        btfsc   OK              ; tester si le bouton "OK" est appuyé
        goto    reaffiche4      ; si non, sauter vers l'étiquette reaffiche4
        movf    etaPB,w         ; si oui, éteindre toutes les LEDs
        movwf   PORTB           ;     sans affecter la charge
        movlw   0xfe            ;
        iorwf   PORTA,f         ;
        bcf     controle        ; sélectionner la partie affichage
affiche5
        movf    jours,w         ;
        movwf   afvar           ;
        call    BCD             ; convertir la valeur jours en bcd
reaffiche5
        call    affichage       ; appeler la routine d'affichage
        btfsc   plus            ; tester si le bouton "plus" est appuyé
        goto    tok4            ; si non, sauter vers l'étiquette tok4
        incf    jours,f         ; si oui, incrémenter la valeur jours
        movlw   31              ;
        subwf   jours,w         ;
        btfss   STATUS,Z        ; tester si  la valeur jours = 31
        goto    affiche5        ; si non, sauter vers l'étiquette affiche5
        movlw   1               ; si oui, charger le registre jours
        movwf   jours           ;       par la valeur 1
        goto    affiche5        ; sauter vers l'étiquette affiche5
tok4
        btfsc   OK              ; tester si le bouton "OK" est appuyé
        goto    reaffiche5      ; si non, sauter vers l'étiquette affiche5
```

```
movf    etaPB,w     ; si oui, allumer toutes les LEDs
movwf   PORTB       ;       sans affecter la charge
movlw   0xfe        ;
iorwf   PORTA,f     ;
bsf     ledmin      ;
bsf     ledheur     ;
bsf     ledon       ;
bsf     ledoff      ;
movlw   2           ;
andwf   PORTB,f     ;
bcf     controle    ; sélectionner la partie affichage
clrf    secondes    ; effacer le registre des secondes
movlw   15          ; charger le registre "cmpt" par la
movwf   cmpt        ;       valeur 15
return              ; fin d'interruption RB0/INT
```

d) routine d'interruption RS232

La liaison RS232 permet à notre carte de communiquer avec un PC à l'aide des deux fils d'émission(TX) et de réception(RX). Le fil d'émission (TX) transmet les données programmées dans l'EEPROM vers le PC tandis que le fil de réception (RX) reçoit les données du PC pour les programmer dans l'EEPROM. Pour que cette communication s'effectue sans erreur, elle doit respecter un protocole de communication compatible avec les normes RS232. Ces normes sont:

- **Longueur des mots :** 7 bits (ex : caractère ascii) ou 8 bits
- **La vitesse de transmission :** les différentes vitesses de transmission retenues sont réglables à partir de 110 bauds (bits par seconde) de la façon suivante : 110 bds, 150 bds, 300 bds, 600 bds, 1200 bds, 2400 bds, 4800 bds, 9600 bds.
- **Parité** : le mot transmis peut être suivi ou non d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux types de parité ;

parité paire : le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit paire sur l'ensemble donné + bit de parité

parité impaire : le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit impaire sur l'ensemble donné + bit de parité

- **Bit de start :** la ligne au repos est à l'état logique 1 pour indiquer qu'un mot va être transmis la ligne passe à l'état bas avant de commencer le transfert. Ce bit permet de synchroniser l'horloge du récepteur.
- **Bit de stop :** après la transmission, la ligne est positionnée au repos pendant 1, 2 ou 1,5 périodes d'horloge selon le nombre de bits de stop.

Format des trames :

Le bit de start apparaît en premier dans la trame puis les données (poids faible en premier), la parité éventuelle et le (les) bit(s) de stop.

