OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

DIRECTION RECHERCHE ET INGENIERIE DE FORMATION

RESUME THEORIQUE
&
GUIDE DE TRAVAUX PRATIQUES

MODULE N°:21 **MICROCONTROLEUR**

SECTEUR : ***GENIE ELECTRONIQUE***

SPECIALITE ***: MMOAMPA***

NIVEAU : ***T.S.***

Document réalisé par :

Nom et prénom	*EFP*	*DR*
PANTAZICA LUCRETIA	*CDC- Electrotechnique*	*DRIF*

Révision linguistique

-
-
-

Validation

MICROCONTROLEUR

OBJECTIF OPÉRATIONNEL DE PREMIER NIVEAU

DE COMPORTEMENT

COMPORTEMENT ATTENDU

Pour démontrer sa compétence le stagiaire doit
savoir programmer en langage C ou langage orienté objet un microcontrôleur selon les conditions, les critères et les précisions qui suivent.

CONDITIONS D'ÉVALUATION

- Individuellement
  - A partir de directives et de logiciels
- À l'aide :
  - D'un micro-ordinateur
  - De la documentation des logiciels utilisés
  - D'une imprimante.

CRITÈRES GÉNÉRAUX DE PERFORMANCE

- Utilisation sécuritaire du poste de travail
- Utilisation adéquate des principales commandes
- Manipulation adéquate des différents menus
- Utilisation méthodique de la documentation
- Réalisation conforme à un cahier des charges
- Travail soigné

(à suivre)

OBJECTIF OPÉRATIONNEL DE PREMIER NIVEAU DE COMPORTEMENT

PRÉCISIONS SUR LE COMPORTEMENT ATTENDU	CRITÈRES PARTICULIERS DE PERFORMANCE
A. Utiliser correctement un ordinateur et ses périphériques, et connaitre les principaux langages informatiques.	- Présentation optimale de la structure interne et externe d'un ordinateur. - Description des langages informatiques. - Utilisation correcte des périphériques et des interfaces d'entrées / sorties.
B. Réaliser un diagnostique précis sur un système à microprocesseur et à microcontrôleurs.	- Maîtrise la structure de base d'un microprocesseur et d'un microcontrôleur. - Présentation optimale du rôle de chaque bloc composant d'un microprocesseur et d'un microcontrôleur. - Utilisation adéquate des instructions assembleur pour les microprocesseurs de la famille 8086 et pour les microcontrôleurs 16F87X. - Réalisation des petits programmes pour les microcontrôleurs 16F87X.
C. Programmer un microcontrôleur.	- Utilisation correcte d'un éditeur de texte. - Utilisation correcte des compilateurs. - Réalisation des simulateurs ou des simulations pour tester le programme sur l'ordinateur. - Utilisation correcte de l'assembleur en ligne. - Manipulation adéquate de fonctions externes. - Utilisation correcte de l'éditeur de liens. - L'écriture correcte d'une fonction de la bibliothèque.
D. Utiliser un logiciel de programmation pour les microcontrôleurs	- Utilisation adéquate du logiciel. - Implantation d'un algorigramme - Identification des entrées sorties du microcontrôleur et des variables internes utiles

OBJECTIFS OPÉRATIONNELS DE SECOND NIVEAU

LE STAGIAIRE DOIT MAÎTRISER LES SAVOIRS, SAVOIR FAIRE, SAVOIR PERCEVOIR OU SAVOIR ÊTRE JUGÉS NECESSAIRES AUX APPRENTISSAGES DIRECTEMENT REQUIS POUR L'ATTEINTE DE L'OBJECTIF DE PREMIER NIVEAU, TELS QUE :

Avant d'apprendre à utiliser correctement un ordinateur et ses périphériques, et connaitre les principaux langages informatiques (A) :

1. Identifié la structure interne et externe d'un ordinateur.
2. Connaitre les langages informatiques.
3. Identifier et utiliser les périphériques et les interfaces d'entrées / sorties.

Avant d'apprendre à réaliser un diagnostique précis sur un système à microprocesseur et à microcontrôleurs (B) :

4. Maitriser la structure de base d'un microprocesseur et d'un microcontrôleur.
5. Présenter le rôle de chaque bloc composant d'un microprocesseur et d'un microcontrôleur.
6. Utiliser les instructions assembleur pour les microprocesseurs de la famille 8086 et pour les microcontrôleurs 16F87X.
7. Réaliser des petits exemples de programmes en C.

Avant d'apprendre à utiliser la programmation des PIC (C) :

8. Savoir utiliser un éditeur de texte, un assembleur, un compilateur et un simulateur.
9. Présenter la composition d'un programme en C intégré.
10. Utiliser les fonctions externes et les l'éditeur de liens.
11. Écrire une fonction de la bibliothèque.

Avant de comprendre l'utilisation un logiciel de programmation pour les microcontrôleurs. (D) :

12. Présenter le logiciel de programmation.
13. Savoir réaliser et Implanter un algorigramme.
14. Identifier les entrées/sorties du microcontrôleur et des variables internes utiles.

TABLE DES MATIÈRES

Chapitre 1

L'ORDINATEUR ET SON ENVIRONEMENT

1. STRUCTURE D'UN ORDINATEUR

En 1935 à 1950 deux mathématiciens ont joué un rôle important dans l'invention de l'ordinateur. Il s'agit de l'anglais Alan TURING et de l'américain John von NEUMANN. Ce dernier a donné son nom à l'architecture des ordinateurs modernes. Elle fut définie dans un texte qu'il écrivit en 1945.
On insiste sur l'architecture de Von Neumann, par la suite, pour décrire de façon simple le fonctionnement d'un ordinateur, d'un système à microprocesseur, ou d'un microcontrôleur. Bien que cette architecture ne soit pas la seule mise en œuvre de nos jours, elle permet d'avoir une vision simple des choses.
Autre architecture : architecture de Harvard.

Structure d'un ordinateur.

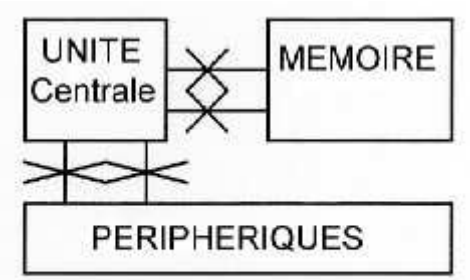


Diagramme de Von Neumann.

Un ordinateur est donc constitué :

- d' Une Unité Centrale (UC ou CPU en anglais) C'est l'unité centrale de traitement des informations. Son rôle est d'aller chercher, automatiquement les instructions du programme en mémoire et de les exécuter.
- d' Une Mémoire Centrale, qui sert au stockage des données et des programmes.
- de Périphériques : ou unités d'Entrées/Sorties qui servent à communiquer avec l'extérieur (ex : clavier, écran, souris, ...).

Le système ainsi constitué doit respecter les 4 règles suivantes :

1. Les instructions et les données sont dans une mémoire unique, banalisée, accessible en lecture/écriture.
2. Les contenus de la mémoire sont accessibles par leurs adresses.
3. La commande de l'ensemble, l'exécution des opérations se fait de manière séquentielle (sauf indication expresse). L'exécution d'une opération doit être terminée avant le lancement de la suivante.
4. L'unité de traitement contient un jeu complet des opérations de l'algèbre de Boole.

REMARQUES : L'Unité Centrale est le coeur du système. C'est un système électronique, mais, seule, l'Unité Centrale ne peut rien faire, il faut lui donner un programme à exécuter. Il faut donc de la mémoire pour contenir le programme à exécuter et les données sur lesquelles on désire travailler.
L'ensemble Unité Centrale et Mémoire peut déjà fonctionner, mais cela ne servirait à rien : il faut que l'on puisse donner des ordres au programme et récupérer des résultats. Il faut que le système communique avec l'extérieur, d'où la présence indispensable des périphériques.

A. Unité Centrale.

Elle peut être décomposée en 2 sous ensembles :

1. **L'Unité de Contrôle** (ou unité de commande, ou automate) :
Son rôle est d'aller chercher une instruction en mémoire centrale, d'analyser cette instruction (décodage de l'instruction), d'exécuter cette instruction, de localiser l'instruction suivante (opération souvent implicite : adresse immédiatement supérieure à celle de l'instruction en cours d'exécution, sauf pour les instructions de branchement). L'Unité de Contrôle contient donc :
   - un décodeur d'instruction.
   - un séquenceur et des circuits de commandes.
2. **Un ensemble de circuits électroniques, commandés par l'unité de contrôle** et permettant :
   - d'échanger des informations avec la mémoire centrale et avec le monde extérieur (avec les périphériques).
   - d'effectuer des opérations sur les données (Unité Arithmétique et logique : U.A.L. ou A.L.U. en anglais).
   - de mémoriser l'adresse de la prochaine instruction dans un registre particulier PC (Compteur de programme (Program Counter)).
   - de mémoriser le résultat d'opérations dans des mémoires spéciales : les Registres de travail.

EXEMPLE : L'unité centrale lit l'instruction LD A,100 en mémoire centrale.
Elle décode l'instruction : il faut charger le registre A avec la valeur contenue dans la case mémoire n° 100. L'unité centrale lance l'exécution : demande de lecture de la mémoire à l'adresse 100. Elle récupère la valeur lue et enfin range cette valeur dans A. Elle met à jour le compteur de programme PC, et l'on continue : lecture de l'instruction suivante ...

Une *horloge* (un circuit oscillateur délivrant des impulsions à une certaine fréquence) sera nécessaire pour exécuter les opérations séquentiellement. Plus la fréquence sera grande, plus l'unité centrale travaillera vite ! (N.B.: Fréquence = 8 MHz se lit 8 Mega Hertz et veut dire 8 millions d'impulsions par seconde.)

B. Mémoires.

Les programmes, les données, vont être stockés *en binaire*, (suite de 0 et de 1).
Les différents types de mémoires :

1. **Les mémoires mortes** (à lecture seule) : Ces mémoires ont la caractéristique importante de conserver l'information même lorsqu'elles ne sont pas alimentées, et ceci pendant des années et même indéfiniment pour certaines.

- R.O.M.: de Read Only Memory. On en peut que les lire. Ces mémoires sont écrites et mises en place par le constructeur (BIOS des PCs, par ex.)

- P.R.O.M.: Programmable Read Only Memory. On les appelle aussi mémoires fusibles, ou encore OTP (One Time Programmable). On achète ces mémoires vierges (vides de tout programme) et on peut les programmer à l'aide d'un programmateur (une simple carte électronique) relié à un PC via un port série (COM1, COM2,...) ou via le port parallèle. On les place ensuite sur le système auquel elles sont destinées. Si l'on a fait une erreur dans le programme, ou si l'on désire le modifier, la mémoire est bonne pour la poubelle, on ne peut pas la programmer une deuxième fois ! Il faut en prendre une nouvelle. **La programmation** de ce type de composant consiste à faire "claquer" des fusibles (qui sont en fait des jonctions semi-conductrices). On comprend que lorsque le fusible est détruit il est impossible de faire machine arrière !
  On les utilise donc lorsque l'on a un programme parfaitement au point, définitif. Elles ont l'avantage d'être peu coûteuses.

- E.P.R.O.M.: Erasable Programmable Read Only Memory. Comme les PROM, on peut écrire dans ces mémoires (avec un dispositif similaire), mais, si le programme n'est pas correct, il est possible d'effacer le programme, puis de l'écrire à nouveau (ceci un très grand nombre de fois)
  C'est bien sur très intéressant, et ce sont ces mémoires que l'on va utiliser pour faire du développement.
  Ces mémoires ont une petite fenêtre transparente par laquelle on voit la puce. Cette fenêtre sert à effacer la mémoire. On éclaire le composant avec des UV, dans une boite (effaceur à UV), pendant une quinzaine de minutes environ.
  Il faut noter que l'effacement (environ 15 min) et l'écriture (quelques minutes) sont des opérations relativement longues. (Principe utilisé : utilisation de transistors FAMOS : Floating gate Avalanche injection Metal Oxide Silicium.)

- E.E.P.R.O.M. : Electrically Erasable PROM, ou aussi mémoires FLASH ces mémoires sont effaçables par impulsions électriques. L'effacement est plus rapide que lors d'une exposition aux UV, et il est possible d'effacer et de réécrire la mémoire in situ.

2. **Les mémoires vives (à lecture et écriture) :** On les appelle, de façon impropre, R.A.M. (Random Access Memory : mémoireà accès aléatoire). Les PROMs, EPROMs, ... sont, elles aussi à accès aléatoires ! Ces mémoires perdent les informations contenues dès que l'on coupe leur alimentation. (sauf s'il s'agit de mémoires secourues (pile, accu)).
   On distingue : - les **Mémoires Statiques** (S.R.A.M.),

   - les **Mémoires Dynamiques** (D.R.A.M.).

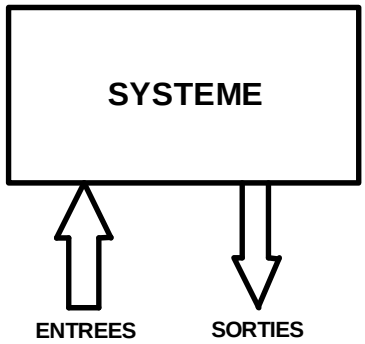
Les mémoires statiques sont réalisées à l'aide de circuits bistables (ou bascules RS). Les mémoires dynamiques conservent l'information, 0 ou 1, dans une petite capacité, qui se décharge au cours du temps (courants de fuite). Ces mémoires nécessitent donc un dispositif de rafraîchissement pour maintenir l'information. Ce sont les plus utilisées car elles permettent une grande densité d'intégration.

Caractéristiques importantes des mémoires :

- Leur CAPACITE : c'est la quantité d'information que le composant peut contenir. Cette capacité est souvent donnée en bits. Exemple : Mémoire 8k = mémoire 8k bits = 1 k octets.
- ORGANISATION, ADRESSAGE : comment on récupère l'information.
  - En parallèle (tous les bits d'un coup) et en série (bit a bit)
  - L'adresse est donnée en une fois, en 2 fois (adressage multiplexé)
- TEMPS D' ACCES : c'est le temps qui s'écoule entre la demande d'une valeur en mémoire et l'instant où cette valeur est disponible.

C. Les périphériques d'entrées/sorties.

Les périphériques vont permettre la communication avec le système Unité Centrale + Mémoires.



On distingue 3 types de périphériques : **les périphériques d'Entrées, les périphériques de Sorties, et ceux qui font les 2 (Entrées/Sorties).**

1. Les périphériques d'entrées sont ceux qui apportent des informations au système ; les périphériques de sorties envoient des informations à l'extérieur du système.

Exemple des périphériques d'entrées : le clavier (indispensable), la souris, un light pen, une table à digitaliser, un scanner, un lecteur de cartes, un lecteur de *codes barre,...*
N.B. *Pour un petit système à microcontrôleur, un interrupteur, un inverseur, un bouton poussoir, un rotacteur, sont des périphériques d'entrées !*

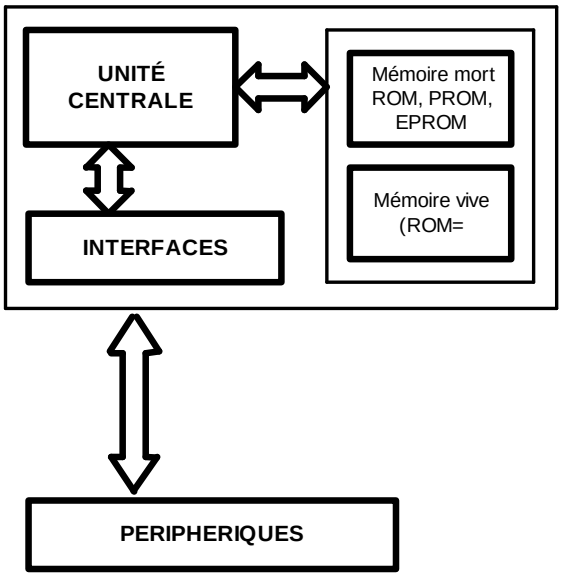
2. Périphériques de sorties : les classiques : un écran (ou moniteur), une imprimante, une table traçante, ...
N.B. *Pour un petit système à microcontrôleur, un voyant, une LED, des afficheurs 7 segments, un HP (haut Parleur), sont des périphériques de sorties !*

3. Périphériques d'entrées / sorties ou mémoires de masse : car ils vont servir à stocker des masses d'informations. On mettra dans cette catégorie les lecteurs de disquette, les disques durs, les lecteurs enregistreurs de bande magnétique, ...

N.B. *Les périphériques ne sont pas reliés directement à l'ensemble Unité Centrale / Mémoires, il faut passer par une INTERFACE. (carte interface, circuit interface, coupleur de périphérique). Il y a quelques années, une interface était une carte électronique qui pouvait être de taille importante, de nos jours, une interface se réduit souvent à un seul circuit intégré, programmable.*

L'interface se charge de l'adaptation des signaux électroniques, de gérer le dialogue entre le système et le périphérique (protocole d'échanges).

On peut donc détailler un peu le diagramme de Von Neumann, de la façon suivante :



D. Les "Bus".

L'Unité Centrale doit pouvoir communiquer avec les mémoires et les périphériques. Par exemple, pour écrire une donnée en mémoire, l'U.C. doit d'abord spécifier l'adresse de la mémoire, puis envoyer la donnée, et, en dernier lieu, envoyer un signal d'écriture qui validera la mémorisation de la donnée. Tous ces signaux seront véhiculés par les "Bus", ensembles de "conducteurs", sur lesquels viennent se brancher les mémoires, les interfaces des périphériques.

On distingue 3 types de "bus" :

- Le bus d'adresses.
- Le bus de données.
- Le bus de contrôle (pour les signaux de service).

E. Interface d'entrée / sortie

Le micro-ordinateur se transforme assez facilement en un automate programmable à partir de l'instant ou il dispose de cartes d'entrée / sorties. Des cartes peuvent être de deux types différents : les cartes d'entrées / sorties numériques et les cartes d'entrées / sorties analogiques.

a**) Cartes d'entrées / sorties numériques**

Des interfaces sont chargées de mémoriser l'état d'un certain nombre d'entrées. Ces états peuvent être transférer à l'unité centrale pour une exploitation quelconque. L'opération inverse est également possible : l'écriture d'un certains nombre d'états de la CPU vers les sorties.

Le nombre d'entrées / sorties n'est pas limité que par la conception de la carte d'interface. La conception d'une carte 32 entrées – 32 sorties ne pose pas de problème majeur.

Du point de vue électrique, les entrées / sorties peuvent être :

- compatible TTL,
- boucles de courant 0-20 mA ou 4-20 mA,
- opto – isolées,
- à relais,
- etc.

Ces cartes relient le « mode extérieur » au micro-ordinateur. La conception de l'automate se résume à l'écriture d'un programme d'acquisition des valeurs d'entrées, d'un certain nombre de calculs et d'écriture des valeurs de sortie.

b) Cartes d'entré / sortie analogiques

La plupart des capteurs délivrent des signaux analogiques : capteurs de température, force, pression, position,…etc. Les grandeurs analogiques sont préalablement converties en numériques pour transiter sur le bus du PC et être traités par la CPU.

De la même manière, il peut être intéressant de disposer en sortie d'une information analogique destinée par exemple à un moteur ; pour les entrées on dispose donc de convertisseurs analogiques-numériques ; pour les sorties, les convertisseurs sot de type numériques-analogiques.

Les caractéristiques essentielles de ses cartes sont la résolution : largeur du mot sur lequel le résultat est exprimé, en général compris entre 8 et 16 bits ; cadence d'échantillonnage, variable en fonction du type de convertisseur mis en service.

F. Interface réseau

Les cartes d'entrée / sortie sont un premier exemple de connexion du microordinateur au monde extérieur. Un deuxième type de connexion est la connexion de micro-ordinateurs entre eux.

On peut distinguer deus types de connexions, les connexions locales ou à la distance.

a) Réseau locaux LAN (Local Area Network)

Un réseau local permet de connecter un groupe de micro-ordinateurs et éventuellement un certain nombre de périphériques comme les imprimantes et les traceurs. Le meilleurs exemple de réseau local est le réseau ETHERNET qui, sur un câble coaxial assez classique, autorise des transmissions à 10 M bits / s.

Le principal intérêt de réseau local réside dans le partage de ressources : chaque station de travail peut être équipée des ressources minimales – mémoire de masse par exemple – et une des stations peut être dédiée au stockage.

Les traceurs, extrêmement couteux, peuvent être partagés par plusieurs utilisateurs.

Les utilisateurs connectés en réseau local peuvent échanger des fichiers ; si ces fichiers sont des simples messages entre les utilisateurs, on parle alors de courrier électronique.

b) Communication à la distance

Pour des communications à plus long distance, le micro-ordinateur est équipé d'un modem « modulateur – démodulateur ».

Une ligne téléphonique suffit alors pour interconnecter plusieurs machines. Les vitesses de transfert sont beaucoup plus faibles que celles des réseaux locaux, de l'ordre de quelque dizaine de K bits par seconde. L'intérêt de ce type de connexion est évident : il peut permettre à plusieurs entreprises de coopérer sur un même projet. Il permet aussi à tout utilisateur d'accéder à des très importantes bases de données, via l'internet par exemple.

2. LES LANGAGES EN INFORMATIQUE.

Il y a une très grande quantité de langages.
On distingue :

- **Les langages de bas niveau** : Langage Machine, Assembleur, Macro Assembleur.
- **Les langages évolués** : BASIC, FORTRAN, COBOL, ALGOL, PASCAL, C, ...

Un langage est défini par :

- **Un lexique** : ensemble des mots valides (mots clefs, identificateurs, ...)
- **Une syntaxe** : Règles d'écriture des instructions. Ensemble des règles d'écriture des phrases appartenant au langage (grammaire du langage).
- **Une sémantique** : signification associée à chaque construction syntaxique valide.

Un langage permet de définir des objets manipulables (entiers, caractères, chaînes, réels, adresses, ...) et d'exprimer des opérations sur ces objets (par des ordres, des instructions qui vont constituer le *PROGRAMME*.)

- ***Le Langage Machine*** : ou binaire, ou Hexa, ou BNPF ...
C'est le seul langage compris par l'électronique (Unité Centrale), mais il est très fastidieux de travailler directement en langage machine, en binaire ou en hexadécimal.
L'assembleur, les langages évolués vont permettre une écriture plus facile, plus cool, et eux vont se charger de la traduction du programme en langage machine, juste avant l'exécution du programme.
Le langage Machine est différent d'un circuit (d'une UC) à l'autre, sauf dans des familles de circuits issus d'un même fabricant.

- ***Le Langage Assembleur*** : C'est une transcription symbolique du langage machine (Mnémonique)
La sémantique est identique à celle du langage machine et l'Assembleur sera aussi différent d'un circuit à l'autre.
Exemple
: LDI A,10 en assembleur ST6 veut dire : charger le registre A avec la valeur 10 en décimal.
: LDI de LOAD Immediate (charger immédiatement). Cette instruction sera traduite en langage machine par: 00010111 00001010 en binaire, ou 17 0A en hexa.
Un Macro-Assembleur autorise les macro-instructions (suite d'instructions que l'on peut insérer en plusieurs endroits d'un programme).

- ***Les Langages évolués*** : permettent une écriture rapide des programmes, ils sont, en général, plus "lisibles" que les programmes en assembleur. Ils permettent de plus l'introduction d'objets nouveaux, d'opérations nouvelles. Ils sont adaptés à certains domaines.
(Ex: on utilisera le COBOL pour un programme de gestion.)

- ***Différence : Interpréteur / Compilateur.***
- Programme réalisé avec un interpréteur : au moment de l'exécution chaque instruction est "interprétée" (traduite en langage machine) et exécutée. Ce processus est recommencé à chaque exécution.
- Un compilateur, lui, interprète toutes les instructions du programme que l'on a écrit (le programme source), et construit un programme en langage machine. Ce programme sera ensuite exécuté, autant de fois que l'on voudra, sans avoir à repasser par la phase d'interprétation.

Les corrections, les modifications sont plus faciles sur un programme interprété, mais un programme compilé sera plus rapide à l'exécution.

Chapitre 2

MICROPROCESSEUR.

Définition
Le microprocesseur, noté aussi M.P.U. (Microprocessor unit) ou encore C.P.U. (Central Processing Unit) est un circuit intégré complexe appartenant à la famille des VLSI (Very large scale intégration) capable d'effectuer séquentiellement et automatiquement des suites d'opérations élémentaires.