Figure 21 : Format d'une trame en transmission RS232

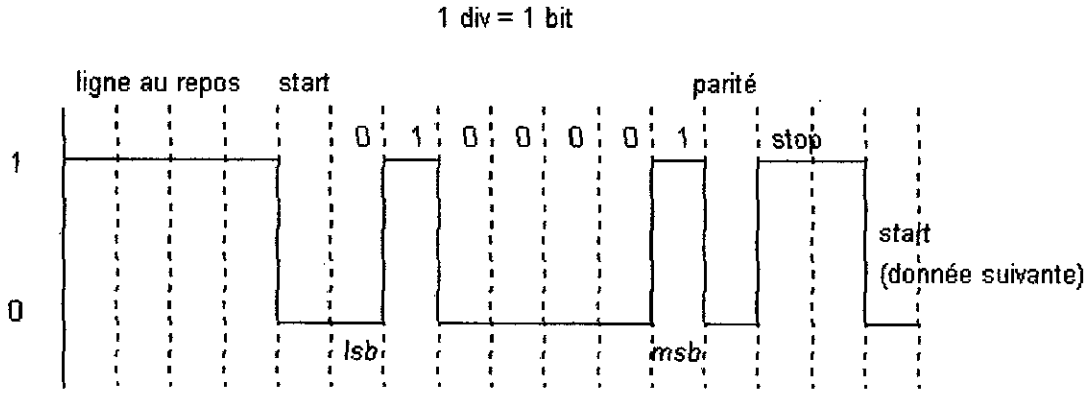
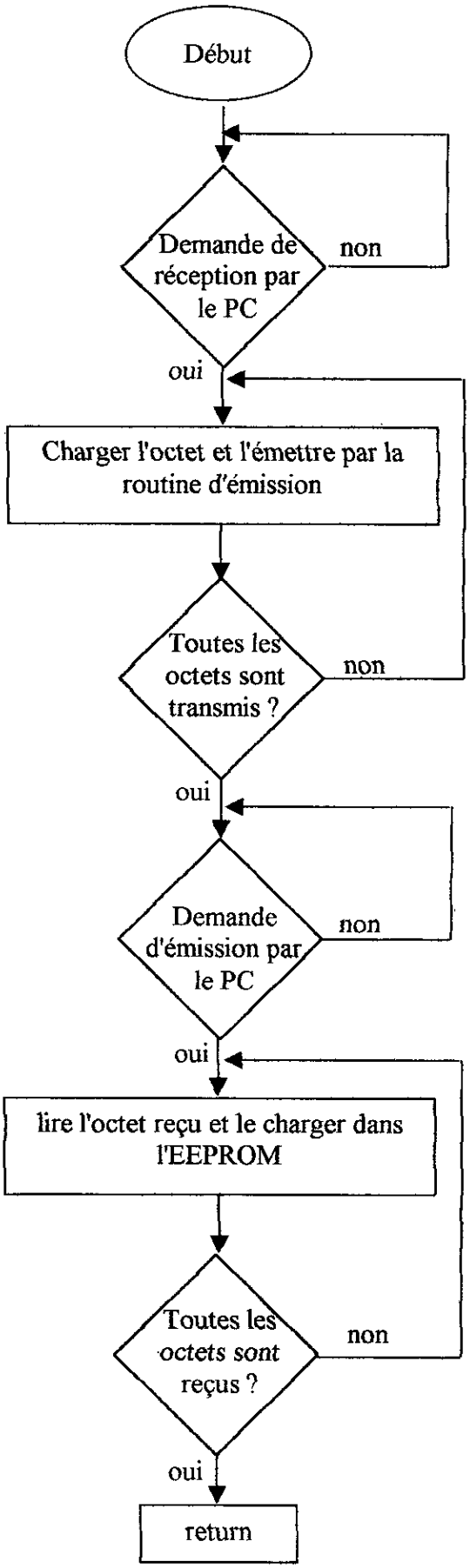


Figure 22: organigramme de la routine RS232



Sous-programme d'interruption RS232

```
RS232
        btfsc     receivepin      ; Tester si le bit de demande de réception
        goto      RS232           ; non, attendre
        movlw     30              ; charger le compteur "cmpt1" par 30
        movwf     cmpt1           ;
        clrf      jourvar         ; effacer le registre jourvar
et1     READEE    jourvar         ; lecture de la donnée de l'EEPROM
        call      send            ; appel de la routine de transmission des données
        incf      jourvar         ; incrémenter jourvar
        btfsc     jourvar,0       ; tester si le bit'0' de jourvar =0
        goto      et1         ; si oui, lecture de la prochaine donnée de l'EEPROM
        movf      cmpt1,w         ; si non, lecture de la donnée codée dans l'adresse
        movwf     FSR             ;       RAM correspondante à ce jour
        movf      INDF,w          ;
        call      send            ; appel de la routine de transmission des données
        incf      cmpt1,f         ; incrémenter le compteur"cmpt1"
        movlw     60              ;
        subwf     jourvar,w       ;
        btfsc     STATUS,Z        ; tester si toutes les données sont transmises
        goto      et1             ; si, non transmettre les données qui restent
        movlw     30              ; charger le compteur "cmpt1" par 30
        movwf     cmpt1           ;
        clrf      jourvar         ; effacer le registre jourvar
et2     call      receive         ; appel de la routine de réception des données
        WRITEE    jourvar         ; écriture de la donnée reçu
        incf      jourvar         ; incrémenter jourvar
        btfsc     jourvar,0       ; tester si le bit'0' de jourvar =0
        goto      et2             ;si oui, écriture de la prochaine donnée dans
                                  ;       l'EEPROM
        movf      cmpt1,w         ; incrémenter le compteur "cmpt"
        movwf     FSR             ;       correspondante à ce jour
```

```
call    receive     ; appel de la routine de réception des données
movwf   INDF        ;  écriture de la donnée dans la RAM
incf    cmpt1,f     ;
movlw   60          ;
subwf   jourvar,w   ;
btfsc   STATUS,Z    ; tester si toutes les données sont reçues
goto    et2         ; si non, continuer la réception des données
                    ;                restantes
return              ; si oui, fin de l'interruption
```

LE TEST DE LA CARTE