Son rôle : Ce circuit remplit deux fonctions essentielles : le traitement des données
On parle d'unité de traitement. Cette fonction est dédiée à l'U.A.L.
Elle concerne la manipulation des données sous formes de transfert, opérations arithmétiques, opérations logiques.... le contrôle du système
Cette fonction se traduit par des opérations de décodage et d'exécution des ordres exprimés sous forme d'instruction.

Puissance d'un microprocesseur
La notion de puissance est la capacité de traiter un grand nombre d'opérations par seconde sur de grands nombres et en grande quantité.
Intrinsèquement la puissance se joue donc sur les trois critères suivants:

- La longueur des mots : données et instructions (on parle de largeur du bus des données).
- Le nombre d'octets que le microprocesseur peut adresser (on parle de largeur du bus des adresses).
- La vitesse d'exécution des instructions liée à la fréquence de fonctionnement de l'horloge de synchronisation exprimée en MHZ.

L'aspect dimensionnel renseigne assez bien de la puissance du composant.

A. SYSTEME A BASE DU MICROPROCESSEUR 6809.

1. Schéma fonctionnel d'un microprocesseur (8 bits)

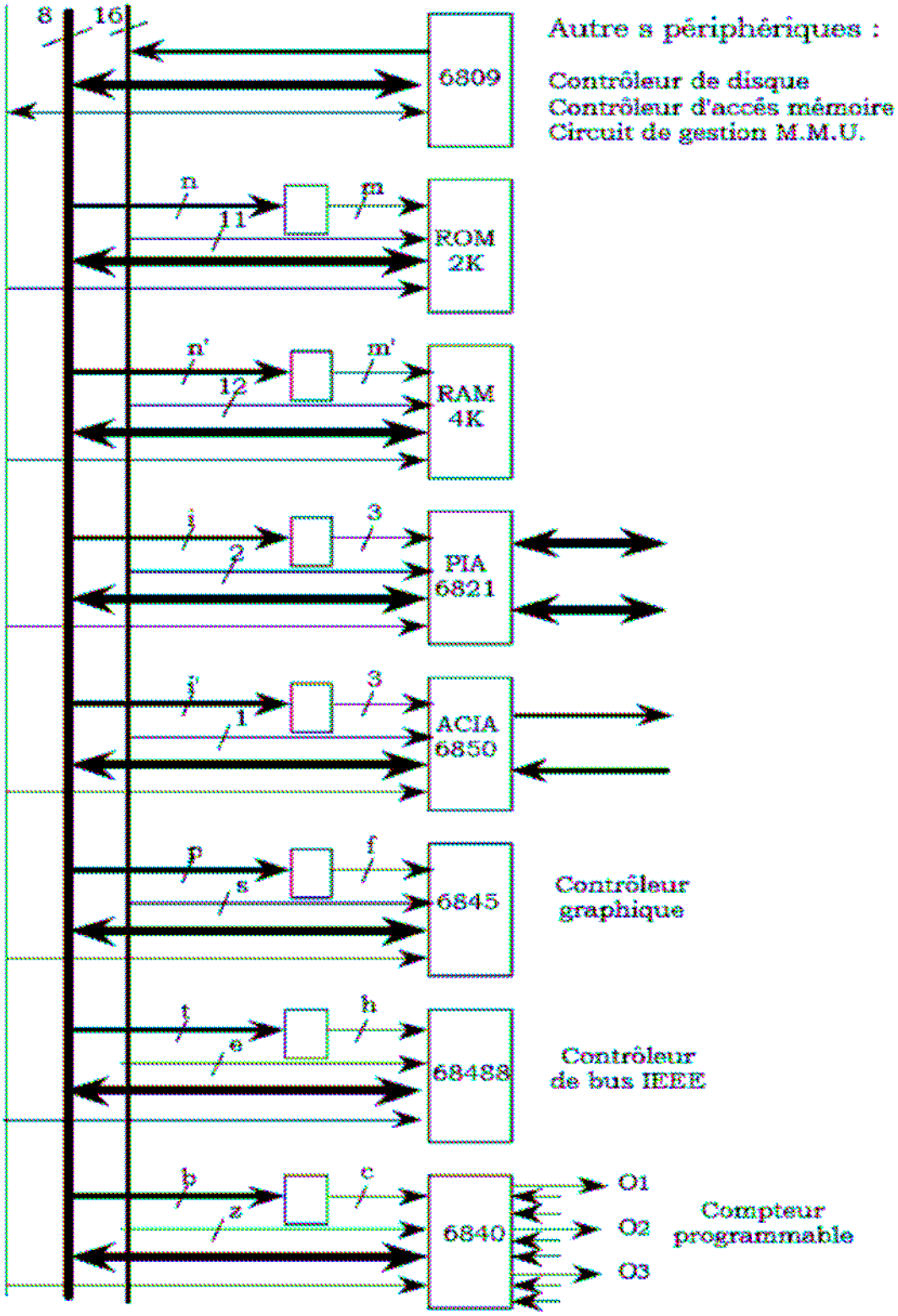
On distingue 3 éléments logiques principaux :

1. Une Unité Arithmétique et Logique (U.A.L.)
2. Un Accumulateur.
3. Des registres que l'on nomme couramment :
    - Le Compteur d'Instructions (C.I.)
    - Le Registre d'état
    - Le Registre d'Instructions (R.I.)
    - Le Registre d'Adresses (R.A.)
    - Le Registre temporaire des données

De base, il existe 6 registres fondamentaux dans une architecture de microprocesseur 8 bits.
(Des registres supplémentaires sont ajoutés pour rendre la vie plus facile aux programmeurs
Cet ensemble est interconnecté au travers de différents bus.
On trouve trois types de bus :

- Le bus des données (bidirectionnel)
- Le bus des adresses (unidirectionnel)
- Le bus de contrôle (bidirectionnel)

Remarquer le bus interne de données qui relie tous les différents éléments du microprocesseur.



1.1 L'Unité Arithmétique et Logique

Son rôle : Ce circuit permet de traiter et tester les données.
Toute instruction qui modifie une donnée fait toujours appel à l'UAL.
L'entrée de L'UAL est connectée au bus interne via :

- des registres "temporaires"
- un registre particulier appelé "accumulateur".

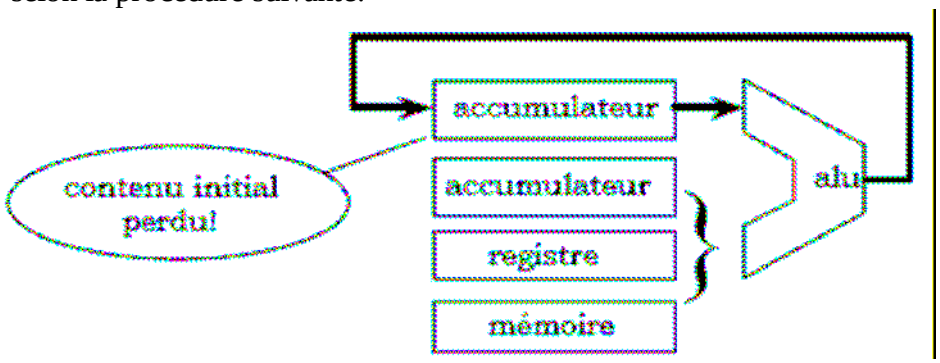
La sortie de l'UAL est connectée uniquement à l'entrée de l'accumulateur.

Noter : les deux entrées sont précédées par une mémoire tampon.
On les appelle encore des registres tampons ou verrou.
Ces registres permettent de stocker des octets aux entrées de l'U.A.L.

L'UAL étant constitué d'une logique combinatoire, elle est dépourvue de moyen propre de stockage.
Ce type de registre ne peut être manipulé par le programmeur. Il lui est totalement transparent.

1.2 L'accumulateur

C'est le registre le plus important du microprocesseur, il sert systématiquement lorsque le microprocesseur a besoin de "manipuler" des données.
La plupart des opérations logiques et arithmétiques sur les données font appel au couple "UAL - accumulateur" selon la procédure suivante:



Il en est de même pour les déplacements et transferts des données d'un endroit à un autre comme :

- de mémoire à mémoire.
- de mémoire à unités d'entrée-sortie (I/O).

Cette action se fait en deux temps : source vers Accumulateur et ensuite Accumulateur vers destination.
Les instructions supportées par un accumulateur sont très nombreuses.
Au niveau de la programmation, il représente une grande souplesse d'utilisation!
Les autres registres ne permettent que des opérations limitées.

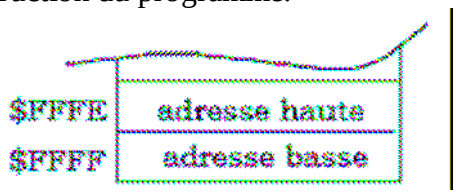
Certains microprocesseur, possèdent des accumulateurs de longueur double tel D chez Motorola et HL chez Intel - dissociés en deux et généralement baptisés individuellement A et B ou H et L respectivement.
Gros avantage présenté par un microprocesseur possédant plusieurs accumulateurs : les opérations logiques et arithmétiques se font entre accumulateurs limitant ainsi les accès (transferts) avec l'extérieur.

1.3 Le Compteur d'Instructions

Appelé encore Compteur Programme (P.C.) ou Compteur Ordinal (C.O.)
Son rôle : Pointer TOUJOURS le premier octet d'une instruction.

Commentaires :
Le programme à exécuter est une succession d'instructions ordonnées (chaque instruction pouvant prend plusieurs octets!) qui se trouve rangé dans une zone mémoire, généralement à des adresses successives.
Le P.C. "repère" le premier octet de chaque instruction du programme.
La taille du PC a une longueur de 16 bits ce qui lui permet d'adresser 65536 adresses mémoire soit 64 k octets (le champ mémoire).
Notons qu'il est connecté au bus interne des données.

A la mise sous tension, une valeur particulière est déposée sur le bus d'adresses (Dans le cas du 6809, cette valeur est $FFFE).
Le contenu des cases mémoires ($FFFE-$FFFF) représente en général l'adresse où se trouve le premier octet de la première instruction du programme.



Cette adresse est transmise aux circuits mémoires par l'intermédiaire du bus d'adresse via le Registre d'Adresses.

Le P.C. pointe toujours l'adresse du début de l'instruction suivante.
(A retenir, car parfois il est utile de connaître l'adresse présente.)
Il est possible de recharger le [P.C.] avec une adresse qui ne correspond pas au déroulement séquentiel du programme.
On trouve les détournements conditionnel et inconditionnel.

1.4 Le registre d'adresses

Son rôle : Le Registre d'adresses ou (R.A.) sert d'interface entre le bus des données interne et le bus des adresses.
Il "pilote" le bus d'adresses du microprocesseur.

Commentaires :
D'une longueur de 16 bits, il est constitué de deux registres 8 bits (partie haute et partie basse).
Son contenu provient de différentes sources :

- Le Compteur d'instruction
- Un registre général
- Un emplacement mémoire
- Le contenu du registre d'adresse pointe la zone mémoire utile au microprocesseur.

Une fois que le premier octet de l'instruction en cours est décodé :

- Le contenu du Compteur d'Instructions est changé (contient l'adresse du début de l'instruction à venir).
- Le contenu du Registre d'adresses change! Donc [C.I.] ≠ [R.A.]

Ce changement correspond :

- soit à une incrémentation du contenu afin de lire l'information complémentaire de l'instruction en cours.

- soit à un chargement d'une nouvelle valeur correspondant à une nouvelle zone mémoire utilisée temporairement par le microprocesseur (zone différente de celle où se trouve le programme).

Cette nouvelle valeur provient soit...

- d'une lecture en mémoire (directement ou indirectement selon les modes d'adressage utilisés)
- d'un calcul (addition, ...)

1.5 Le Registre d'instructions

Sa tâche : Le registre d'instructions contient le premier octet de l'instruction en cours d'exécution.

Commentaires :
Le registre est chargé pendant le cycle de base **extraction - exécution**.
Il reçoit l'information (octet) grâce au bus de données auquel il est connecté.
L'information qu'il "capture" sur le bus des données est utilisé par le décodeur d'instructions.
Suivant le protocole ci-dessous :

- La donnée extraite de la mémoire est stockée dans le R.I. (c'est la phase extraction).
- Ensuite ce contenu est interprété par le décodeur d'instructions qui agit alors sur la logique de contrôle (c'est la phase exécution).

Cet octet indique au microprocesseur deux choses :

- Une action (une lecture, une écriture ou autre ...)
- Un lieu d'action (un registre, un accumulateur, une case mémoire...)

Le résultat de cette interprétation se traduit par des niveaux logiques sur le bus de contrôle.

1.6 Le registre d'état

L'existence de ce registre permet de distinguer le simple calculateur du véritable ordinateur.

Son rôle : Stocker les résultats des tests effectués par l'U.A.L. après traitement sur les données

Commentaires :
L'existence de ces résultats permet d'écrire des programmes avec des branchements conditionnels (nouvelle adresse dans le C.I.).
En fonction de l'état des bits de ce registre le microprocesseur peut, alors, exécuter des programmes différents. le microprocesseur prend en quelque sorte des "décisions".
Les bits les plus couramment utilisés sont :

a) **le bit de retenue :** ce bit est dans l'état actif lorsque le huitième bit du résultat de l'opération génère une retenue.`
b) **le bit de zéro :** ce bit est actif lorsque l'opération a pour effet de mettre tous les bits d'un accumulateur ou d'un registre à la valeur logique 0 (très utilisé pour réaliser des compteurs).
c) **le bit de signe :** information qui indique que le bit le plus significatif (MSB) du contenu de l'accumulateur est un 1 logique.

Exemples d'application : Pour une soustraction - Selon les règles de l'arithmétique du complément à 2 cela signifie que le nombre est négatif.
Ces 3 bits de base sont parfois complétés par des bits supplémentaires choisis par le constructeur.

1.7 Les registres généraux

En plus des 6 registres de base que possèdent tous les microprocesseurs 8 bits, il peut en exister d'autres destinés à faciliter la tâche du programmeur. On les nomme registres généraux.

Sur schéma fonctionnel type donné, nous avons 3 registres généraux B, C et D.
Ce ne sont pas des registres puissants (tel un accumulateur) puisqu'ils n'ont pas de liaison directe avec la sortie de l'U.A.L.
Ces registres peuvent néanmoins affecter le Registre d'Etat.
Parfois, ils peuvent constituer un registre 16 bits - appelé paire de registre (ex : BC chez Intel ou D chez Motorola). Ainsi, il est possible de réaliser des opérations sur un mot (ex : incrémentation de la paire).

1.8 La logique de contrôle

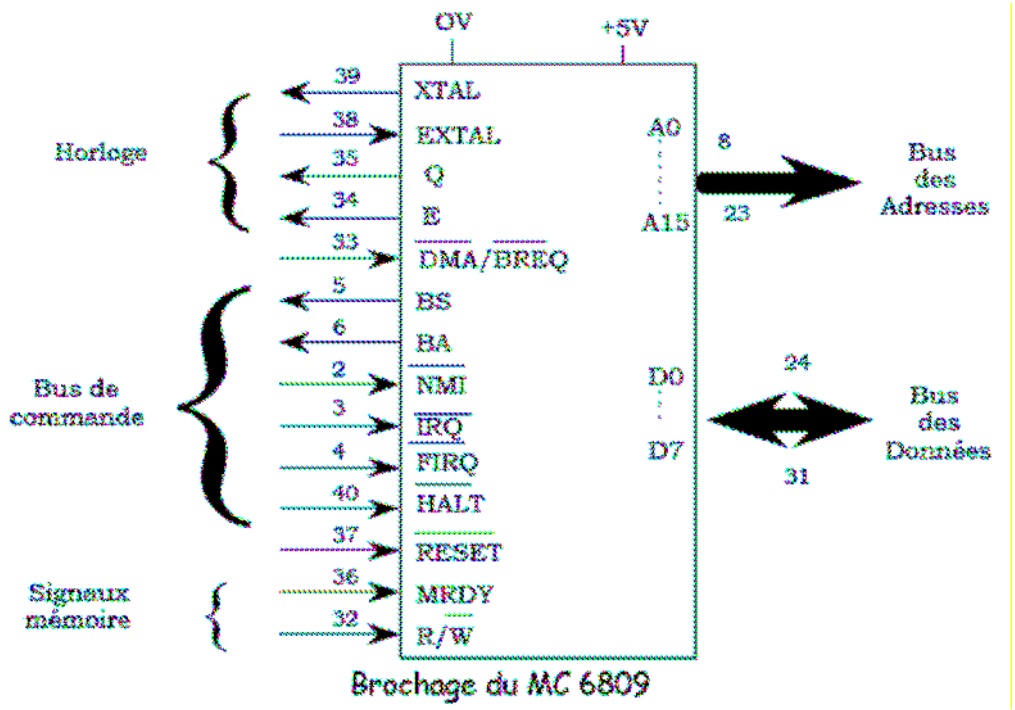
Appelé encore Séquenceur ou Unité de contrôle (U.C.)

Son rôle : Permet à tous les éléments constitutifs du microprocesseur de travailler ensemble et dans l'ordre.
Commentaires :
La logique de contrôle est pilotée par le Registre d'Instruction via le décodeur d'instruction.
Cette unité joue en quelque sorte un rôle d'intendance puisqu'elle décide de la disponibilité du bus à tel ou tel élément logique.
La logique de contrôle possède une architecture complexe et très spécialisée. L'élément central est représenté par le décodeur d'instructions qui décode les informations (premier octet) stockées dans le R.I. pour générer les signaux nécessaires à l'exécution de l'instruction.

La logique de contrôle génère sur les lignes de contrôle des niveaux logiques qui activent les différents circuits environnant tels que mémoires et circuits I/O.
Cette unité fournit, à partir d'un signal de référence qui est l'horloge, tous les signaux de synchronisation utiles au bon fonctionnement de l'ensemble.
Cette horloge est crée à partir d'un oscillateur interne qui utilise un signal en provenance d'un quartz externe.
Deux actions complémentaires à noter :
Cette unité assure :

- le contrôle de mise sous tension du microprocesseur (initialisation des registres).
- le traitement des interruptions.

L'interruption c'est une requête présentée à la logique de contrôle par des éléments extérieurs (périphériques).



Brochage du MC 6809

2. Architecture du 6809

Le microprocesseur 6809 est un processeur 8 bits dont l'organisation interne est orientée 16 bits. Il est fabriqué en technologie MOS canal N et se présente sous la forme d'un boîtier DIL 40 broches. Il est mono tension (5V).
Il existe deux versions différenciées par l'horloge.

- le 6809 est rythmé par une horloge interne (f=1 MHZ, 1.5 MHZ et 2 MHZ).
- le 6809E est rythmé par une horloge externe.

Ce dernier est adapté aux applications multiprocesseur. Il présente la particularité de pouvoir être synchronisé par une horloge extérieure.
Compatibilité complète entre les 2 versions.

2.1 Présentation du brochage

- L'alimentation (Vss - Vcc)
- Le bus des données 8 bits (Do à D7) Ces huit broches sont bidirectionnelles. Elles permettent la communication avec le bus des données interne du microprocesseur. Chaque broche peut "piloter" 1 charge TTL et 8 entrées de circuits appartenant à la famille 680 0. Bus en logique 3 états.
- Le bus des adresses 16 bits (A0 à A15) Ces broches unidirectionnelles transfèrent l'adresse 16 bits fournie par le microprocesseur au bus d'adresse du système. Mêmes caractéristiques électriques que pour le bus des données. Bus en logique 3 états. **Les adresses sont validées sur le front montant de Q.**
- Le bus de contrôle
- La broche **Read**/Write Cette broche indique le sens de transfert des données sur le bus des données. Ligne à logique 3 états
  - R/W = 1 lecture en cours (D0 - D7 sont des entrées)
  - R/W = 0 écriture en cours (D0 - D7 sont des sorties)

  Cette ligne est validée sur le front montant de Q.

Les lignes d'état du bus

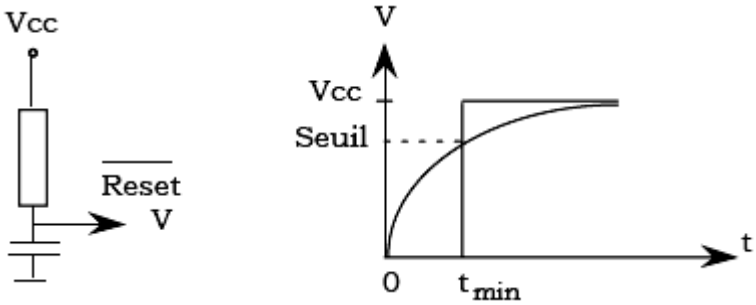
BA (Bus available) et **BS** (Bus state)
Information qui permet de connaître l'état du microprocesseur à tout moment.

BA	BS	Etat
0	0	normal
0	1	reconnaissance d'interruption
1	0	reconnaissance de synchronisation externe
1	1	arrêt bus disponible

1er cas :
Le microprocesseur est en fonctionnement normal, il gère les bus d'adresses et de données.
2ème cas :
Le microprocesseur est en phase de reconnaissance d'interruption pendant deux cycles. Cet état correspond à la recherche des vecteurs d'interruption : Reset, NMI, IRQ, SW1,2 et 3.
3ème cas :
Ce signal apparaît lorsque le microprocesseur rencontre l'instruction de synchronisation externe (niveau bas sur SYNC). Il attend alors cette synchronisation sur une des lignes d'interruption. Les bus sont en haute impédance pendant ce temps.
Dernier cas :
Correspond à l'arrêt du microprocesseur (niveau bas sur HALT).
Le microprocesseur laisse la gestion des bus des données et des adresses à un circuit annexe (contrôleur de DMA). Les bus sont en haute impédance.
La ligne BA au niveau haut indique que les bus sont en haute impédance.

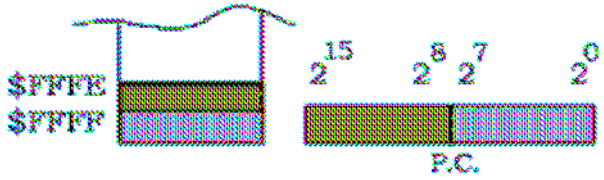
- Broche d'initialisation **RESET :** un niveau bas sur cette broche entraîne une réinitialisation complète du circuit.

Conséquences : l'instruction en cours est arrêté le registre de pagination (DP) est mis à zéro les interruptions IRQ et FIRQ sont masquées l'interruption non masquable NMI est désarmée
Pour être active cette ligne doit être maintenue à un niveau bas durant un temps suffisamment long (plusieurs cycles d'horloge).
Schéma adopté généralement



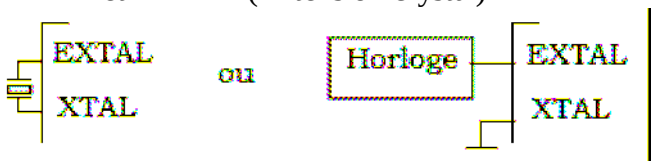
Le P.C. est initialisé avec le contenu des vecteurs d'initialisation qui se trouvent aux adresses $FFFE et $FFFF.

Ce contenu représente l'adresse du début du programme qui sera exécuté par le microprocesseur.



- La broche : **HALT** (Arrêt du microprocesseur). Un niveau bas sur cette broche provoque l'arrêt du microprocesseur (mais à la fin de l'exécution de l'instruction en cours). Il n'y a pas perte des données. (BA = BS = 1)
  Dans ce cas :
  - les demandes d'interruption IRQ et FIRQ sont inhibées
  - les demandes d'accès direct (DMA) à la mémoire sont autorisée.
  - les demandes d'interruptions RESET et NMI sont prises en compte mais leur traitement est différé.
- Les broches d'interruption
  - **NMI** (No Masquable Interrupt)
  - **IRQ** (Interrupt Request)
  - **FIRQ** (Fast Interrupt Request)

  Entrées (actives sur un niveau bas) qui peuvent interrompre le fonctionnement normal du microprocesseur sur front descendant de Q.
- Entrées d'horloge XTAL et EXTAL (Extension crystal)



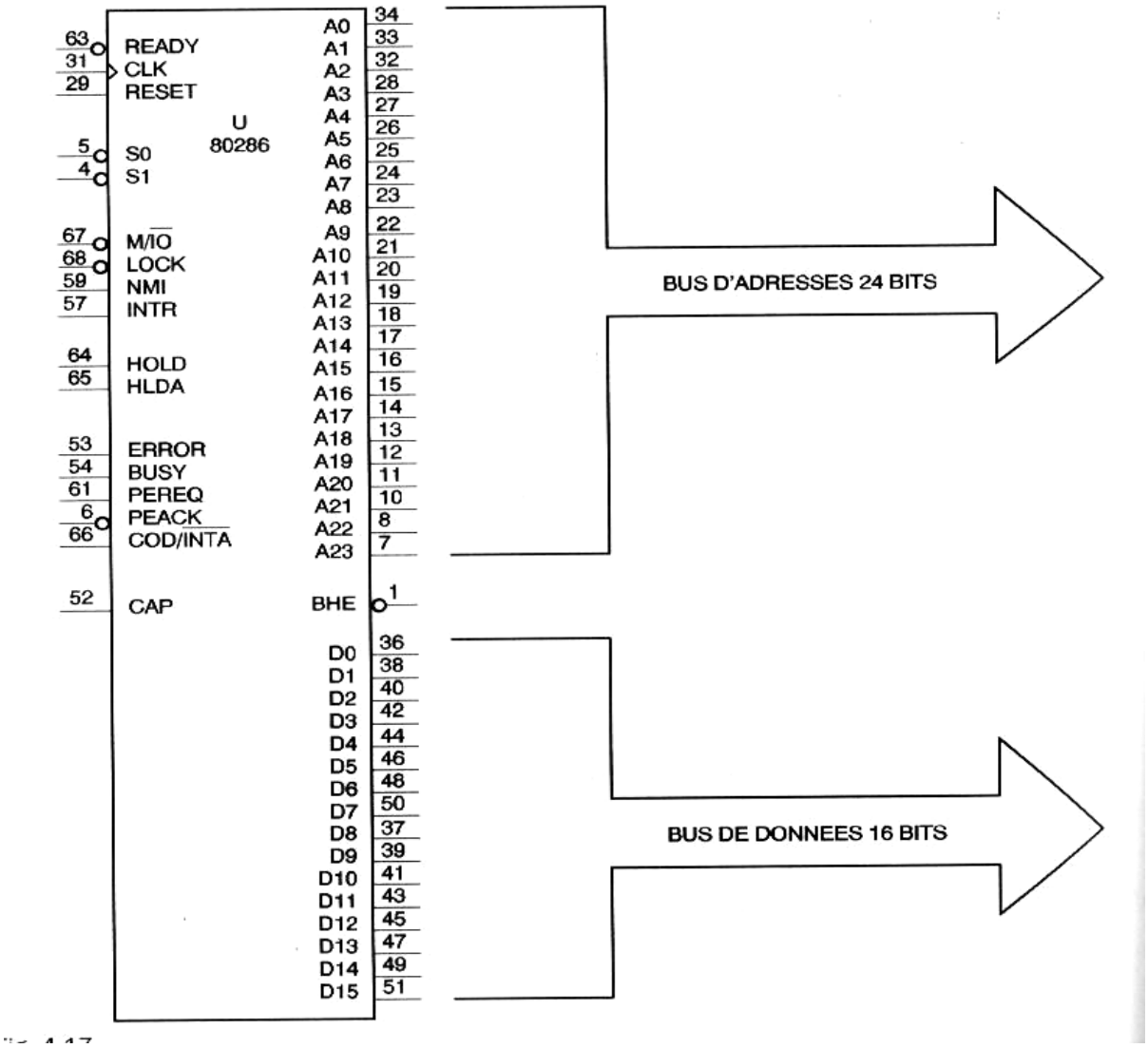
  La fréquence du quartz (horloge) est quatre fois la fréquence du microprocesseur.
  Eout représente le signal d'horloge commun au système. Il permet la synchronisation du microprocesseur avec la périphérie.
  Qout représente le signal d'horloge en quadrature avec Eout.

  Les données sont lues ou écrites sur le front descendant de Eout.
  Les adresses sont correctes à partir du front montant de Qout.

- Broches complémentaires du bus de contrôle :
  - **MRDY** (Memory ready) : cette broche de commande permet d'allonger la durée de Eout pour utiliser des mémoires à temps d'accès long. Active sur un niveau bas. L'allongement est un multiple de un quart de cycle et sa valeur maximale est de 10 cycles.
  - $\overline{DMA}/\overline{BREQ}$ (Direct Memory Acces/Bus Request) : cette broche permet de suspendre l'utilisation des bus du microprocesseur, pour faire de l'accès direct ou du rafraîchissement mémoire.
    Fonctionnement : pendant que Q est au niveau haut (si DMA/BREQ bas) cela entraîne l'arrêt du microprocesseur à la fin du cycle en cours ... et non de l'instruction.
    (BA = BS = 1 ce qui veut dire que tous les bus sont en haute impédance). Le circuit ayant généré cette commande dispose de 15 cycles machines avant que le microprocesseur ne reprenne le contrôle des bus.

B. EXEMPLE : MICROPROCESSEUR 80286

Le brochage du circuit intégré est représenté sur la figure 1. La largeur du bus de données est de 16 bits. Le bus d'adresse, constitué de 24 bits, peut géré un espace physique de 16 Mo.



Le processeur 80826 peut fonctionner selon deux modes.

- En mode réel qui présente un système d'adressage compatible avec celui du 8086 mais un jeu d'instructions étendu et sensiblement plus complet.
- En mode protégé qui intègre des mécanismes complexes comme le contrôle de débordement de zone, la gestion des taches et commutation de contexte, la protection des données à plusieurs niveaux de privilège d'exécution et fonction de leurs niveau de privilèges propre.

Contrairement à un microcontrôleur, un microprocesseur ne peut fonctionner seul. Quelle que soit l'application il est donc accompagné d'un certain nombre de circuits périphériques. Dans le cas de 80286 ces circuits périphériques sont :

- Un générateur d'horloge 82284. Les trois fonctions essentielles de ce circuit sont la synchronisation du signal RESET, la génération de l'horloge de base et la synchronisation du signal READY permettant l'accès à un circuit mémoire ou interface d'entrée / sortie.
- Un coprocesseur mathématique optionnel 80287. Ce processeur spécialisé dans les traitements numériques peut généralement accélérer les calculs portant sur des nombres réels flottants ou sur des nombres entiers en format étendu dans des rapports voisin de 100, en général fonction de l'opération.
  L'intérêt majeur de ce coprocesseur réside dans sa facilité de connexion au processeur maître car son bus est directement branché en parallèle sur celui de 80286.
  Le 80287 reconnaît automatiquement les opérations pour lesquels il est concerné.
  La présence ou l'absence de ce coprocesseur reste transparente au programmeur jusqu'au moment de l'édition des liens. La plus part des compilateurs des langages évolués comportent une option avec ou sans coprocesseur.

Les circuits annexes sont :

- Un contrôleur de RAM dynamique 8027
- Des tampons bidirectionnels 8286-8287
- Des bascules 8283-8282
- Un contrôleur d'interruption 8259A

1. Architecture système

Une architecture autour de 80286 comporte deux types de bus comme le montre le synoptique de la figure 2 :

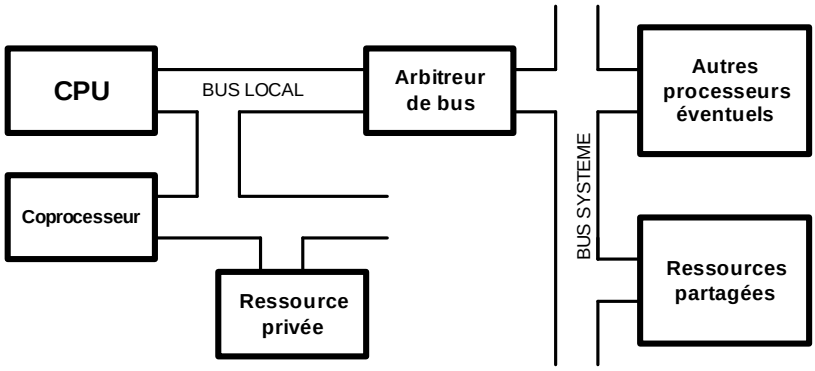


Figure 2 Architecture système multiprocesseur autour d'un 80286

- Un bus local composé des signaux directement produits par le processeur lui-même (adresses, données et signaux de contrôles).
- Un bus système qui relie ou connecte le processeur à des ressources publiques de mémoires ou d'entrée / sortie.

Cette architecture est utilisée dans les systèmes Intel Multibus I et Multibus II.
Une architecture plus simple est représentée sur la figure 3 :

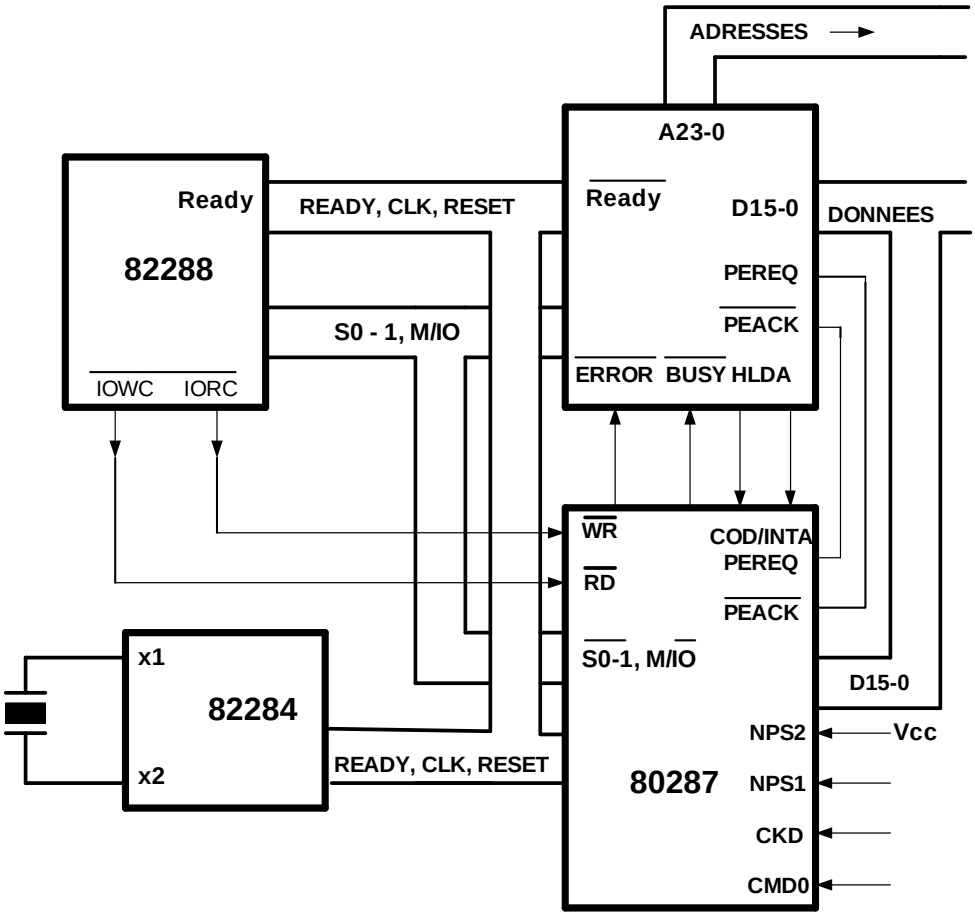


Fig.3 Architecture de 80286 avec ses circuits périphériques

2. Structure interne du processeur 80286

Dérivée du 8086, l'unité centrale du 80286a été conçu comme un microprocesseur 16 bits autorisant une utilisation dans un environnement multitâches – multi-utilisateurs.
Cette caractéristique n'été pas employée par Microsoft dans son système d'exploitation DOS.
Le boîtier intègre de façon câblée un certain nombre de mécanismes de base permettant l'implantation simple d'un système d'exploitation performant.
Les principaux mécanismes implantés traitent le partage de la mémoire, la protection de celle-ci par niveaux de privilège répartis en anneau,le contrôle de non débordement de zone lors d'un accès mémoire par une tache spécifique.
Les quatre niveaux de privilège sont schématisés par la figure 4. Le schéma de la figure 5 représente la structure interne du 80286 constitué de quatre sous-ensembles élémentaires, détaillés ci-après.

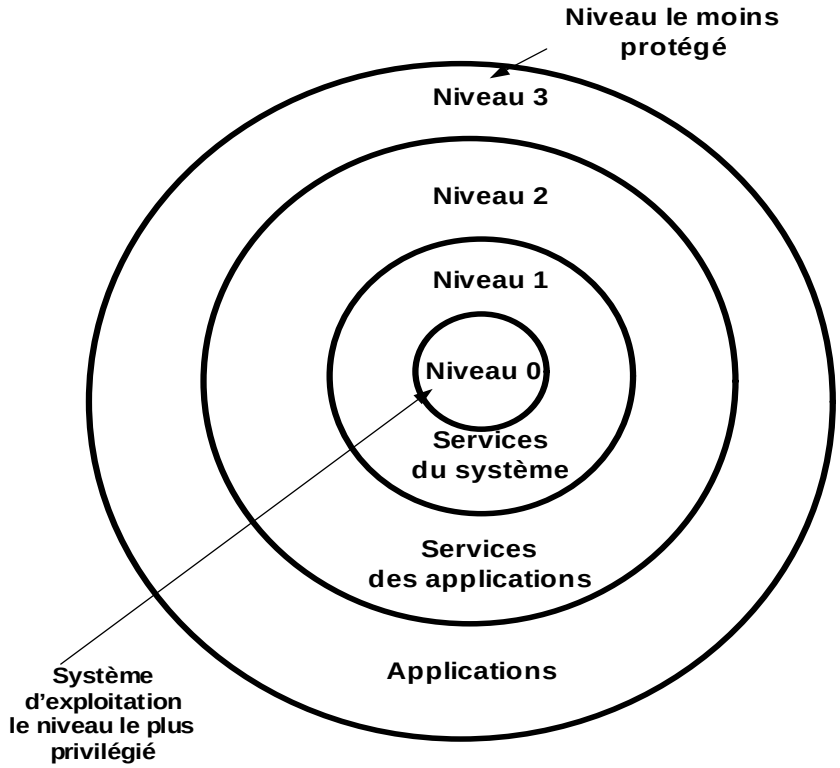


Fig.4 Les quatre niveaux de privilège du 80286

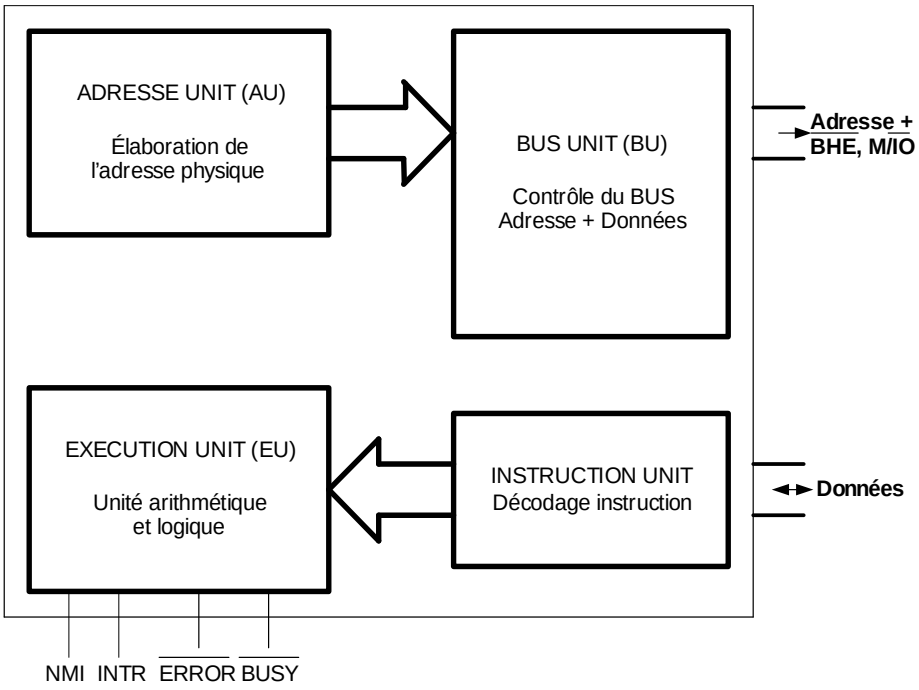


Fig. 5 Structure interne de 80286

2.1 Le bus unit

Le bus unit réalise toutes les opérations portant sur le bus d'entrée / sortie, mémoire ou périphérique : génération des adresses des données, des signaux de commande nécessaires aux accès mémoire.
Le bus unit établit un régime pipeline en accès au bus par la lecture anticipé des instructions qui sont ensuite mémorisées à l'intérieur de l'instruction unit.
Cette lecture anticipé permet de stocker jusqu'à six octets d'instruction : ce processus permet une plus grande vitesse de traitement.

2.2 L'instruction unit

L'instruction unit reçoit les instructions de la file de six octets du bus unit, décode ces instructions et les stocke, décodées, dans une file d'attente pouvant contenir jusqu'à trios instructions complètes.

2.3 L'exécution unit

Les instructions décodées sont alors récupérées par l'exécution unit qui les exécute et peut éventuellement solliciter le bus unit pour réaliser des accès en lecture ou en écriture sur le bus de données.

2.4 L'adresse unit

La fonction de cette unité est la gestion de la mémoire et la génération des adresses physiques utilisées par le bus unit.
La génération des adresses nécessite la présence d'une unité arithmétique de 24 bits pour prétendre à l'accès effectif sur une zone de 16 Mo.
La gestion de la mémoire comporte également les contrôles de protection.

3. Segmentation de la mémoire

3.1 Mode réel

Le microprocesseur /80286 peut adresser 1 Mo de mémoire RAM en mode réel et 16 Mo en mode protégé.
Dans les deux cas, la mémoire globale est tronçonnée en segments de 64 Ko.
Un segment est donc définit par son adresse physique absolue de départ dans la mémoire et sa longueur maximale de 64 Ko.
En règle générale, un segment devra commencer dans la mémoire à une adresse divisible par 16.
La figure 6 montre que les segments peuvent être consécutifs, disjoints, se chevaucher ou se superposer.
A un instant donné, l'adresse d'une information est définie à l'aide du contenu de deux registres CS et IP.
Le registre CS est le registre sélecteur de segment et le registre IP donne la valeur de déplacement dans ce segment. Ce déplacement est aussi appelé offset.
L'adresse complète est notée CS :IP.
Les deux registre CS et IP sont des registres 16 bits. L'adresse réelle sur 20 bits peut se calculer par la relation 16.CS + IP qui donne un résultat sur 20 bits.

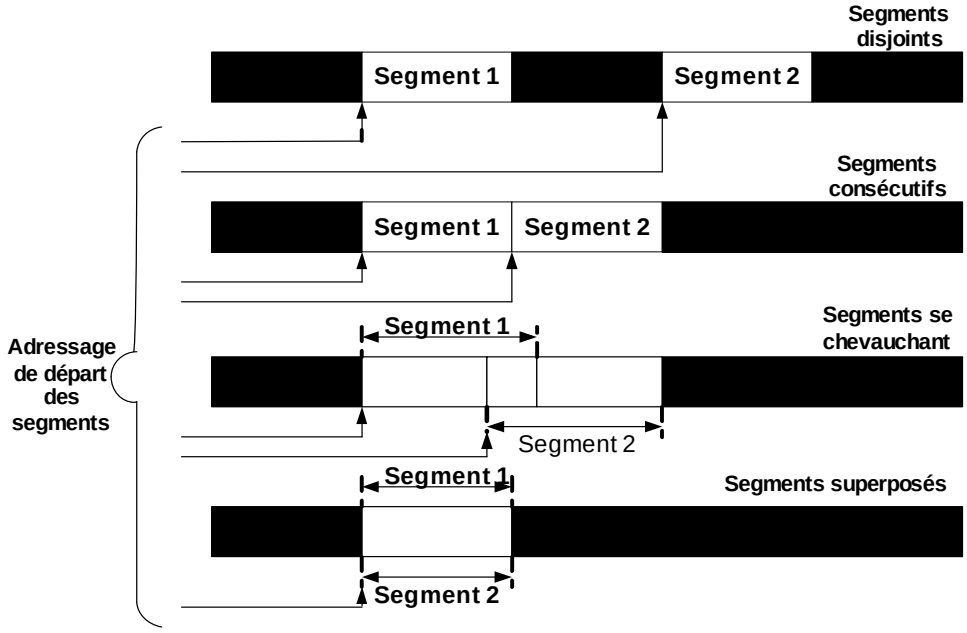


Fig. 6 Répartition des segments dans la mémoire

3.2 Mode protégé

Dans cette configuration, le 80286, utilise à plein les 24 bits de son espace d'adressage, ce qui autorise l'accès à 16 Mo de mémoire. Une mémoire cache, interne à l'Adresse Unit, permet alors d'effectuer la traduction d'adresse, d'évaluer les niveaux de protection et d'élaborer éventuellement cette adresse physique de la mémoire cible.
On parle ici de mémoire virtuelle car l'espace adressable est beaucoup plus large que l'espace physique. On suppose donc que des parties de la mémoire sont supportées donc des mémoires auxiliaires et chargées lors de leur utilisation effective.
En mode protégé comme en mode réel l'adresse peut être exprimée par CS:IP.

4. Les registres de 80286

Les registres équipant le processeur 80286 sont répartis en trois groupes :

- les registres généraux
- les registres de segmentation
- les registres d'état et de contrôle.

4.1 Registres généraux

Il s'agit de 8 registres 16 bits. Le 80286 n'a pas de registre accumulateur. Chaque registre peut être employé pour servir d'opérande ou de résultat dans une opération arithmétique ou logique. Certains de ces registres sont parfois dédiés à des opérations particulières.
- AL, AX : sont obligatoirement utilisés comme support d'opérande dans les instructions d'entrées / sorties ainsi que dans les multiplications et divisions.

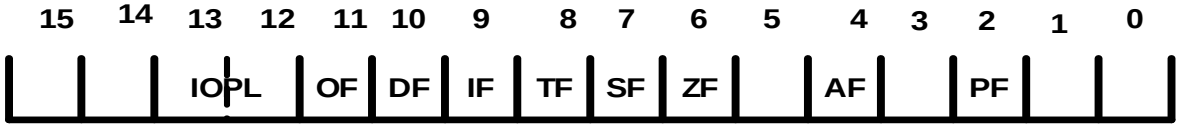
- CX : compteur de boucle dans les instructions de comptage (Loop) ou dans les instructions de traitement de chaine ainsi que dans les décalages de plusieurs pas.
- DX : support d'adresse dans les instructions d'entrées / sorties ou poids forts d'une quantité 32 bits pour la multiplication ou la division.
- SP : pointeur de pile.
- BX, BP : sont très fréquemment utilisés dans les modes d'adressage indirect et sont de ce fait appelés registres de base. BP et plus spécifiquement dédié à la pile et trouve son utilisation principale dans la récupération des paramètres envoyés à une procédure par un programme appelant.
- SI, DI : s'ile peuvent être utilisés pour l'adressage indirect, ils sont également dédiés à la fonction d'index dans l'adressage indirect indexé ou dans les instructions de traitement de chaines. Ils sont de fait appelés registres d'index.

4.2 Registres de segmentation

Le 80286 est muni de quatre registres de segments distincts (DS, ES, SS et CS) permettant donc d'accéder immédiatement à des objets situés dans quatre segments logiques différents, situés à une implantation quelconque en mémoire.
Le code segment (CS) pointe le segment contenant le programme en cours d'exécution.
Le stack segment (SS) pointe le segment dans lequel se trouve utilisée la zone de pile courante, indexée par le Stack Pointer (SP) ou le Base Pointer BP.
Le data segment (DS) référence le segment des données utilisées par le programme en cours.
L'extra segment (ES) est analogue au registre DS ; il permet éventuellement d'accéder à un deuxième segment de données.

4.3 Registres d'état et de contrôle

Le 80286 contient deux registres d'état et de contrôle. Le premier registre contient les 11 flags – indicateurs – dont la signification est donnée à la figure 7.



0 : Retenue (Carry)
2 : Parité (Paruty)
4 : Retenue auxiliaire
6 : Zéro
7 : Signe
8 : Pas à pas (trap flag)
9 : Autorisation d'interruptions masquables
10 : Direction d'incrémentation
11 : Dépassement
13 : Niveau de privilège d'E/S (I/O Privilege Level)
14 : Emboitement des taches (Nested Task Flag)

8, 9, 10 : registres de contrôle
13, 14 : registres spécifiques au mode protégé.