Après la réalisation de la carte, on a introduit notre programme dans le microcontrôleur à travers le port série d'un PC par le logiciel de programmation de PIC *IC-Prog*, puis on a placé le µC dans la carte et on l'a alimenté par un générateur de tension pour le tester.
Après des tests il s'avère que notre programme marche bien à part une remarque qu'on peut signaler sur la montre du système.
Nous avons remarqué que la montre est un peux rapide, donc elle présente un peux d'avance et cela est remarquable à partir d'une journée de marche. Cela est peut être du à la plage d'erreur de notre quartz .

CONCLUSION

Le programmateur horaire réalisé dans ce projet est un prototype de base qui peut servir dans d'autres applications industrielles. Il peut être amélioré en lui ajoutant quelques modifications dans les parties hardware et software.

On propose de remplacer les LEDs d'indication par un afficheur LCD qui offre une meilleure visualisation dans le sens de la quantité d'informations qui peut nous donner par rapport aux LEDs. D'autre part on suggère aussi d'ajouter un circuit qui effectue le calcul de la base du temps au lieu de le faire avec le µC cela peut améliorer considérablement la précision de la montre.

REFERENCES BIBLIOGRAPHIQUES :

[1] BIGONOFF, *La Programmation des PIC*, site Web :www.abcelectronique.com/bigonoff.

[2] C. TAVERNIER, *Applications Industrielles des PIC*, Paris, DUNOD, 2001.

[3] PIC16F8X Data Sheet, *Microchip*, 1999.

[4] ULN2003 Data Sheet, *Texas Instruments*, 2003.

[5] MAX232 Data Sheet, *Maxim*, 2001.

ملخص:
هذا المشروع يهدف الى دراسة و انجاز مبرمج زمني لتشغيل و توقيف حمولة واحدة، و يتضمن برنامجا واحدا لكل يوم لمدة 30 يوما. وينبني على مراقب دقيق 16F84 وملحقات لضبط التغذية و الوقت و المؤشرات الضوئية ، ويحتوي على صنفين من البرمجة :
برمجة يدوية عن طريق ازرار وبرمجة عن طريق الحاسوب بواسطة رابطة RS232.

كلمات أساسية : مبرمج زمني ، مراقب دقيق 16F84 ، رابطة RS232، حمولة.

Abstract

This work consist to study and realize atimer programmer which allow to switch on or switch off one device during 30 days. This programmer can be prgrammed by two kinds of méthodes:
Manually by a switch or by a Pc with RS232 connection.

Key words : 16F84 microcontroller , timer programmer, simulation, adaptive control.

Résumé :

Ce projet fait l'étude et la réalisation d'un programmateur horaire à base d'un microcontrôleur PIC16F84 qui permet de mettre en service ou d'arrêter à de moments programmés(minutes, heure, jours) un appareil, pendant 30 jours. La programmation de ce dernier ce fait de méthodes. Une programmation manuelle avec des boutons poussoirs ou avec un ordinateur (liaison RS232).

Mots clés : microcontrôleur PIC16F84, liaison RS232, programmateur horaire.