5. Quelques instructions assembleur de la famille 8086

La liste des instructions suivantes est destinée a donner une idée de langage assembleur de la famille 8086.

5.1 Instructions arithmétiques

La destination est la source, A et B peuvent etre un registre ou une mémoire, la source peut etre une donnée imédiate.

- MOV : destination, source
- ADD : destination, source
- ADD B, A : le resultat A + B est mis dans B, et A est inchangé.
- SUB B, A : le résultat B – A est mis dans B, et A est inchangé.
- AND B, A : le résultat est mis dans B, et A est inchangé.
- CMP B, A : compare A et B et positione les flags.
- INC RM : RM est remplacé par RM + 1.
- DEC RM : RM est remplacé par RM – 1.
- NOT RM : donne le complément de RM.
- NEG RM : donne le complément à 2 de RM.

5.2 Instructions sur la pilr

- PUSH RM : place RM dans la pile.
- POP RM : extrait RM de la pile.

5.3 Instructions de contrôle

- JMP : branchement inconditionnel
- J** : branchement conditionnel (égalité, plus grand, plus petit, etc..)
- CALL : appel à un sous – programme.
- RET : retour d'un sous – programme
- IRET : retour d'interruption.
- STI : validation des interruptions.
- CLI : invalidation des interruprtion

Chapitre 3

MICROCONTROLEUR.

I. CONSIDERATIONS GENERALES

Un système minimal, pour fonctionner, a besoin :

- d'une Unité Centrale.
- de Mémoire *morte*, pour le programme (PROM, EPROM, ...).
- de Mémoire *vive*, pour les calculs, pour stocker les données.
- de circuits Interfaces, pour connecter les périphériques qui vont permettre la communication avec l'extérieur.

D' où l'apparition des Microcontrôleurs (ou Monochip) :
Dans un seul circuit, on va trouver :

- Une Horloge (oscillateur).
- Un Processeur (Unité Centrale).
- De la Mémoire Vive (RAM).
- De la Mémoire Morte (PROM, EPROM ou EEPROM).
- Des Interfaces, qui vont dépendre, en général, du type de microcontrôleur choisi :
  - Compteurs / timer.
  - Convertisseur Analogique/Numérique (C.A.N.)
  - chien de Garde («Watch Dog").
  - Gestion d'un "port" parallèle.
  - Gestion d'une liaison série RS232.
  - Gestion d'interruptions..
  - Gestion de moteurs en PWM.
  - Gestion d'écran LCD.
  - Gestion de Bus I2C.
  - etc ...

Il suffit de choisir le microcontrôleur le mieux adapté à l'application que l'on doit réaliser !

1. Architecture des microcontrôleurs

Les microcontrôleurs ont été conçus sur une architecture dite HARVARD (RISC) et non sur un modèle VON NEUMANN (COMPLEX).

- L'architecture VON NEUMANN (figure 1) employée par la plupart des microcontrôleurs actuels (INTEL80XX, Motorola HC05, HC08 et HC11, ou ZILOG Z80) est basée sur un bus de données unique. Celui-ci véhicule les instructions et les données.

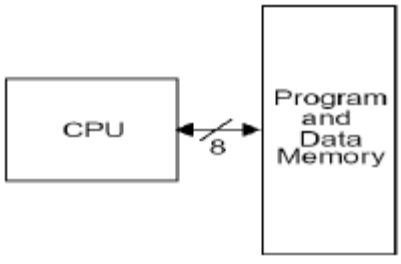


Fig. 1 Architecture Von Neumann

- L'architecture HARVARD (figure 2) utilisée par les microcontrôleurs PICS est basée sur deux bus de données. Un bus est utilisé pour les données et un autre pour les instructions.

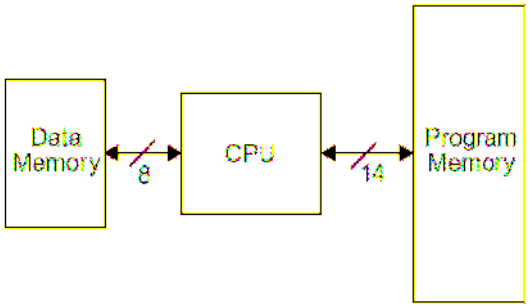


Fig.2 Architecture Harvard

Avantages et inconvénients

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

Remarque : La taille mémoire spécifiée pour un PICs s'exprime en Kilo Mots (14 bits pour la famille 16F87X) et non en kilo octets. Comme chaque instruction est codée par un mot de 14 bits, comparées aux microcontrôleurs classiques (1,2 ou 3 octets par instruction), les PICs ont un code plus compact et ils utilisent moins de mémoire.

1.2 Le PIC

Un PIC n'est rien d'autre qu'un microcontrôleur, c'est à dire une unité de traitement de l'information de type microprocesseur à laquelle on a ajouté des périphériques internes permettant de réaliser des montages sans nécessiter l'ajout de composants externes.
La dénomination PIC est sous copyright de Microchip, donc les autres fabricants ont été dans l'impossibilité d'utiliser ce terme pour leurs propres microcontrôleurs.
Les PICs sont des composants dits RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. Il faut retenir que plus on réduit le nombre d'instructions, plus facile et plus rapide en est le décodage, et plus vite le composant fonctionne.

Sur le marché on trouve 2 familles opposées, les RISC et les CISC (Complex Instructions Construction Set). Chez les CISC, on diminue la vitesse de traitement, mais les instructions sont plus complexes, plus puissantes, et donc plus nombreuses. Il s'agit donc d'un choix de stratégie.
Tous les PICs Mid-Range ont un jeu de 35 instructions, stockent chaque instruction dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en 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.

L'horloge fournie au PIC est prédivisée par 4 au niveau de celle-ci. C'est cette base de temps qui donne le temps d'un cycle.
Si on utilise par exemple un quartz de 4 MHz, on obtient donc 1000000 de cycles/seconde, or, comme le PIC exécute pratiquement 1 instruction par cycle, hormis les sauts, cela vous donne une puissance de l'ordre de 1MIPS (1 Million d'Instructions Par Seconde). Les pics peuvent monter à 20 MHz.

1.2.1 Les différentes familles des PICs

La famille des PICs est subdivisée à l'heure actuelle en 3 grandes familles :

- La famille Base-Line, qui utilise des mots d'instructions (nous verrons ce que c'est) de 12 bits,
- La famille Mid-Range, qui utilise des mots de 14 bits (et dont font partie les 16F84 et 16F876),
- La famille High-End, qui utilise des mots de 16 bits.

1.2.2 Identification d'un PIC

Pour identifier un PIC, vous utiliserez simplement son numéro.
Les 2 premiers chiffres indiquent la catégorie du PIC, 16 indique un PIC Mid-Range.
Vient ensuite parfois **une lettre** L : Celle-ci indique que le PIC peut fonctionner avec une plage de tension beaucoup plus tolérante.
Ensuite, vous trouvez :

- C indique que la mémoire programme est une EPROM ou plus rarement une EEPROM
- CR pour indiquer une mémoire de type ROM
- F pour indiquer une mémoire de type FLASH.

Il faut noter que à ce niveau que seule une mémoire FLASH ou EEPROM est susceptible d'être effacée, donc n'espérez pas reprogrammer vos PICs de type CR. Pour les versions C, voyez le datasheet. Le 16C84 peut être reprogrammé, il s'agit d'une mémoire EEPROM. Le 12C508, par exemple, possède une mémoire programme EPROM, donc effaçable uniquement par exposition aux ultraviolets. Donc, l'effacement nécessite une fenêtre transparente sur le chip, qui est une version spéciale développement, et non la version couramment rencontrée.
Un composant qu'on ne peut reprogrammer est appelé O.T.P (One Time Programming : composant à programmation unique).
Les derniers chiffres identifient précisément le PIC. (84)
Finalement **sur les boîtiers est donné le suffixe** « -XX » dans lequel XX représente la fréquence d'horloge maximale que le PIC peut recevoir. Par exemple –04 pour un 4MHz.

Exemple :
Un 16F84-04 est un PIC Mid-Range (16) donc la mémoire programme est de type FLASH (F) donc réinscriptible de type 84 et capable d'accepter une fréquence d'horloge de 4MHz.

Une dernière indication que vous trouverez est le type de boîtier.
Exemple : le boîtier PDIP, est un boîtier DIL 18 broches, avec un écartement entre les rangées de 0.3'' (étroit). La version 4 MHz sera amplement suffisante.

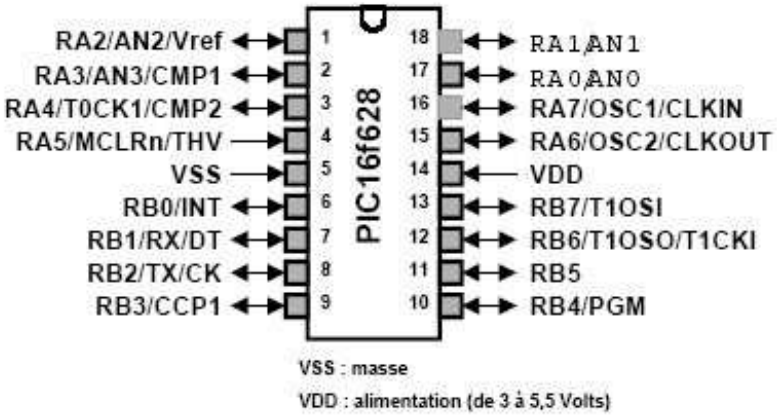
Notez dès à présent que les PICs sont des composants STATIQUES, c'est à dire que la fréquence d'horloge peut être abaissée jusqu'à l'arrêt complet sans perte de données et sans dysfonctionnement. Une version –10 peut donc toujours être employée sans problème en lieu et place d'une –04. Pas l'inverse, naturellement.
Ceci par opposition aux composants DYNAMIQUES (comme les microprocesseurs de votre ordinateur), donc la fréquence d'horloge doit rester dans des limites précises. N'essayez donc pas de faire tourner votre PIII/500 à 166MHz, car c'est un composant dynamique.

2. Exemple : PIC16F628

Description du PIC16F628

Ce composant intègre un microcontrôleur 8 bits, c'est-à-dire un processeur et des périphériques, dans un boîtier « Dual in line » de 18 broches. Il est réalisé en technologie CMOS et peut cadencé par une horloge allant de 0 à 20 MHz ; il doit être alimenté par une tension allant de 3 à 5,5 V.

Les broches du composant possèdent plusieurs affectations entre les portes d'E/S, les périphériques et les fonctions système.
Brochage du PIC16F628



En périphérie de l'unité centrale, on peut recenser les ressources suivantes sur le composant :

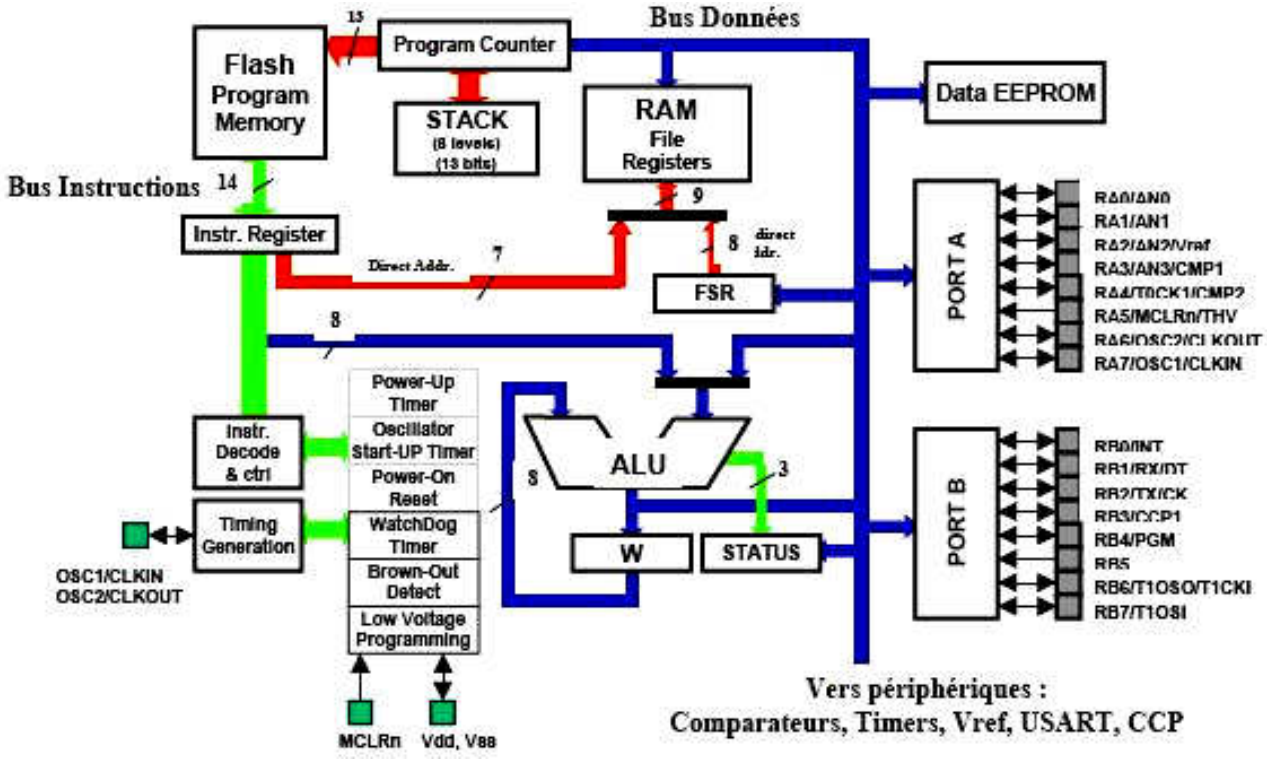
- Mémoire Flash Programme : 2048 instructions
- Mémoire RAM Données : 224 Octets
- Mémoire EEPROM Données : 128 Octets
- Ports d'E/S : 2 ports 8 bits
- Périphériques :
  - 3 Timers (8 et 16 bits)
  - 1 module Capture/compare/PWM
  - 2 comparateurs analogiques

1 référence de tension

1 USART (émission/réception série synchrone et asynchrone)

Architecture interne

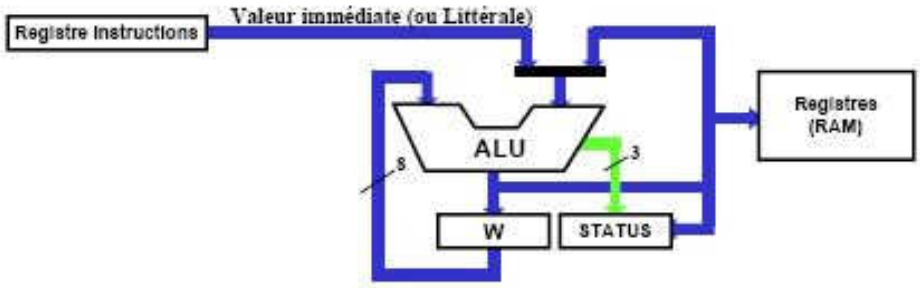
Le microcontrôleur est composé d'une unité centrale et de périphériques ; le fonctionnement est géré par un séquenceur qui, en fonction des modes opératoires, fournit les signaux de contrôle à chaque module.
Le fonctionnement de l'unité centrale est de type RISC (Reduced Instruction Set Computer), le jeu d'instructions est réduit à 35.



Unité centrale

Le cœur du processeur est composé de 3 entités :

- L'ALU (opérations arithmétiques et logiques sur 8 bits) et son registre d'état (STATUS)
- Le registre du travail : W (Warking Register)
- Les registres d'usage général (File Registres)



Les opérations possibles sont alors les suivantes (avec un exemple) :

- W ← W (op) valeur immédiate ADDLW 0xFF
- Dest. ← WQ (op) Registre ADDWF REGISTRE, dest.
- Registre ← W MOVWF REGISTRE
- Dest. ← Registre MOVF REGISTRE,dest.
- Dest. ← Registre ± 1 INCF REGISTRE,dest.
- Dest. ← Registre ± 1 plus test INCFSZ REGISTRE,dest.
- Manipulation et test individuel de bits sur les registres

Ou :
« (op) » représente une opération arithmétique ou logique
« Dest. » représente la destination des données (W ou le REGISTRE concerné)

II Les microcontrôleurs 16F87X

1. Caractéristiques des microcontrôleurs 16F87X

Ces microcontrôleurs appartiennent à la famille des PICs. Le 16 signifie qu'ils font partie de la famille des 16F de MICROCHIP et le PIC16F876 est une version 28 broches alors que le 16F877 est une version 40 broches.

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

Caractéristiques communes :

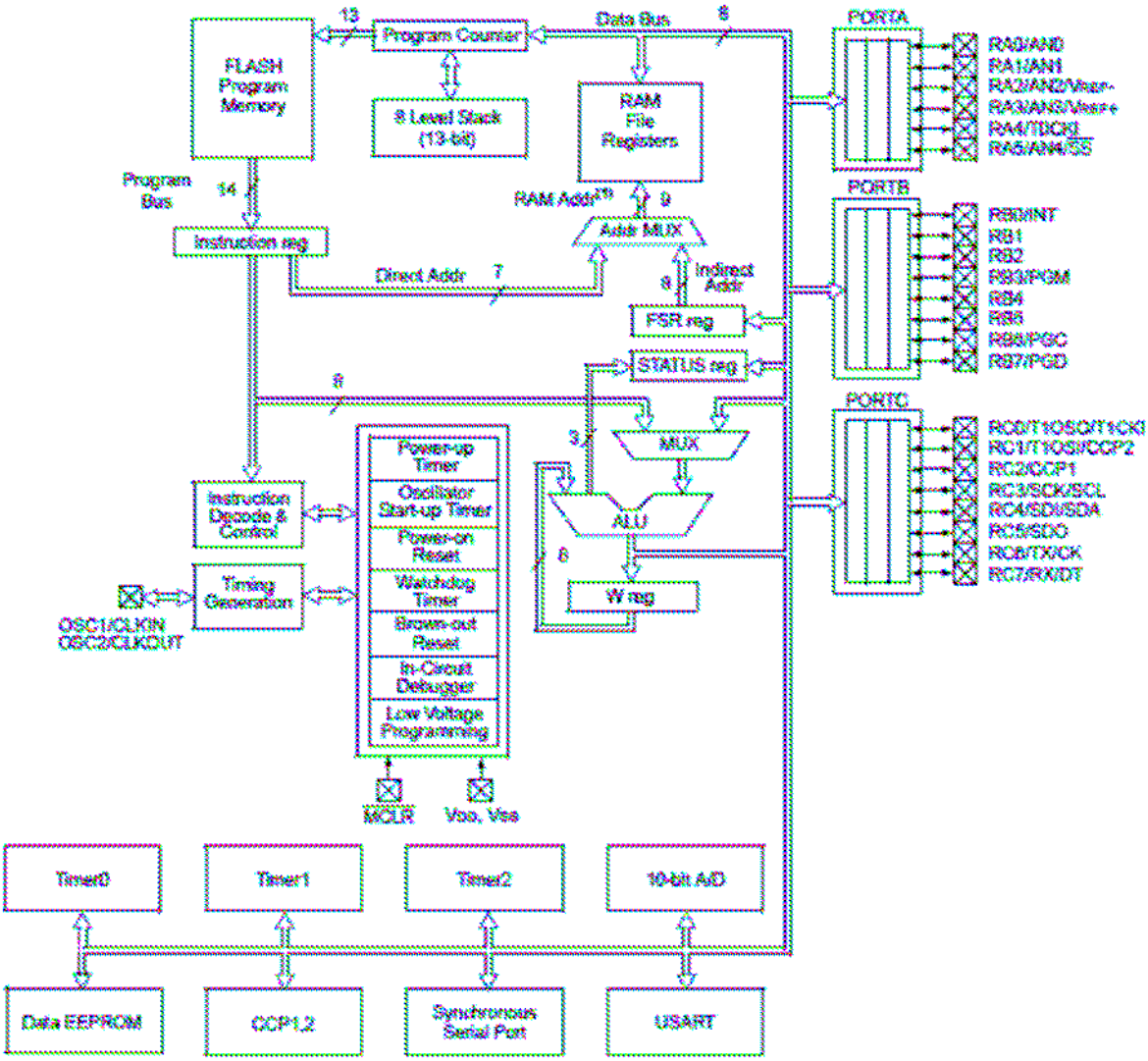
- Programmable sur site ICSP (In Circuit Serial Programming).
- Débuggable sur site ICD.
- Fréquence de fonctionnement élevée, jusqu'à 20 Mhz.
- Une mémoire vive de 192 à 368 octets.
- Une mémoire EEPROM pour sauver des paramètres de 128 à 256 octets.
- Une mémoire morte de type FLASH de 4 Kmots à 8 Kmots (1mot = 14 bits).
- Chien de garde WDT.
- Surveillance d'horloge OST.
- Surveillance de tension d'alimentation BOR.
- De 21 à 32 Entrées et sorties suivant le type de micro contrôleur.
- Chaque sortie peut sortir un courant maximum de 25mA.
- 3 Temporisateurs : TIMER0 (8 bits avec pré diviseur), TIMER1 (16 bits avec pré diviseur avec possibilité d'utiliser une horloge externe réseau RC ou QUARTZ) et TIMER2 (8 bits avec pré diviseur et post diviseur)
- 2 entrées de captures et de comparaison avec PWM (Modulation de largeur d'impulsions).
- Convertisseur analogique numérique 10 bits avec de 5 à 8 entrées multiplexées maximum.
- Une interface de communication série asynchrone et synchrone. (USART/SCI).
- Une interface de communication série synchrone. (SSP/SPI et I2C).
- Plusieurs modes de fonctionnements faible consommation.
- Une seule tension d'alimentation 2 ou 5V.
- Conservation des informations en mémoire vive jusqu'à 1.5V.
- Faible consommation :
    - <2mA à 5V pour Fquartz=4Mhz.

- 20µA à 3V pour Fquartz à 32Khz.
- <1µA pour en mode sommeil.

2. Organisation interne

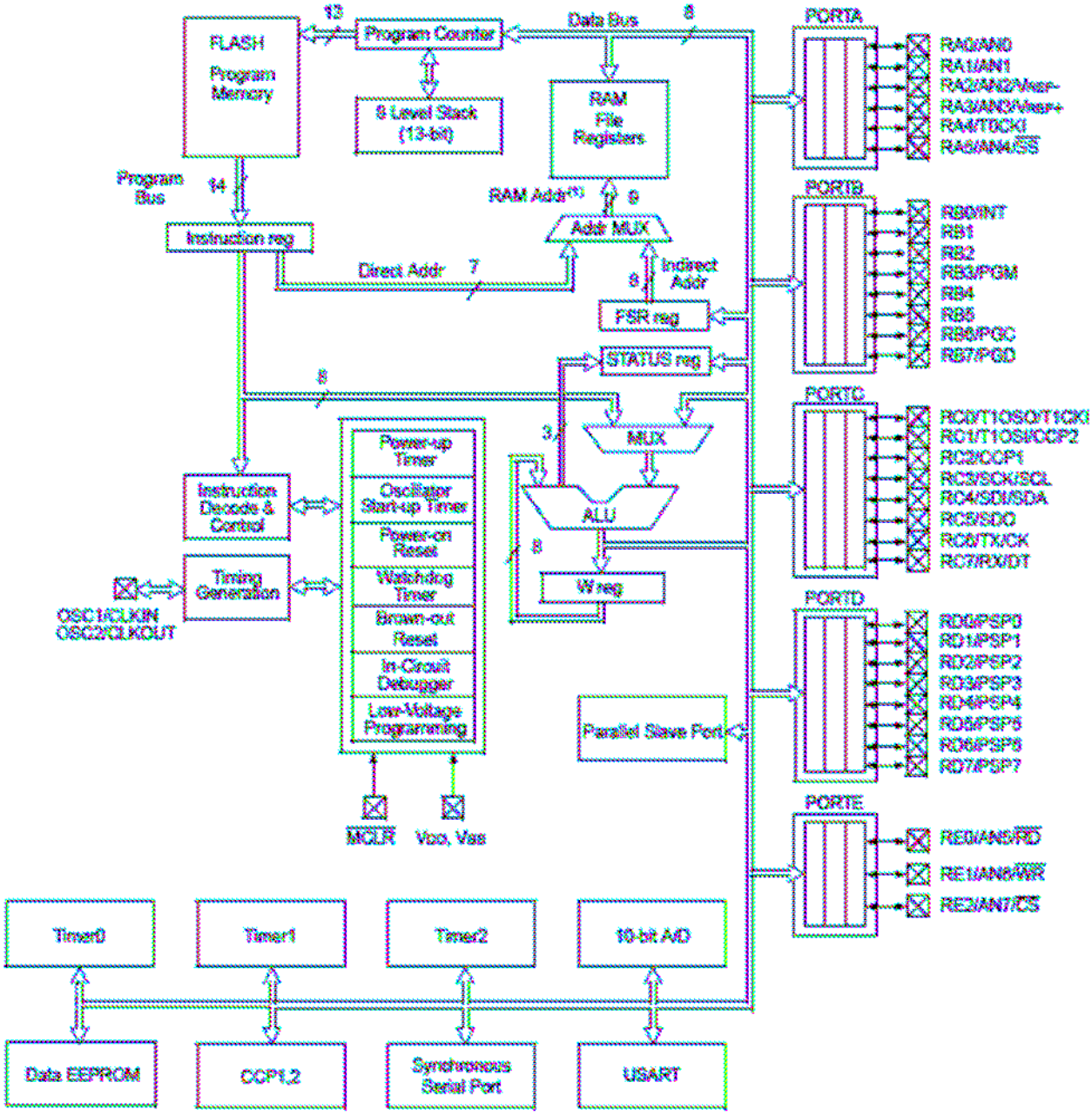
2.1) Les µCs 16F873 et 16F876.

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes



2.2) Les µCs 16F874 et 16F877.

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Remarque : Les 16F873 et 16F874 présentent peu d'intérêt par rapport aux 16F876 et 16F877, en effet ils possèdent moins de mémoires programmes et ils sont à peu près au même prix.

3. Description des différentes broches

3.1) Les µCs 16F873 et 16F876.

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	1	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/SS/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	25	25	I/O	TTL	Interrupt-on-change pin.
RB5	26	26	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I²C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I²C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
Vss	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

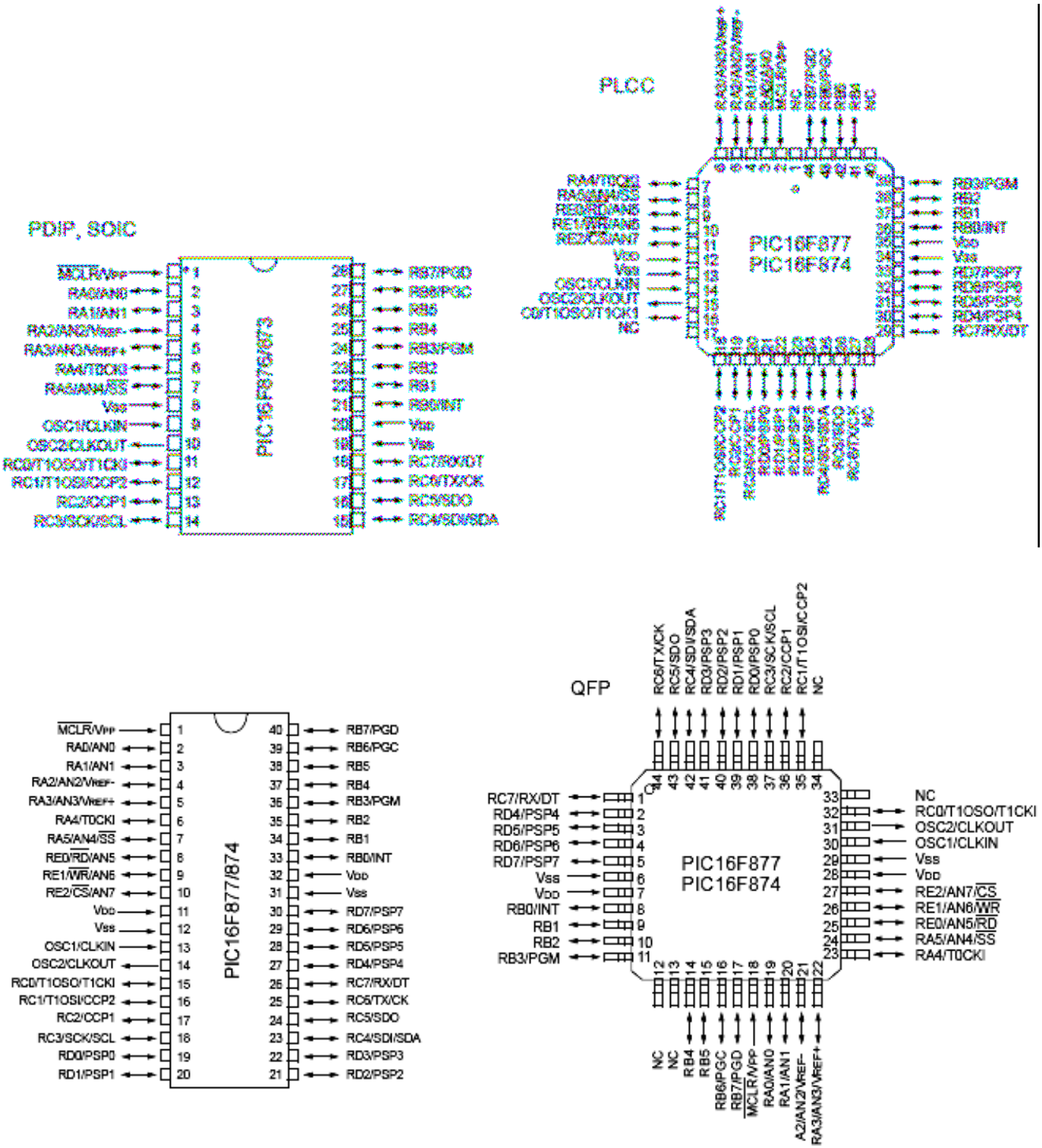
Legend: I = input, O = output, I /O = input/output,
P = power , — = Not used TTL = TTL input
ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

3.2) Les µCs 16F874 et 16F877.

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/ counter. Output is open drain type.
RA5/SS/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/ output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/RD/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/WR/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/CS/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
Vss	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

4. Brochages physiques des différentes versions de microcontrôleurs 16F87X.



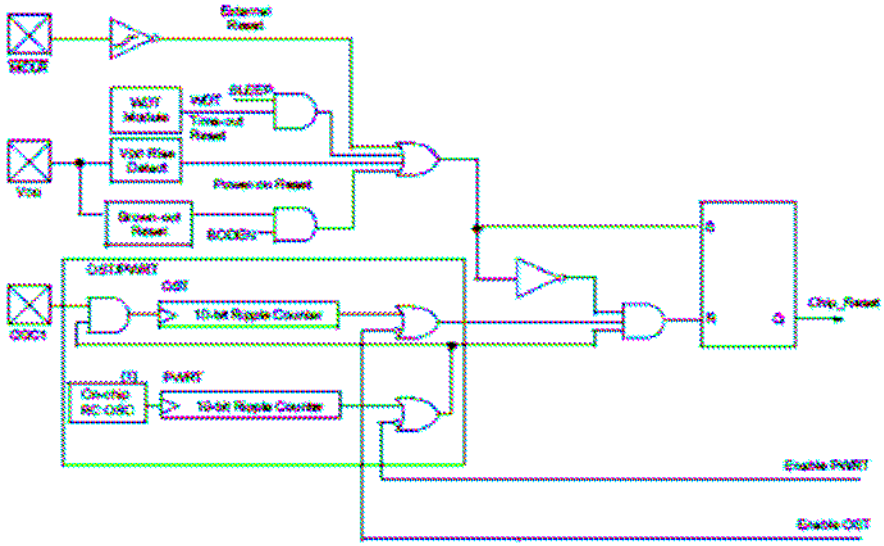
Les broches du microcontrôleur (µc).

4.1) MCLR.

Cette broche sert à initialiser le µC.
Le µC dispose de plusieurs sources de RESET :
• POR.
• EXTERNAL RESET.
• WDT.
• BOR.

- POR: (POWER ON RESET) Mise sous tension.
  Un front montant sur MCLR déclenche l'initialisation du µC. Le temps nécessaire est au minimum de 72mS et au maximum de 72mS+1024*Tosc. Le µC dispose en interne d'un circuit de détection de niveau, quand la tension VDD est comprise entre 1.2V et 1.7V, il démarre une procédure d'initialisation.
  Cette broche peut être simplement reliée à VDD si on n'a pas besoin de RESET externe. Par contre si on souhaite implanter un bouton de remise à zéro, on pourra câbler un simple réseau RC sur la broche MCLR.
  Remarque importante : On peut se passer de circuit RC à la seule condition que le temps de monté de VDD soit suffisamment rapide (au minimum 50mV/mS). Si le temps de montée est inférieur à 50mV/ms, il faut rajouter un réseau RC.
- EXTERNAL RESET (Mise à l'état bas de MCLR). Remise à zéro extérieure. Il faut appliquer un niveau bas sur l'entrée RESET pendant au moins 2µS pour que l'Initialisation soit prise en compte.
- WDT: Chien de garde.
  Si le WDT arrive à la fin du temps de garde sans avoir été rafraîchi il y aura alors une initialisation du µC.
- BOR: Baisse de l'alimentation.
  Si la tension VDD chute en dessous de 4V pendant 100µS au moins, le microcontrôleur peut générer un RESET.

Schéma structurel du câblage de la broche MCLR.



4.2) Oscillateur : OSC1 et OSC2 ou CLKIN et CLOUT.

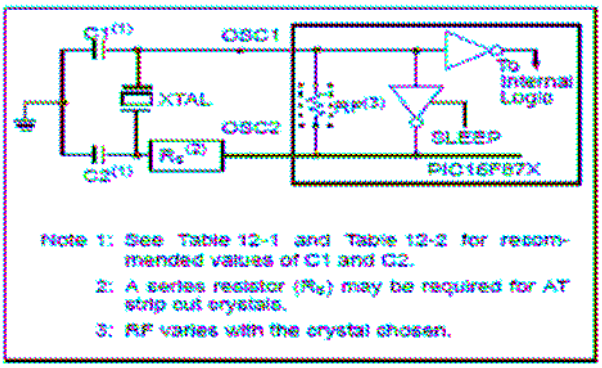
Ces broches permettent de faire fonctionner l'oscillateur interne du PIC.

On peut utiliser 3 types d'oscillateurs :

- Un quartz ou résonateur céramique.

TABLE 12-1: CERAMIC RESONATORS

	Ranges Tested:		
Mode	Freq.	OSC1	OSC2
XT	455 kHz 2.0 MHz 4.0 MHz	68 - 100 pF 15 - 68 pF 15 - 68 pF	68 - 100 pF 15 - 68 pF 15 - 68 pF
HS	8.0 MHz 16.0 MHz	10 - 68 pF 10 - 22 pF	10 - 68 pF 10 - 22 pF

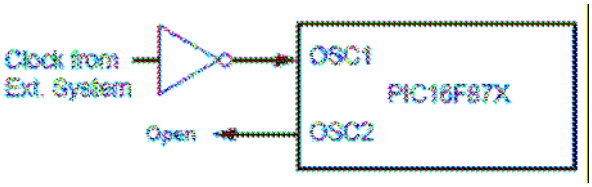


Note 1: See Table 12-1 and Table 12-2 for recommended values of C1 and C2.
2: A series resistor (R_S) may be required for AT strip cut crystals.
3: RF varies with the crystal chosen.

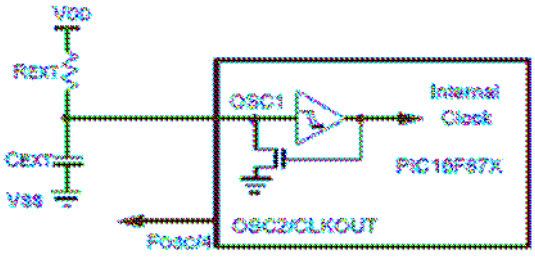
TABLE 12-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

- Un oscillateur externe



- Un réseau RC



Remarque : Les instructions standards durent 1 cycle machine (sauf les instructions de sauts 2 cycles). Le µC utilise 4 coups d'horloge pour réaliser un cycle machine.
Si la fréquence du QUARTZ est de 20 MHz (T=50 nS), une instruction sera exécutée toutes les 200 nS, Dans ce cas là, le µC a une puissance de calcul de 5 MIPS (5 Millions d'instructions par secondes ! ! !).
La fréquence MAX est de 20 MHz pour les µC dont les références se terminent par -20.
Par exemples : 16F876-20 (20 MHz max) et 16F876-04 (4 MHz max).
La fréquence MIN est le continu.
Remarque : La consommation du circuit sera d'autant plus faible que la fréquence sera petite, cela peut être intéressant pour des applications de faible consommation (alimentation autonome).
Pour des applications faible consommation, on peut utiliser les séries LF (Low Frequency and Low Power).

4.3) Alimentation : VDD et VSS.

Ce sont les broches d'alimentation du circuit. Les tensions qui peuvent être appliquées vont :

- De 4,5V à 6V pour la gamme standard F.
- De 2 à 6V pour la gamme étendue LF.

L'intensité du courant consommé peut aller de 1µA à 10mA.
La consommation du µC sera fonction de :

- La tension d'alimentation.
- La fréquence interne.
- Le mode de fonctionnement.

De plus ces bornes doivent être découplées par deux condensateurs :

- 1µF électrolytique.
- 10nF céramique.

4.4) L'Interruption : RBO/INT.

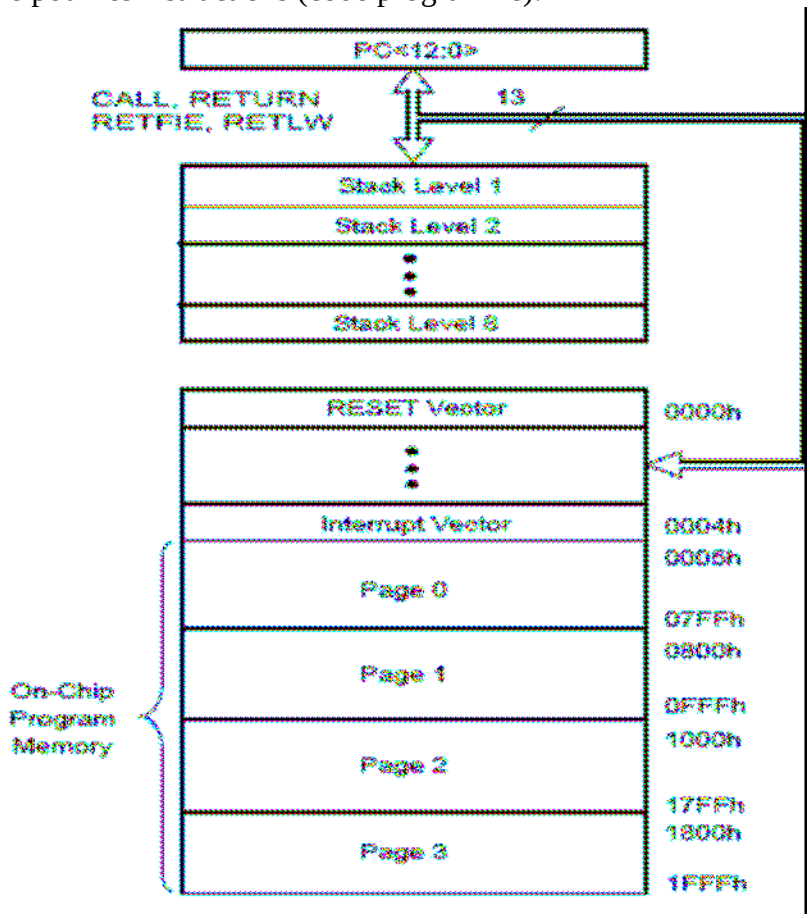
Cette broche a une double fonction elle peut être utilisée comme une broche standard RBO ou comme une entrée d'interruption INT.
Si cette broche est utilisée comme une entrée d'interruption externe, elle doit être maintenue à un niveau haut par l'intermédiaire de résistances de 10 kΩ pour ne pas déclencher d'interruptions imprévues, cela permet aussi de relier plusieurs sources d'interruptions sur une même ligne (OU CABLE).

5. L'unité centrale.

5.1) Organisation mémoire.

Comme les PICs utilisent un bus pour les instructions et un bus pour les données, il faut considérer deux plans mémoire l'un pour les instructions et l'autre pour les données ainsi que les registres internes.

7.1.1) Plan Mémoire pour les instructions (code programme).



Le plan mémoire est linéaire les adresses vont de 0000h à 1FFFh (8k mots de 14 bits), par page de 2K mots . On peut remarquer, le vecteur de reset est figé en 0000h.

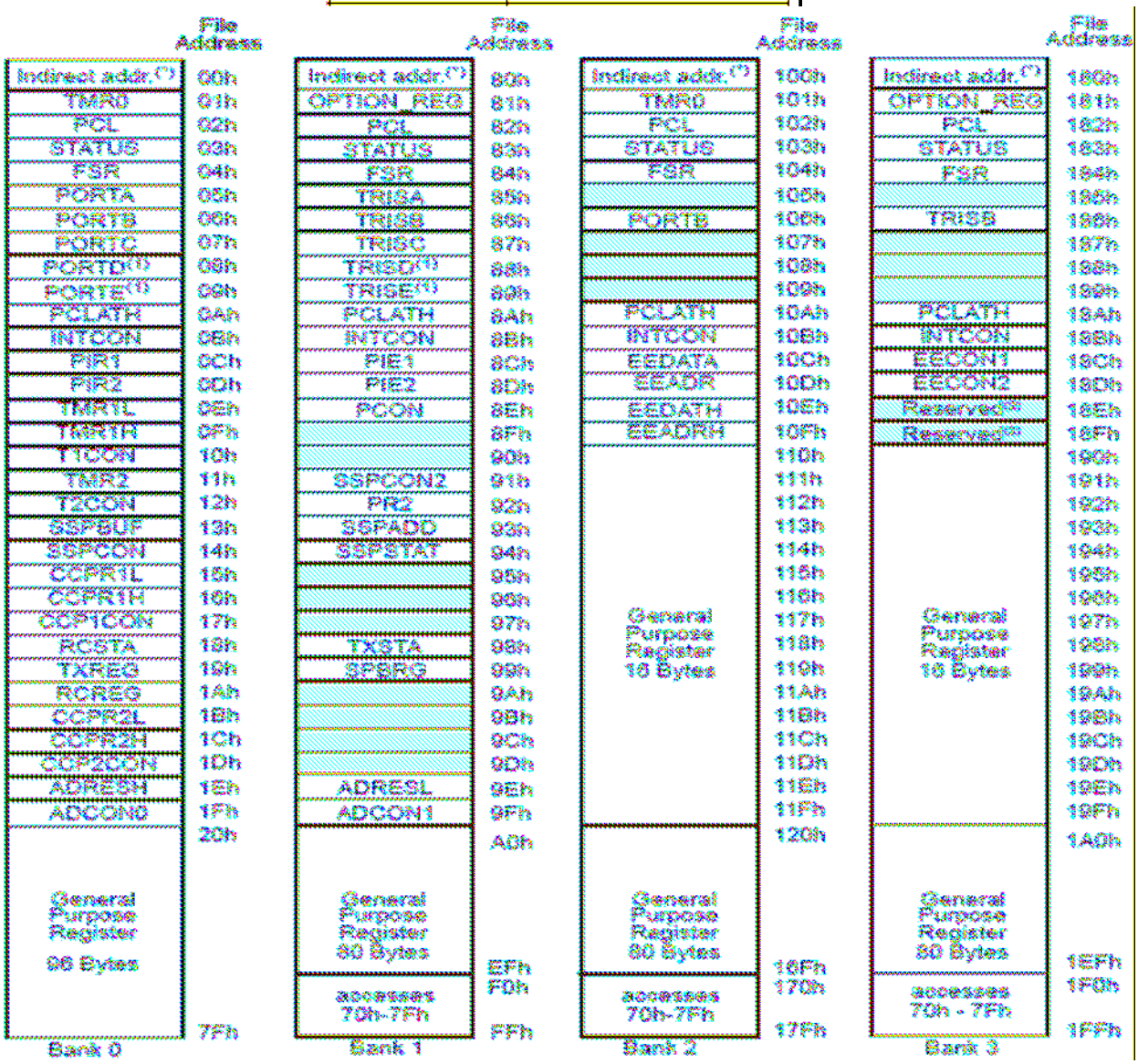
Les PICs n'ont qu'un seul vecteur d'interruption en 0004h. Lors d'une interruption, le sous programme associé devra déterminer quel périphérique a demandé une interruption.

La pile utilisée par les sous programmes n'est pas implantée en mémoire de donnée comme avec les microcontrôleurs classiques, mais dans la mémoire programme. Elles sont utilisées lors d'appels de sous programmes, on ne peut pas imbriquer plus de 8 sous programmes (Ce qui est déjà beaucoup ! !).

5.1.2) Plan Mémoire pour les données et registres internes (SFR : Special Function Register).

Le plan mémoire des données et des registres internes est découpé en 4 zones ou bank de 128 octets, pour accéder à une zone il faut positionner les bits RP0 (bit 5) et RP1 (bit 6) du registre STATUS

RP1 : RP0	Zone sélectionnée (BANK)
0 0	De 00h à 7Fh : BANK 0
0 1	De 80h à FFh : BANK 1
1 0	De 100h à 17Fh : BANK 2
1 1	De 180h à 1FFh : BANK 3



Unimplemented data memory locations, read as '0'.
*Not a physical register.
Note1: Ce registre n'est pas implémenté sur PIC
2: Ces registres sont réservés, il faut maintenir ces registres propres.

Les registres appelés General Purpose Register ne sont ni plus ni moins que des cases mémoires pour stocker les données.
Remarque : Les 4 banques mémoires ne facilitent pas la gestion de la mémoire.
Par exemple, pour accéder à la case mémoire 1A0h (BANK3), il ne faut pas oublier de positionner correctement les bits RP0 et RP1 du registre STATUS.

Code : Transfert du contenu de l'adresse 1A0h dans le registre W.

```
BSF     STATUS,5        ; RP0 = 1 Sélection de la BANK 3
BSF     STATUS,6        ; RP1 = 1 Sélection de la BANK 3
MOVF    0x20,0          ; Transfert du contenu de l'adresse 1A0h dans W.
```

Pourquoi les concepteurs des PICs ont été obligés de scinder en 4 banques mémoires ?
L'adresse des données est codée sur 7 bits, donc les adresses vont de 00h à 7Fh. Les 2 bits RP0 et RP1 sont 2 bits d'adresses supplémentaires et extérieurs, ce qui porte à 2^9 adresses possibles ou 512 registres.

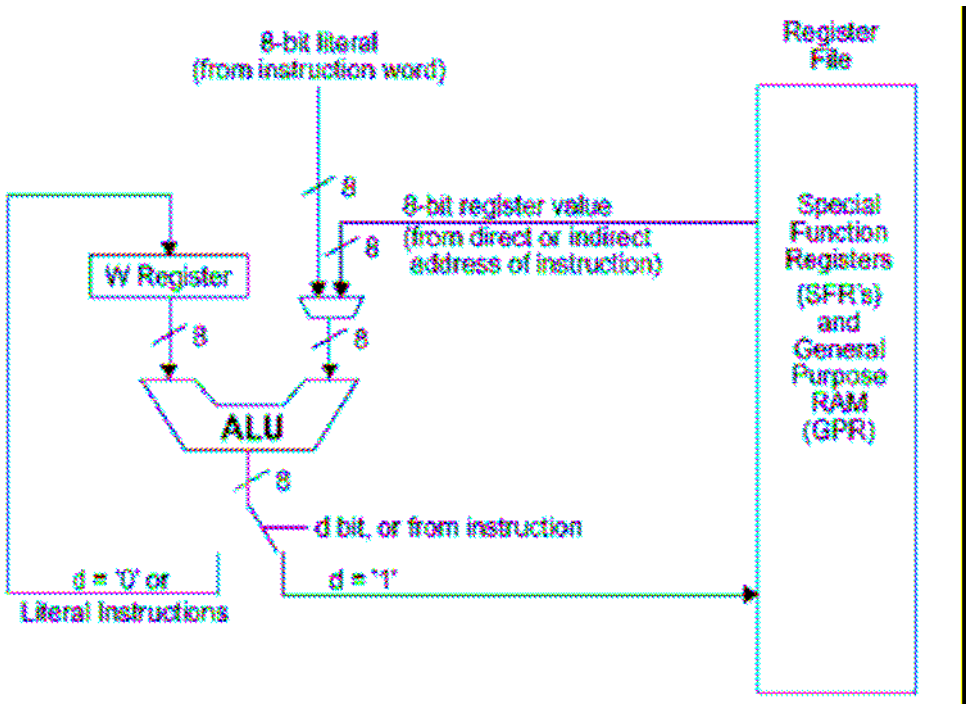
Code binaire de MOVF.
001000 d fff ffff d : destination 0 si W et 1 si F
fff ffff : adresse mémoire.

Pour l'exemple ci-dessus :
MOVF 0x20,0
d=0 et fff ffff = 010 0000

Donc le code de MOVF 0x20,0 est : 001000 0 010 0000

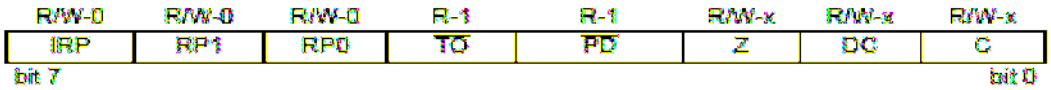
En conclusion, la gestion de la mémoire avec les PICs n'est pas très facile, la programmation demande de la rigueur.
La programmation des PICs avec des langages de hauts niveaux tels que BASIC, PASCAL et langage C permet de s'affranchir de la gestion des pages mémoires.

5.1.3) L'UAL : L'unité Arithmétique et Logique.



Elle est composée :
- D'un accumulateur 8 bits W : WORKING (travail), c'est lui qui effectue toutes les opérations arithmétiques et logiques.

- Un registre d'état 8 bits STATUS.



bit 7 IRP: Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 RP1:RP0: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 TO: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 PD: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 Z: Zero bit
1 = the result of an arithmetic or logic operation is zero
0 = the result of an arithmetic or logic operation is not zero

bit 1 DC: Digit carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions) (for borrow, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 C: Carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For borrow, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit isloaded with either the high, or low order bit of the source register.

6. Jeu d'instructions

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	df ff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	df ff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lf ff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0x xx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	df ff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	df ff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	df ff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	df ff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	df ff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	df ff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	df ff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lf ff	ffff		
NOP	-	No Operation	1	00	0000	0x x0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	df ff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	df ff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	df ff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	df ff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	df ff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bf ff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bf ff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bf ff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bf ff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kk kk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kk kk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kk kk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kk kk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kk kk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kk kk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kk kk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kk kk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kk kk	kkkk	Z	

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

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

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

LEGENDE DES TABLEAUX D'INSTRUCTIONS

d	*Détermine la destination :* *d=0 Registre Travail* *d=1 Registre Mémoire SFR ou RAM.*
f	*Adresse SFR ou RAM en hexa.*
fff ffff	*Adresse SFR ou RAM en binaire.*
k	*Valeur immédiate sur 8bits ou Adresse de destination sur 11 bits.*
b	*Valeur décimale, elle détermine le bit a modifier ou à tester.*
bbb	*Valeur binaire, elle détermine le bit a modifier ou à tester.*

Remarque: toutes les instructions ne durent qu'un seul cycle machine, sauf les instructions de sauts tels que GOTO, CALL.

7. Les modes d'adressages

7.1) Adressage inhérent ou implicite

7.1.1) Description.

Le mnémonique de l'instruction mentionne la donnée sur laquelle porte l'opération (contenu des registres), ou aucune donnée n'est nécessaire.

7.1.2) Syntaxe.

MNEMONIQUE

7.1.3) Exemples.

```
CLRW        ;  Mise à zéro de W
NOP         ;  aucune opération (temporisation)
SLEEP       ;  Mise en sommeil du µC
```

7.2) Adressage immédiat.

7.2.1) Description.

L'instruction porte sur une valeur constante indiquée immédiatement après le mnémonique.

7.2.2) Syntaxe.

MNEMONIQUE constante

7.2.3) Exemples.

```
MOVLW 255       ;  charge 0xFF dans W
ADDLW 0x20      ;  additionne 32 avec W et met le résultat dans W
```

7.3) Adressages direct et étendu.

9.3.1) Description.

Les PICs ne disposent pas vraiment de modes d'adressages DIRECT et ETENDU, l'adressage de la mémoire de données se fait dans la page sélectionnée par les BIT 5 (RP0) et BIT 6 (RP1) du registre STATUS.

9.3.2) Syntaxe.

MNEMONIQUE f,d

d = 0 Registre W comme destination (WORKING)

d = 1 Registre f comme destination (un des registres SFR)

9.3.3) Exemple.

```
PORTB       EQU     0x06
VARIABLE    EQU     0x20 ; page 0

    ; S'assurer que nous sommes bien en BANK 0
BCF         STATUS, 5       ; RP0 = 0 Sélection de la BANK 0
```

```
BCF         STATUS, 6          ; RP1 = 0 Sélection de la BANK 0

; PORTB < -  VARIABLE
  MOVF              VARIABLE, 0        ; Transfert le contenu de VARIABLE
                                       ; dans W
  MOVWF             PORTB              ; Transfert le contenu de W
                                        ; dans le registre PortB
```

7.4) Adressage relatif.

7.4.1) Description.

Ce mode d'adressage n'existe pas vraiment, mais des instructions permettent de réaliser des sauts de programme, ceux sont les instructions GOTO et CALL.

7.4.2) Syntaxe.

MNEMONIQUE Adresse

7.4.3) Exemple N°1.

Deux cas sont à considérer :

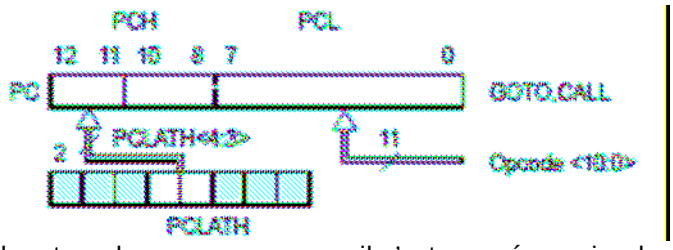
1) Les sauts dans la même page mémoire :

```
CALL TEMPO          ; Appel du SP TEMPO
GOTO FIN            ; Branchement à l'étiquette FIN
```

2) Les sauts dans une page mémoire différente, dans ce cas il faut positionner correctement les bits 4 et 3 du registre PCLATCH, pour accéder à la bonne page mémoire.

```
BSF PCLATCH,3       ; 1 et 1 Sélection de la BANK 3 de 1800h à 1FFFh
BSF PCLATCH,4
CALL CONV
```

En effet pourquoi PCLATCH et pas PCH ? PCH n'est pas accessible directement il faut passer par PCLATCH pour que les bits 4 et 3 de celui-ci soient recopiés dans PCH.



Remarque : Pour le retour de sous programme, il n'est pas nécessaire de ce préoccuper de ces bits car la valeur du PC est mémorisée sur 13 bits dans la pile.

7.5) Adressage indirect ou encore indexé.

7.5.1) Description.

Les PICs disposent à travers les registres INDF(ou f0) et FSR(ou f4) d'un mode d'adressage indexé, la structure est un peu particulière, FSR est le registre d'index et INDF permet d'accéder à son contenu.

7.5.2) Syntaxe.

MNEMONIQUE INDF,d

7.5.3) Exemple :

PORTB <- TAB_VAL[4]

Récupérer le 4ème élément d'une table TAB_VAL ;

```
; PORTB  < -   TAB_VAL [4]
MOVLW        TAB_VAL          ; W <- Adresse de TAB_VAL
ADDLW        4                ; W <- W + 4
MOWF         FSR              ; Adresse + 4 dans le registre d'index FSR
MOVF         INDF, 0          ; Transfert du contenu de TABLE[4] dans W
MOVWF        PORTB            ; Transfert du contenu de W sur le PORTB
```

7.6) Manipulation de bits.

7.6.1) Forçage de bits.

7.6.1.1) Description.

Il s'agit de 2 instructions permettant de mettre à 0 ou 1 un bit d'un octet de l'espace mémoire SFR. Elles sont le plus souvent utilisées pour positionner des bits des registres du µC.

7.6.1.2) Syntaxe.

```
BSF      f, b      pour mettre à 1
ou BCF   f, b      pour mettre à 0
```

7.6.1.3) Exemples.

```
BCF      PORTA, 2          ; Mise à 0 du bit 2 du PORTA
BSF      STATUS, 0         ; Mise à 1 du bit 0 du registre STATUS
                           ; C'est-à-dire la CARRY
```

7.6.2) Test de bits.

7.6.2.1) Description.

Il s'agit de 2 instructions permettant de tester un bit d'un octet de l'espace mémoire SFR. Elles sont le plus souvent utilisées pour déterminer l'état des bits des registres du µC.

En fonction du résultat du test :

- le programme se poursuit avec l'instruction suivante (résultat du test faux)
- le programme saute l'instruction qui suit le test.

7.6.2.2) Syntaxe.

```
BTFSS      f,b
ou BTFSC   f,b
```

7.6.2.3) Exemple.

```
MOVF       CMP, 1        ; Transfert du contenu CMP dans CMP
                         ; Cela permet de tester si le contenu
                         ;  de CMP est nul en positionnant le bit Z
BTFSS      STATUS, Z     ; Test du bit Z ?
```

```
        GOTO    SINON    ; Z=0 alors exécuter le code pour SINON
        MOVLW   0xFF     ; Alors CMP <- 0xFF
        MOVWF   CMP
        GOTO    FSI      ; allez en fin de si
SINON   DECF    CMP, 1   ; CMP <- CMP -1 Décrémenter CMP
        ……..
FSI     …….
```

8. Les ports d'entrées / sorties

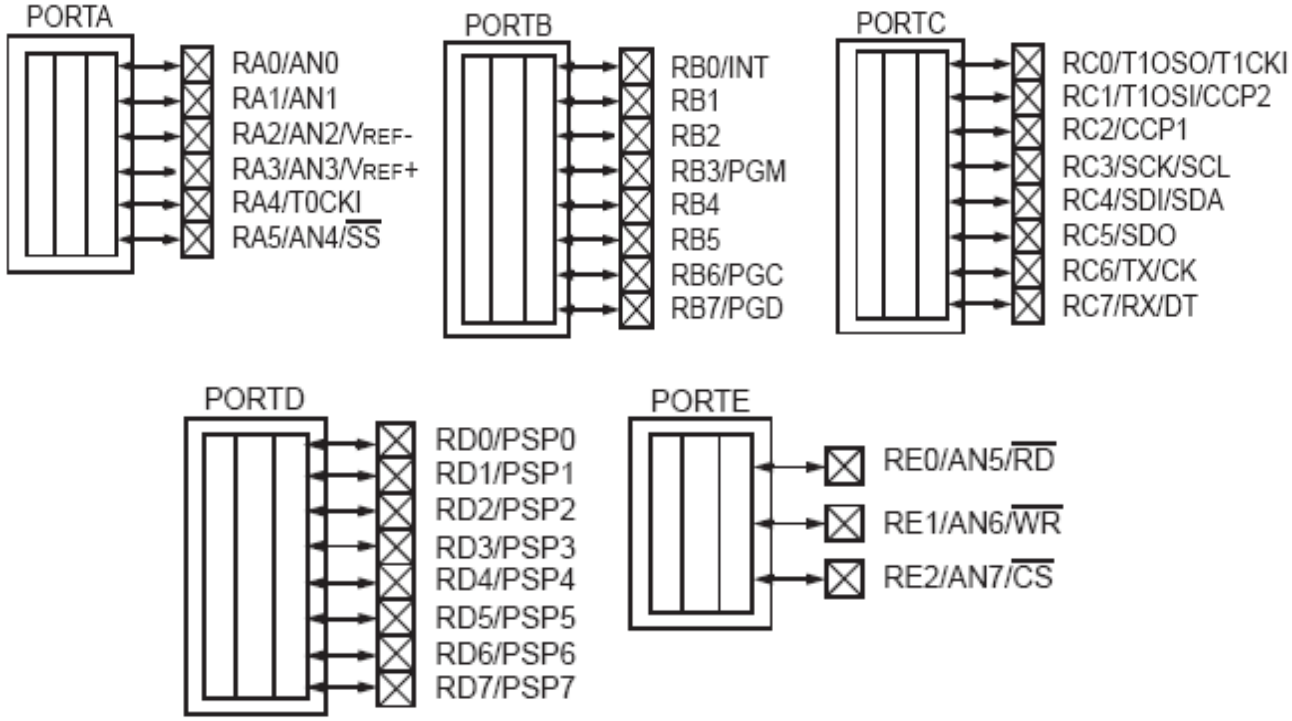
8.1. Généralités.

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

La plupart des lignes de PORTs ont une double fonction.

- Le PORT A (5 bits) I/O pure et/ou convertisseur analogique et/ou TIMER 0.
  La broche RA4 du PORT A (Entrée du TIMER 0 T0CKI) est du type DRAIN OUVERT.
- Le PORT B (8 bits) I/O pure et/ou programmation in situ ICSP/ICD (Broche RB3/PGM, RB6/PGC et RB7/PGD) et l'entrée d'interruption externe RB0/INT.
  Remarque : Si le PIC est utilisé en mode ICSP/ICD il faut laisser libre les broches RB3/PGM, RB6/PGC ainsi que RB7/PGD) et les configurer en entrée.
- Le PORT C (8 bits) I/O pure et/ou TIMER 1 et/ou SPI / I2C et/ou USART.
- Le PORT D (8 bits) I/O pure et/ou port parallèle 8 bits associé au PORT E.
- Le PORT E (3 bits) I/O pure et/ou pilotage du PORT E RE0/RD, RE1/WR et RE2/CS.

Toutes les lignes de PORTs peuvent fournir un courant de 25mA par ligne de PORT. Une limite de 40mA par PORT doit être respectée pour des questions de dissipation.



8.2 Configuration des PORTx , les registres PORTx et TRISx.

Tous les ports sont pilotés par deux registres :

- Le registre de PORTx, si le PORT x ou certaines lignes de PORT X sont configurées en sortie, ce registre détermine l'état logique des sorties.
- Le registre TRISx, c'est le registre de direction. Il détermine si le PORTx ou certaines lignes de port sont en entrée ou en sortie. L'écriture d'un 1 logique correspond à une entrée (1 comme Input) et l'écriture d'un 0 logique correspond à une sortie (0 comme Output).

Au RESET toutes les lignes de ports sont configurées en entrées.

Remarque : Les registres TRISx appartiennent à la BANQUE 1 des SFR.

Lors de l'initialisation du µC il ne faut pas oublier de changer de page mémoire pour les configurer.

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

SENS	NC	NC	S	E	S	S	S	E
PORTA	NC	NC	RA5	RA4	RA3	RA2	RA1	RA0

SENS	E	E	S	S	E	S	S	E
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Remarque toutes les lignes de sorties des PORTs sont mises à zéro.

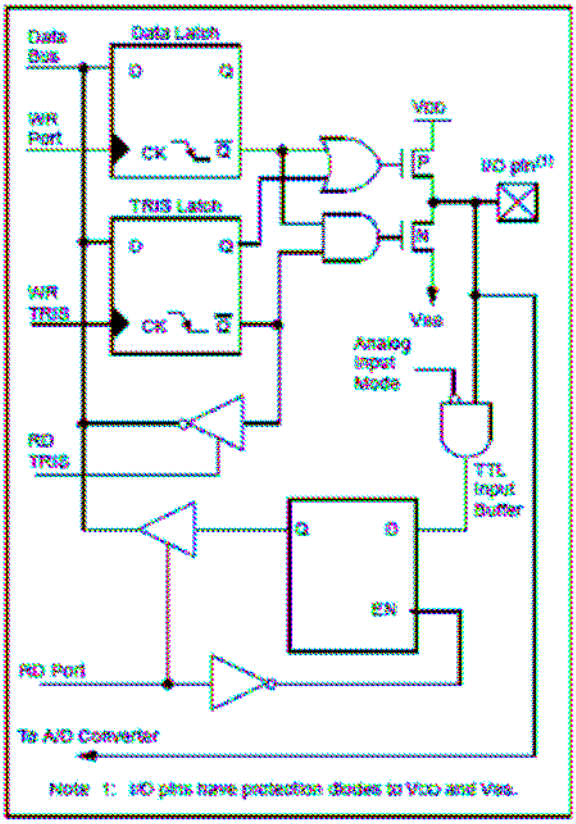
```
; Mise à zéro des registres de données des ports A et B
clrf PORTA
clrf PORTB
; Configuration des PORTA et B

; Accès aux registres TRISx (Banque mémoire 1)
bsf STATUS,RP0 ; RP0 = 1
bcf STATUS,RP1 ; RP1 = 0

; Configuration des registres de direction
                                ; Configuration du PORTA X X S E S S S E
movlw B'11010001'               ; valeur binaire 1 1 0 1 0 0 0 1
movwf TRISA
                                ; Configuration du PORTB E E S S E S S E
movlw B'11001001'               ; valeur binaire 1 1 0 0 1 0 0 1
movwf TRISB

; Retour en banque mémoire 0
bcf STATUS,RP0                  ; RP0 = 0
bcf STATUS,RP1                  ; RP1 = 0
…….
…….
```

8.3. Le PORT A.

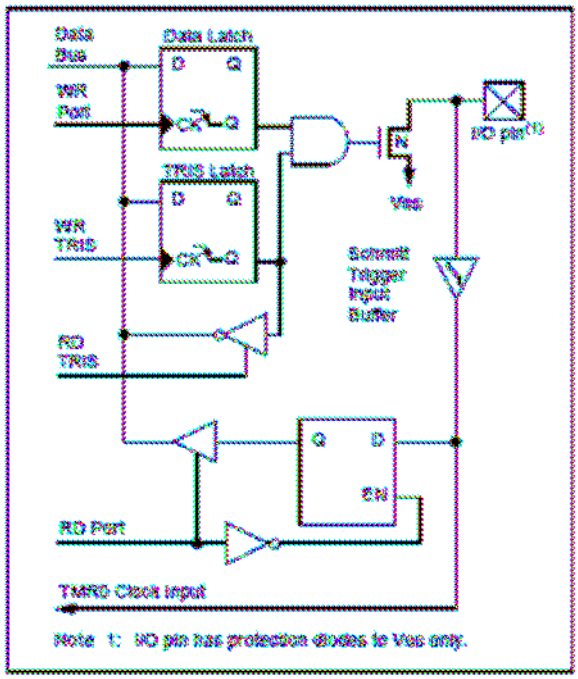
Le PORT A (5 bits) I/O pure et/ou convertisseur analogique et/ou TIMER 0.

Attention à PA4 (Entrée du TIMER 0 T0CKI), elle est de type DRAIN OUVERT.

BLOCK DIAGRAM OF RA3: RA0 AND RA5 PINS



BLOCK DIAGRAM OF RA4/T0CKI PIN



PORT A FUNCTIONS

Name	Bit#	Buffe r	Func tion
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2	bit2	TTL	Input/output or analog input.
RA3/AN3/VREF	bit3	TTL	Input/output or analog input or VREF.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/SS/AN4	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Registres associés au PORT A

Addre ss	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000

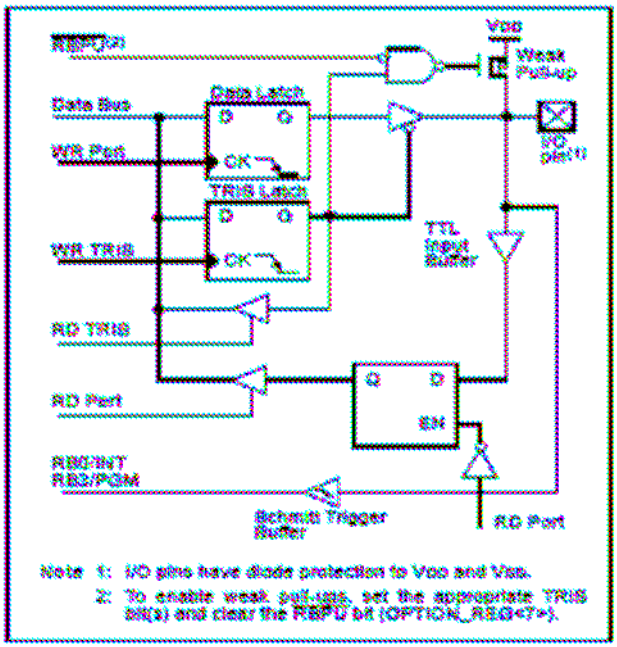
Legend: :x = unknown (inconnu), u = unchanged (inchangeable) , - = unimplemented locations read as '0' (location non implémentée lire comme “0”).

8.4. Le PORT B.

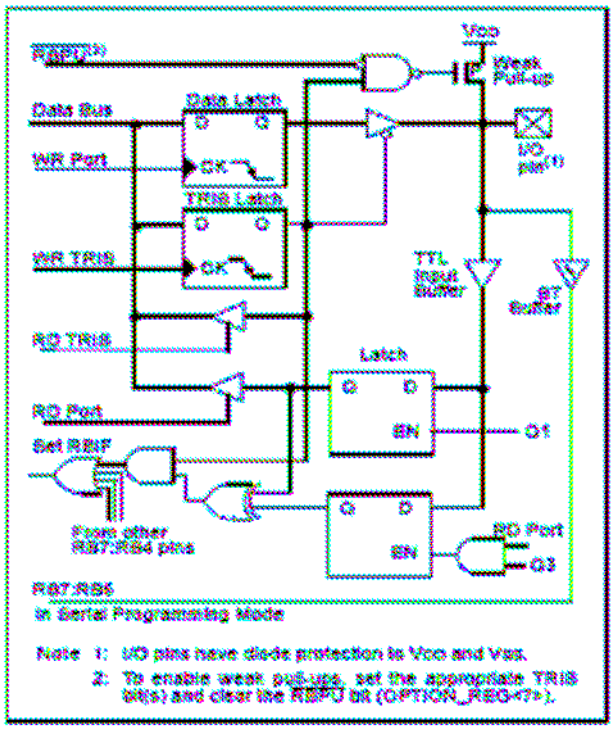
Le PORT B dispose de (8 bits) I/O pure et/ou programmation in situ ICSP/ICD (Broche RB3/PGM, RB6/PGC et RB7/PGD) et une entrée d’interruption externe RB0/INT.
Il dispose de deux fonctions spéciales :

- La première c'est la possibilité de configurer toutes ses lignes avec une résistance de PULL-UP en configurant le bit RBPU à 0 du registre OPTION.
- La deuxième, c'est la possibilité de générer une interruption sur un changement d'état des broches RB4 à RB7. C'est très pratique pour la gestion des claviers matricés.

BLOCK DIAGRAM OF RB3:RB0 PINS



BLOCK DIAGRAM OFRB7:RB4 PINS



PORT B FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT	bit0	TTL/ST(1)	input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	input/output pin. Internal software programmable weak pull-up.
RB3/PGM(3)	bit3	TTL	input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit4	TTL	input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit5	TTL	input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6/PGC	bit6	TTL/ST(2)	input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST(2)	input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: Low Voltage ICSP Programming (LVP) is enabled by default, which disables the RB3 I/O function. LVP must be disabled to enable RB3 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

Registres associés au PORT B

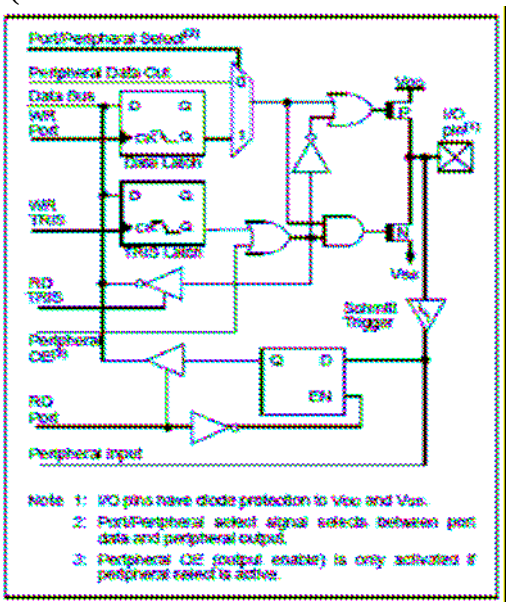
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown (inconnu) , u = unchanged (inchangeable) .

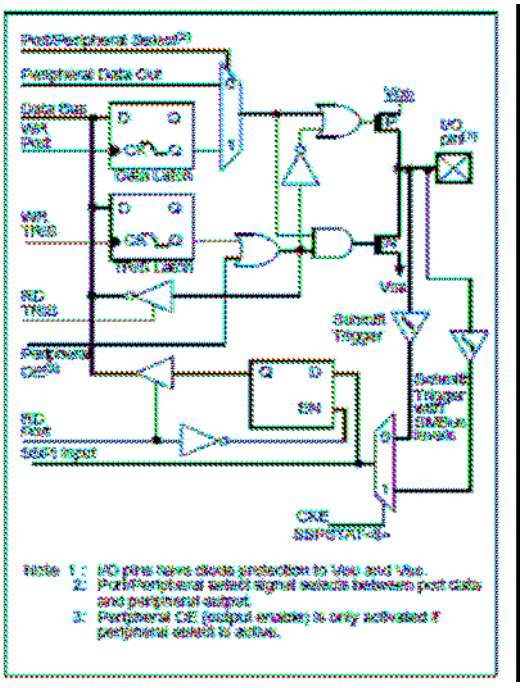
8.5. Le PORT C.

Le PORT C (8 bits) I/O pure qu'il partage avec le TIMER 1, la liaison SPI / I2C et l'USART.

PORT C BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC <2:0 >, RC < 7:5 >



PORT C BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC < 4:3 >



PORT C FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin or USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin or USART Asynchronous Receive or Synchronous Data.

Légende: ST = Schmitt Trigger input

Registres associés au PORT C

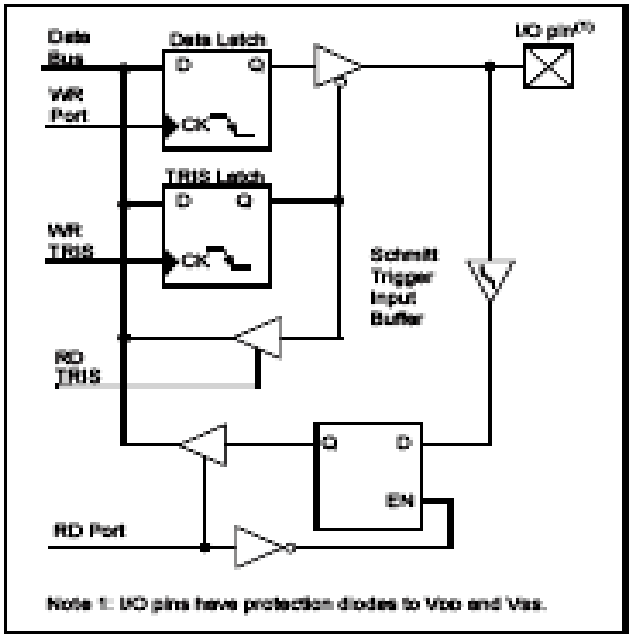
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown (inconnu), u = unchanged (inchangeable)

8.6. Les PORT D et E.

Le PORT D (8 bits) I/O et PORT E (3 bits) utilisent la même type structure interne.

PORT D BLOCK DIAGRAM (IN I/O PORT MODE)



PORT D FUNCTIONS

Name	Bit	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL[1]	Input/output port pin or parallel slave port bit0.
RD1/PSP1	bit1	ST/TTL[1]	Input/output port pin or parallel slave port bit1.
RD2/PSP2	bit2	ST/TTL[1]	Input/output port pin or parallel slave port bit2.
RD3/PSP3	bit3	ST/TTL[1]	Input/output port pin or parallel slave port bit3.
RD4/PSP4	bit4	ST/TTL[1]	Input/output port pin or parallel slave port bit4.
RD5/PSP5	bit5	ST/TTL[1]	Input/output port pin or parallel slave port bit5.
RD6/PSP6	bit6	ST/TTL[1]	Input/output port pin or parallel slave port bit6.
RD7/PSP7	bit7	ST/TTL[1]	Input/output port pin or parallel slave port bit7.

Legend: ST = Schmitt Trigger input, TTL = TTL input

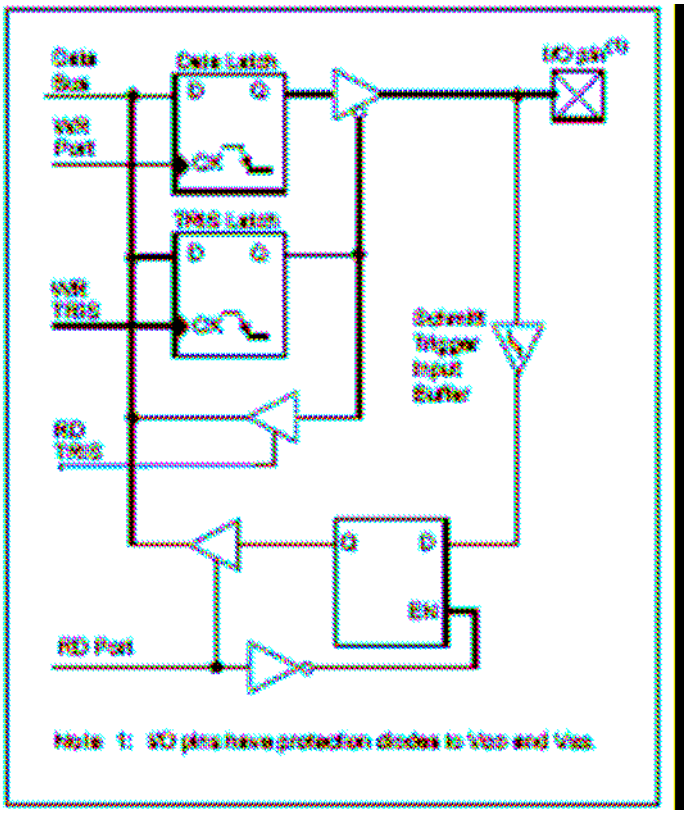
Note 1:Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

Registres associés au PORTE D

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0x9	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	xxxx xxxx
0x89	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
0x9	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown (inconnu), u = unchanged (inchangeable).

PORTE BLOCK DIAGRAM (IN I/O PORT MODE)



PORT E FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/$\overline{RD}$/AN5	bit0	ST/TTL[1]	I/O port pin or read control input in Parallel Slave Port mode or analog input: $\overline{RD}$ 1 = Idle 0 = Read operation. Contents of PORTD register are output to PORTD I/O pins (if chip selected)
RE1/$\overline{WR}$/AN6	bit1	ST/TTL[1]	I/O port pin or write control input in Parallel Slave Port mode or analog input: $\overline{WR}$ 1 = Idle 0 = Write operation. Value of PORTD I/O pins is latched into PORTD register (if chip selected)
RE2/$\overline{CS}$/AN7	bit2	ST/TTL[1]	I/O port pin or chip select control input in Parallel Slave Port mode or analog input: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

Reagistres associés au PORT E

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	-0- 0000	-0- 0000

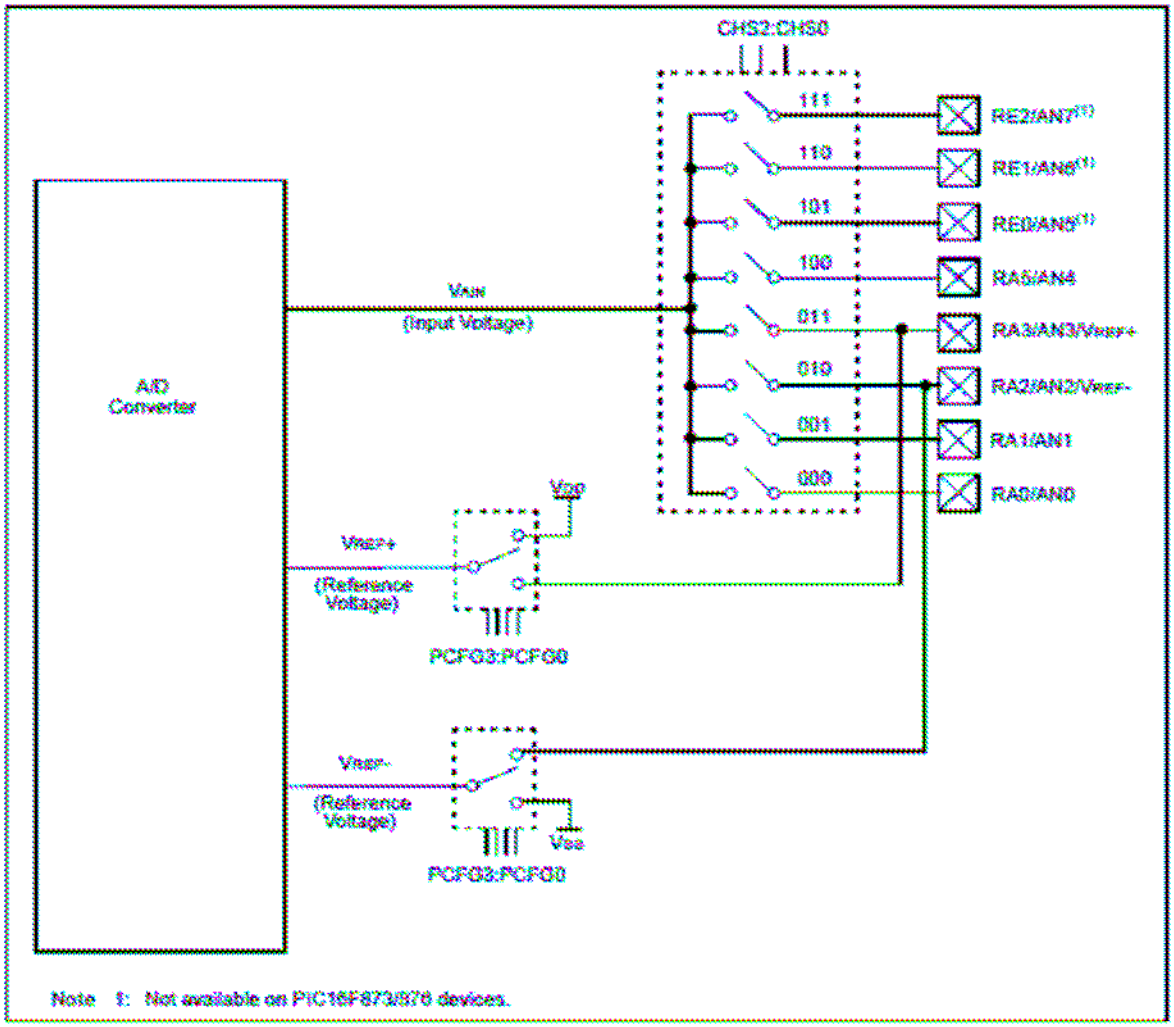
Legend: x = unknown (inconnu), u = unchanged (inchangeable).

9. Le convertisseur analogique numérique.

Le convertisseur analogique numérique est à approximations successives et il possède une résolution de 10 bits. Il est composé de :
- Un multiplexeur analogique **5** voies (**PIC16F876**) ou **8** voies **PIC16F877**).
- Un échantillonneur bloqueur.
- Un Convertisseur Analogique Numérique de **10** bits.

9.1. Organisation interne.

A/D BLOCK DIAGRAM



Remarque : Les entrées analogiques **RE1/AN5**, **RE2/AN6** et **RE3/AN2** sont disponibles avec les **PICs 16F877** et **16F874**.

De plus la broche **RA4** n'est pas concernée pas le convertisseur.

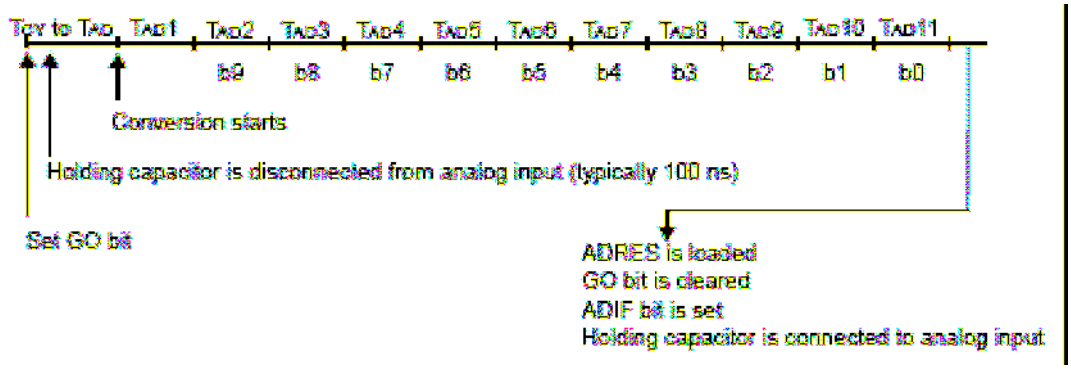
9.2. Fonctionnement du convertisseur.

La conversion se passe en **2** temps :

- **1er temps** le signal à convertir est appliqué sur l'entrée à convertir, ce signal doit être présent au moins pendant le temps **Tacq (temps d'acquisition** environ **20µS** pour **5V**).
- **2ème temps** la conversion, approximations successives.

Le temps de conversion minimum est de **12 TAD** (**TAD** c'est le temps de conversion dépendant de l'horloge interne, typiquement **1,6µS**).
Une conversion commence toujours par la mise à **1** du bit **GO/DONE** du registre **ADCON0**. Lorsque la conversion est terminée se bit repasse à **0**.

Donc pour pouvoir lire le résultat dans les registres **ADRESL** et **ADRESH** il suffit d'attendre que le bit **GO/DONE** passe à **0**.



La valeur résultante **N** de la conversion **ADRSH : ADRESL** est égale à :
N (valeur numérisée) = ((V_{IN} - V_{REF-}) / (V_{REF+} - V_{REF-})) * 1023)
Si $V_{REF+} = V_{DD} = 5$ V et $V_{REF-} = V_{SS} = 0$ V alors
N (valeur numérisée) = 1023 * (VIN / 5)
Mais avant de réaliser une conversion il faut définir la configuration du convertisseur :
- Le nombre d'entrées analogiques.
- Le nombre d'entrées logiques.
- Le type de tension de référence :
- Interne $V_{REF} = V_{DD} - V_{SS}$.
- Externe, soit $V_{REF} = V_{REF+} - V_{SS}$ ou $V_{REF} = V_{REF+} - V_{REF-}$.

Cette configuration ce fait à travers le registre **ADCON1**, voir page suivante.

9.3. Le registre ADCON1.

Il permet de choisir une configuration parmi les **16** proposées.
Remarque : La configuration de ce registre **ADCON1** ne dispense pas de configurer les registres de directions des **PPORTA** et **PORTE** respectivement **TRISA** et **TRISE**.

ADCON1 REGISTER (ADDRESS 9Fh)

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

bit 7 ADFM: A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.

OFPPT / DRIF – G.E.

0 = Left justified. 6 Least Significant bits of ADRESL are read as ‘0’.

bit 6-4 Unimplemented: Read as '0'

bit 3-0 PCFG3:PCFG0: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7[1] RE2	AN6[1] RE1	AN5[1] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs[2]
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss	2/1
011x	D	D	D	D	D	D	D	D	VDD	Vss	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input; D = Digital I/O

Note 1 : These channels are not available on PIC16F873/876 devices.

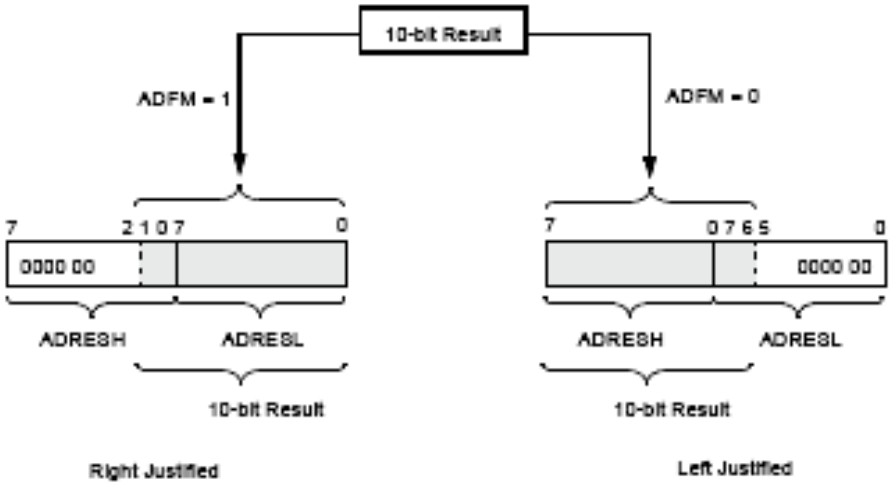
2 : This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

De plus le bit **ADFM** permet de choisir entre deux types de justification pour le résultat.

- Si **ADFM=1** alors le résultat sera justifié à **droite** dans les registre **ADRESH** et **ADRESL**, c'est-à-dire **ARESL** contient les bits **7 à 0** du résultat de la conversion et **ADRESH** contient **6** zéros suivi des bit **9** et **8** du résultat de la conversion.

- Si **ADFM=0** alors le résultat sera justifié à **gauche,** c'est-à-dire **ARESH** contient les bits **9 à 2** du résultat de la conversion et **ADRESH** contient les bit **1** et **0** du résultat de la conversion suivi de **6** zéros.

A/D RESULT JUSTIFICATION



9.4. Le registre ADCON0.

Ce registre permet de définir l'horloge de conversion (bit **ADCS1** et **ADCS0**), le canal à convertir (**CHS2**, **CHS1** et **CHS0**) et **ADON** bit de mise en fonctionnement.

Remarque : Lors de la mise en fonctionnement du **CAN** par le bit **ADON**, Le bit **GO/** $\overline{DONE}$ ne doit pas être modifié en même temps, c'est-à-dire dans la même instruction. .

ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1 : ADCS0 : A/D Conversion Clock Select bits**
00 =FOSC/2
01 =FOSC/8
10 =FOSC/32
11 =FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0: Analog Channel Select bits**
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5) (1)
110 = channel 6, (RE1/AN6) (1)
111 = channel 7, (RE2/AN7) (1)

bit 2 **GO/** $\overline{DONE}$ **: A/D Conversion Status bit**
If ADON = 1:
1 =A/D conversion in progress (setting this bit starts the A/D conversion)
0 =A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 Unimplemented: Read as '0'

bit 0 **ADON: A/D On bit**1 = A/D converter module is operating 0 = A/D converter module is shut-off and consumes no operating current

Note 1 : These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Le choix de l'horloge est déterminé par les bits **ADCS1** et **ADCS0**, sachant que le temps **TAD** doit être au minimum de **1,6 µS**.

TAD vs. MAXIMUM DEVICE OPERATING FREQUENCIES (STANDARD DEVICES (C))

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS1:ADCS0	Max.
2Tosc	00	1.25 MHz
8Tosc	01	5 MHz
32Tosc	10	20 MHz
RC(1, 2, 3)	11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 s, but can vary between 2-6 s.
2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recom-mended for SLEEP operation.
3: For extended voltage devices (LC), please refer to the Electrical Characteristics (Sections 15.1 and 15.2).

9.5. Exemple d'utilisation.

Exemple : On souhaite obtenir la configuration suivante avec un **PIC16F877** :

RE2 : Sortie logique **RE1** : Sortie logique **RE0** : Entrée logique
RA5 : Sortie logique **RA4** : Entrée Logique **RA3** : Entrée analogique
RA2 : Entrée logique **RA1** : Entrée analogique **RA0** : Entrée analogique
Tension de référence $\mathbf{V_{REF}} = \mathbf{V_{DD}} - \mathbf{V_{SS}} = \mathbf{5\ V}$ et fréquence du quartz égale à **12 MHz**.

Programme en assembleur.

Remarque : Toutes les lignes de sorties des **PORTs** sont mises à zéro.

```
        ; Mise à séro des registres de données des ports A et B
        clrf   PORTA
        clrf   PORTB

        ; Configuration des registres de directions des PORT A et B
        ; Accès aux registres TRISx  (Banque mémoire 1)
        bsf    STATUS,RP0    ; RP0 = 1
        bcf    STATUS,RP1    ; RP1 = 0

        ; Configuration des registres de directions
        ; Configuration du PORTA X X S B B B B B
        movlw  B'11011111'   ; valeur binaire             1 1 0 1 1 1 1 1
        movwf  TRISA
        ; Configuration du PORTB 0 0 0 0 0 S S B
        movlw  B'00000001'   ; valeur binaire          0 0 0 0 0 0 0 1
        movwf  TRISB

        ; Configuration du registre ADCON1 Page 1
        ; ADFM1 = 1 justification à droite du résultat
        ; PCFG 3:0 0100 => RB3:RB0 Type D comme Digitale
        ;                  => RA5 : D, RA3 : A comme Analogique
        ;                  => RA2 : D, RA1 : A et RA0 : A
        movlw  B'10000100'   ; valeur binaire          1 0 0 0 0 1 0 0
        movwf  ADCON1

        ; Retour en banque mémoire 0
        bcf    STATUS,RP0    ; RP0 = 0
        bcf    STATUS,RP1    ; RP1 = 0

        ; Configuration du registre ADCON0 Page 0
        ; ADCS1 et ADSC0 = 1 0 Fréquence Max 20MHz
        ; ADON = 1 Mise en route du CAN
        ; 0 pour les autres bits
        movlw  B'10000001'   ; valeur binaire          1 0 0 0 0 0 0 1
        movwf  ADCON0

        ; Conversion du canal RA3
        ; Sélection du canal 3 avec les bits CHS2, CHS1 et CHS0 :  0 1 1
        ; GO/DONE = 1 Lancement d'une conversion
        bcf    ADCON0,CHS2
        bsf    ADCON0,CHS1
        bsf    ADCON0,CHS0
        bsf    ADCON0,GO     ; Déclenchement de la conversion
ATT     btfsc  ADCON0, GO_DONE      ; attendre la fin de conversion
        goto   ATT
        ; fin de conversion, lecture du résultat
        movf   ADRESH,0      ; Partie haute
        movwf  RES_HAUT,1
        bsf    STATUS,RP1    ; Passage en page 1
        movf   ADRESL,0      ; Partie basse
        bsf    STATUS,RP0    ; Passage en page 0
        movwf  RES_BAS,1     ;
```

10. Les timers.

Les **PICs 16F87x** disposent de **3** timers :

- **Le timer 0 (8 bits):** Il peut être incrémenté par des impulsions extérieures via la broche **(TOCKI/ RA4)** ou par l'horloge interne (**Fosc/4**).

- **Le timer 1 (16 bits):** Il peut être incrémenté soit par l'horloge interne, par des impulsions sur la broche **T1CKI/RC0** ou par un oscillateur **(RC ou quartz)** connecté sur les broches **TOSO/RCO** et **T1OSI/RC1**.

- **Le timer 2 (8 bits) :** Il est incrémenté par l'horloge interne, celle peut être pré divisée.

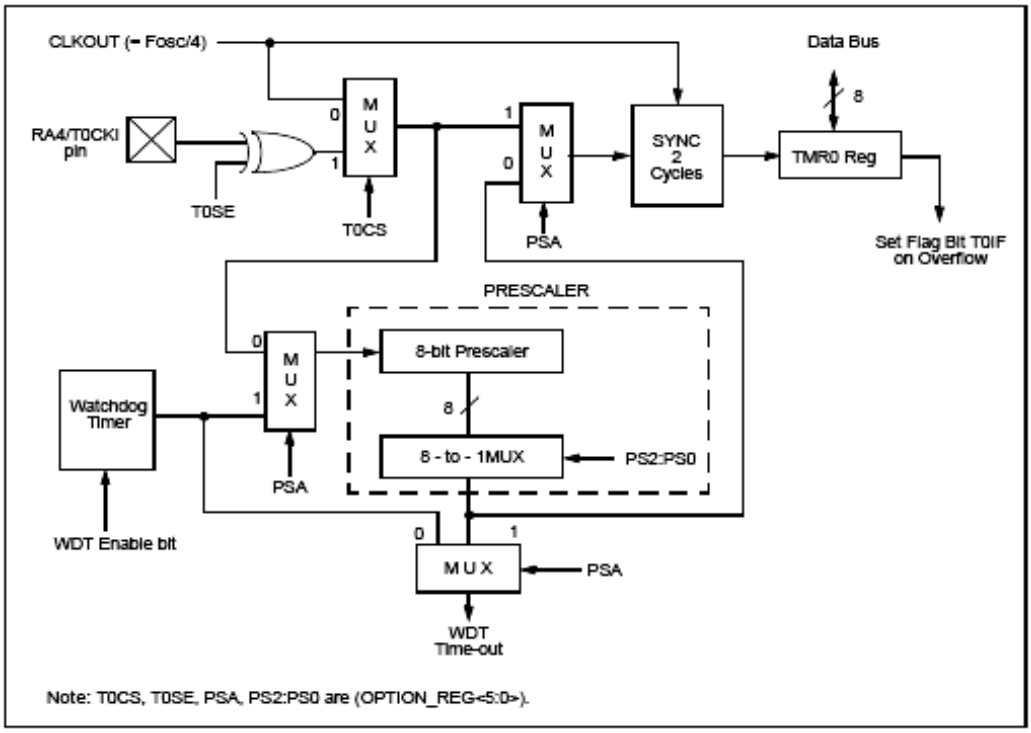
Tous ces **timers** peuvent déclencher une interruption interne, si ils ont été autorisés.

10.1 Le timer 0.

10.1.1) Présentation :

C'est le plus ancien des **timers** implantés dans les **PICs**, son ancienne appellation était **RTC**, pour **Real Time Clock** (horloge temps réelle). On peut se servir de celui-ci pour générer des événements périodiques, comme le rafraîchissement d'afficheurs multiplexés ou l'incrémentation de variables (secondes, minutes ….).
Celui-ci est incrémenté soit par l'horloge interne (**Fosc/4**) ou par une horloge appliquée sur la broche
TOCKI/ RA4 .

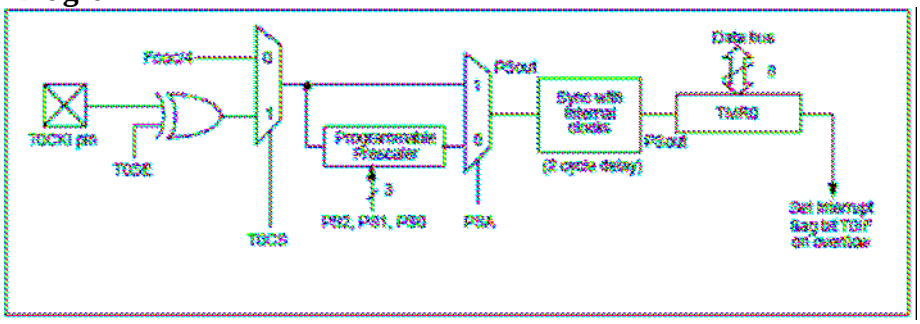
BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



Comme on peut le constater sur ce schéma le **timer 0** partage avec le chien de garde **Watchdog** le pré diviseur. Celui-ci est affecté à l'un ou à l'autre, suivant la valeur du bit **PSA (0 : Timer0 et 1 : chien de garde).**

On peut obtenir un schéma simplifié du fonctionnement du **timer 0** sans le chien de garde.

Timer0 Block Diagram



10.1.2) Fonctionnement :

Le bit **T0CS** permet de choisir l'horloge, interne (**Fosc/4**) ou externe **T0CKI/RA4**. Dans ce dernier cas l'incrémentation du **timer 0** peut se faire soit sur front montant ou descendant suivant la valeur du bit **T0SE**. Le bit **PSA** de choisir si horloge permet de pré diviser l'horloge d'un rapport allant de **2** à **256**. La valeur de pré division est fixée par les bits **PS2, PS1 et PS0**.

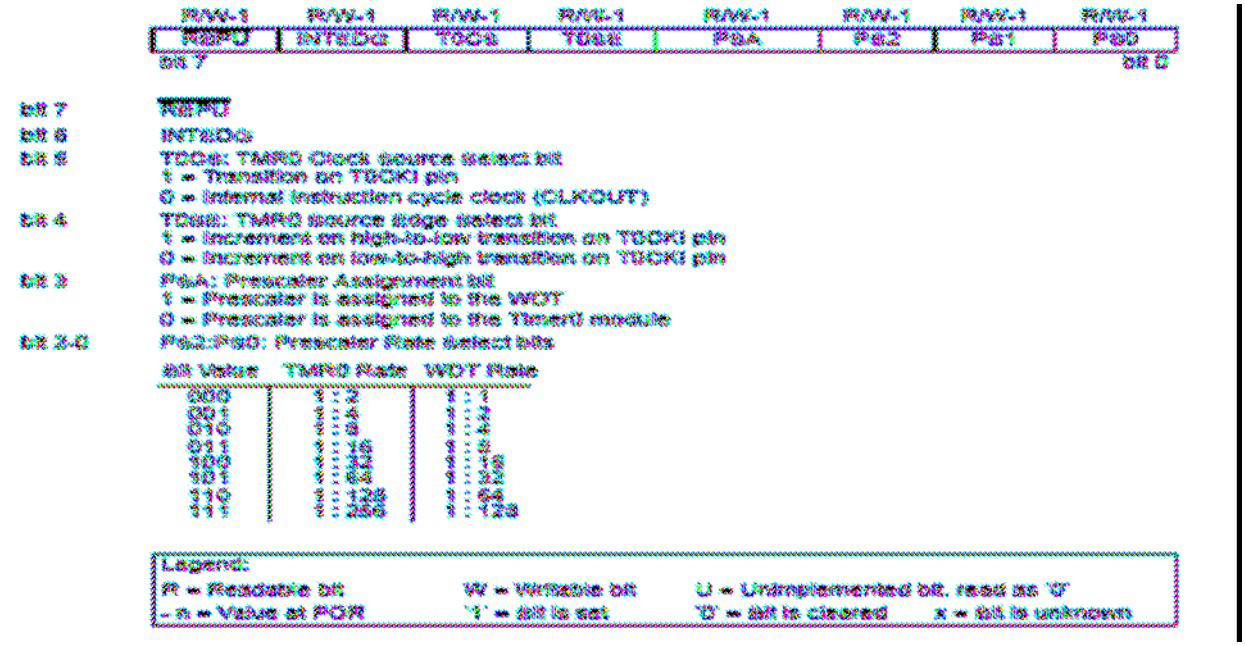
Quand le contenu du **timer 0** passe de **FF** à **00** le bit **T0IF** passe à **1** pour signaler un débordement, si le bit **T0IE** est à **1** alors une interruption **timer 0** est déclenchée.
Le contenu du **timer 0** peut être modifié à tout instant, à une condition près, la nouvelle valeur inscrite dans le registre **TMR0** sera prise en compte après **3** cycles machines.

<u>Remarque importante :</u> Lorsque le bit **T0IF** passe à **1** lors du passage de la valeur **FFh** à **00h** du registre **TMR0**, il doit être remis à **0** de façon logicielle par une instruction du type : **bcf INTCON,T0IF**

10.1.3) Configuration et registres associés :

La configuration du **timer 0** passe par les registres **TMR0** (adresse **01h)**, **OPTION_REG** (adresse **81h page 1**) et **INTCON** (adresse **0Bh** : toutes les pages).

OPTION_REG REGISTER (@ 81h or 181h)



REGISTERS ASSOCIATED WITH TIMER 0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
01h,101h	TMR0	Timer0 Module's Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h,181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

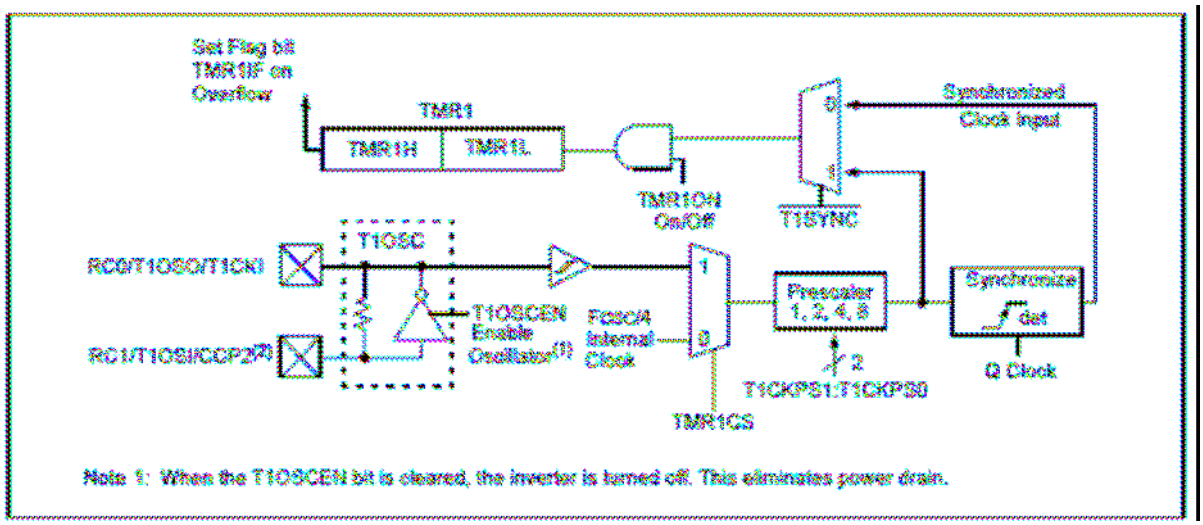
Legend : x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by Timer 0.

10.2. Le timer 1.

10.2.1) Présentation :

Il fonctionne sur le même principe que le **timer 0**, mais il est plus moderne dans sa conception. C'est un compteur **16** bits.

TIMER1 BLOCK DIAGRAM



10.2.2) Fonctionnement :

Le bit **TMR1CS** permet de choisir l'horloge soit interne (**Fosc/4**), externe **T1CKI** ou un oscillateur à quartz connecté sur les broches **T1OSO** et **T1OSI**.
Les bits **T1CKPS1 et T1CKPS0** permettent de choisir la valeur de la pré division à appliquer à l'horloge choisie, de **1** à **8**.
Le bit **T1SYNC** permet de choisir si l'horloge de sortie du pré diviseur doit être synchrone avec l'horloge du microcontrôleur. Dans le cas où l'on choisit l'horloge interne **Fosc/4**, il n'est pas nécessaire de la synchroniser.
Le bit **TMR1ON** active ou désactive le **timer 1**. Si ce bit est à **1** alors le **timer 1** est en fonctionnement et les registres **TMR1H :TMR1L** sont incrémentés à chaque coup d'horloge.
Quand le contenu du **timer 1** passe de **FFFF** à **0000** le bit **TMR1IF** passe à **1** pour signaler un débordement, de plus si le bit **TMR1IE** est à **1** alors une interruption **timer 1** est déclenchée.

Remarque importante : Lorsque le bit **TMR1IF** passe à **1** lors du passage de la valeur **FFFF** à **0000** des registres **TMR1H :TMR1L**, il doit être remis à **0** de façon logicielle par une instruction du type : **bcf PIR1, TMR1IF**

10.2.3) Configuration et registres associés :

La configuration du **timer 1** passe par les registres : **PIR1** (adresse **0Ch**), **PIE1** (adresse **8Ch page 1**) le registres **TMR1L** et **TMR1H** (adresses **0Eh** et **0Fh**), **T1CON** (adresse **10h page 0**) et **INTCON** (adresse **0Bh** : toutes les pages).

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 Unimplemented: Read as '0'

bit 5-4 T1CKPS1:T1CKPS0: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 T1OSCEN: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 T1SYNC: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 TMR1CS: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 TMR1ON: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

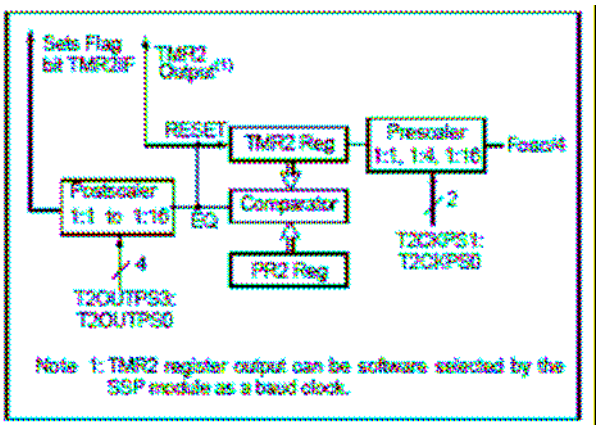
Note1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

10.3. Le timer 2.

10.3.1) Présentation :

C'est un **timer 8** bits, son horloge ne peut être que l'horloge interne divisée par **4** (**Fosc/4**)

TIMER 2 BLOCK DIAGRAM



10.3.2) Fonctionnement :

Il est incrémenté par l'horloge interne (**Fosc/4**) pré divisée ou non. Les bits **T2CKPS1** et **T2KPS0** permettent de choisir la valeur de la pré division (**1,4 ou 16**).

Le contenu du registre incrémenté **TMR2** et il est comparé au registre **PR2**, si ces deux registres sont égaux alors une impulsion d'horloge est générée et le contenu de **TMR2** est remis à **00h**. Celle-ci peut servir d'horloge pour piloter les liaisons **I2C** et **SPI** du module **SSP** ou encore être divisée par un post diviseur appelé : **POSTSCALER**.

Son rapport de division peut être de **1** à **16**. Les bits **T2OUTPS0, T2OUTPS1, T2OUTPS2,** et **T2OUTPS3,** permettent de choisir la valeur de la post division **1, 2, 3, 4, 5 …. 16**.

Quand la sortie du post diviseur passe à **1** le bit **TMR2IF** est positionné, celui-ci peut déclencher une interruption si celle-ci a été autorisée (Bit **TMR2IE** à **1** du registre **PIE1**).

Remarque importante : Lorsque le bit **TMR2IF** passe à **1** lorsqu'une impulsion est générée en sortie du **POSTSCALER**, il doit être remis à **0** de façon logicielle par une instruction du type :
bcf PIR1, TMR2IF

10.3.3) Configuration et registres associés :

La configuration du **timer 2** passe par les registres : **PIR1** (adresse **0Ch**), **PIE1** (adresse **8Ch page 1**), **TMR2** (adresse **11h page 0**), **PR2** (adresse **92h page 1**), **T2CON** (adresse **12h page 0**) et **INTCON** (adresse **0Bh** : toutes les pages).

T2CON: TIMER 2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 Unimplemented: Read as '0'

bit 6-3 TOUTPS3:TOUTPS0: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
□
□
□
1111 = 1:16 Postscale

bit 2 TMR2ON: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

10.4. Les modules CCP1 et CCP2 (C.C.P. :Capture Compare Pwm).

Ces deux modules peuvent fonctionner dans l'un des trois modes ci-dessous :

- Mode **capture** (**CAPTURE**) : Ce mode permet en autre d'effectuer des mesures de temps.
- Mode **comparaison** (**COMPARE**) : Ce mode permet en autre de générer des événements périodiques.
- Mode **PWM** (**PULSE WITH MODULATION**): Ce mode permet de générer des signaux dont le rapport cyclique est variable.

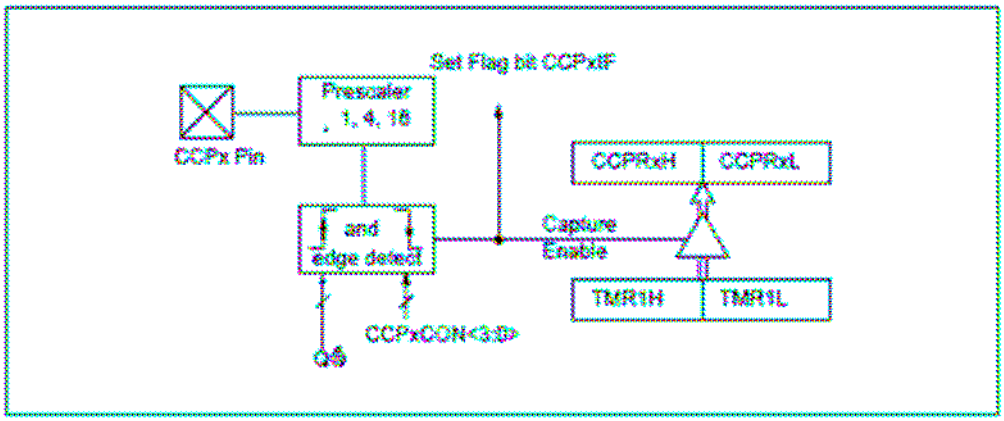
Ces modules sont associés aux broches **RC2/CCP1** et **RC1/T1OSI/CCP2.** Suivant le mode choisit, les **timers 1** ou **2** vont être utilisés. Les modes **Capture** et **Comparaison** utilise le **timer** 1, tandis que le mode **PWM** utilise le **timer** 2.

10.4.1) Le mode CAPTURE :

Il mémorise la valeur du **timer 1** dans les registres **CPP1R1H : CPP1R1L** ou **CPP1R2H : CPP1R2L** quand un événement se produit sur une des broches **CCP1** ou **CCP2**.
Cette mémorisation peut avoir lieu :

- Tous les fronts montants.
- Tous les fronts descendants.
- Tous les **4** fronts montants.
- Tous les **16** fronts montants.

Capture Mode Operation Block Diagram



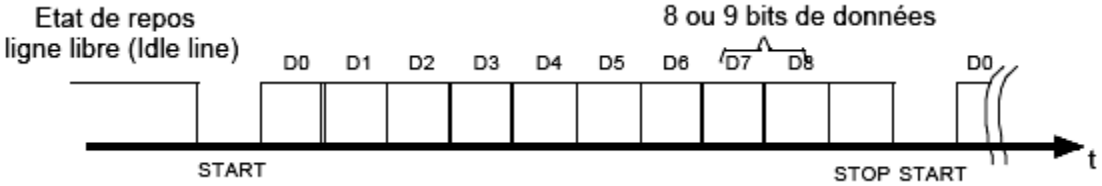
11. La liaison série USART ou SCI (serial communication interface).

11.1. Présentation :

Remarque : La liaison **USART** du **PIC** peut fonctionner soit en mode synchrone ou asynchrone, seul le mode asynchrone sera étudié.

La liaison série **SCI** est une interface série asynchrone de type **START** / **STOP**. Elle permet d'effectuer des communications avec d'autres systèmes ou objets techniques sur de longues distances (quelques mètres à quelques kilomètres).

Elle dispose des fonctionnalités suivantes:
- Fonctionnement en Full Duplex, c'est à dire émission et réception de données en même temps.
- Transmission et réception de données (compatibles avec la norme **RS232** en utilisant une fonction d'adaptation de niveaux).
- Contrôle des erreurs de transmission et de réception.
- Transmission sur 8 ou 9 bits.
- Mode réveil automatique lors de la réception de signaux valides.
- 4 Sources d'interruptions possibles.

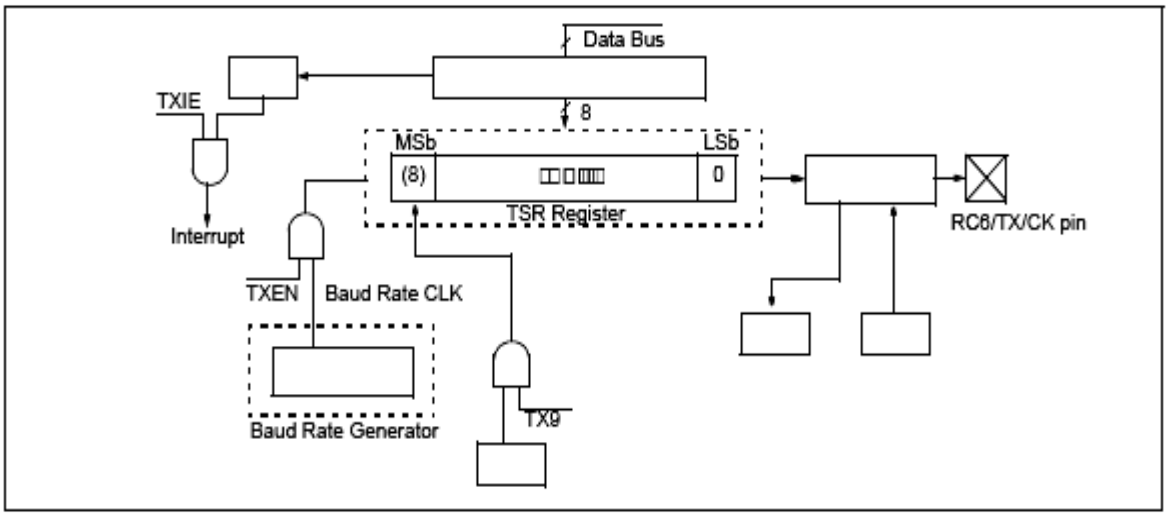


Cette interface est composée de 3 fonctions :
- La transmission.
- La réception.
- Le générateur d'horloge (choix de la vitesse de transmission et de réception).

11.2. La transmission :

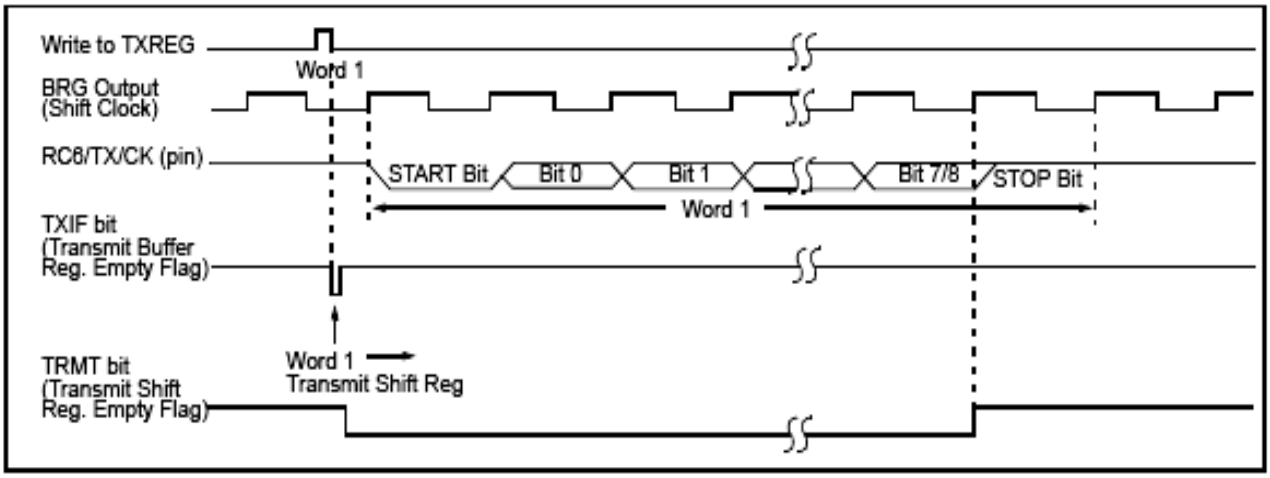
11.2.1) Présentation :

USART TRANSMIT BLOCK DIAGRAM



11.2.2) Fonctionnement :

ASYNCHRONOUS MASTER TRANSMISSION



Cette fonction utilise un registre à décalage pour transmettre les **8** ou **9** bits de l'information du registre **TXREG**.
Pour que cette fonction soit opérationnelle, il faut que la broche **RC6** du **PORTC** soit configurée en sortie, positionner à **1** le bit **SPEN** du registre **RCSTA** et le bit **TXEN** du registre **TXSTA.**
Dans le cas où l'on utilise une transmission sur **9** bits (Bit de parité par exemple), il faut autoriser la transmission sur **9** bits via le bit **TX9** du registre **TXSTA**, la valeur du neuvième bit doit être mise dans le bit **TX9D** du registre **TXSTA**.
Avant de transmettre une information, il faut s'assurer que le registre de transmission soit libre à travers le bit **TXIF** (**1** libre et **0** occupée). Le bit **TMRT** du registre **TXSTA** indique si la transmission est complètement terminée (**1** terminée et **0** occupée**).**

11.2.3) Configuration et registres associés :

La configuration de la transmission de la **SCI** passe par les registres : **PIR1** (adresse **0Ch**), **PIE1** (adresse **8Ch page 1**), **RCSTA** (adresse **18h page 0**), **TXREG** (adresse **19h page 0**), **TXSTA** (adresse **98h page 1**), **SPBRG** (adresse **99h page 1**) et **INTCON** (adresse **0Bh** : toutes les pages).

REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE							0000 000x	0000 000u
0Ch	PIR1									0000 0000	0000 0000
18h	RCSTA									0000 -00x	0000 -00x
19h	TXREG									0000 0000	0000 0000
8Ch	PIE1									0000 0000	0000 0000
98h	TXSTA									0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.

Note1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

Le registre le plus important pour la transmission c'est le registre **TXSTA**.

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 CSRC: Clock Source Select bit
Asynchronous mode:
Don't care
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 TX9: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 TXEN: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in SYNC mode.

bit 4 SYNC: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 Unimplemented: Read as '0'

bit 2 BRGH: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode

bit 1 TRMT: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

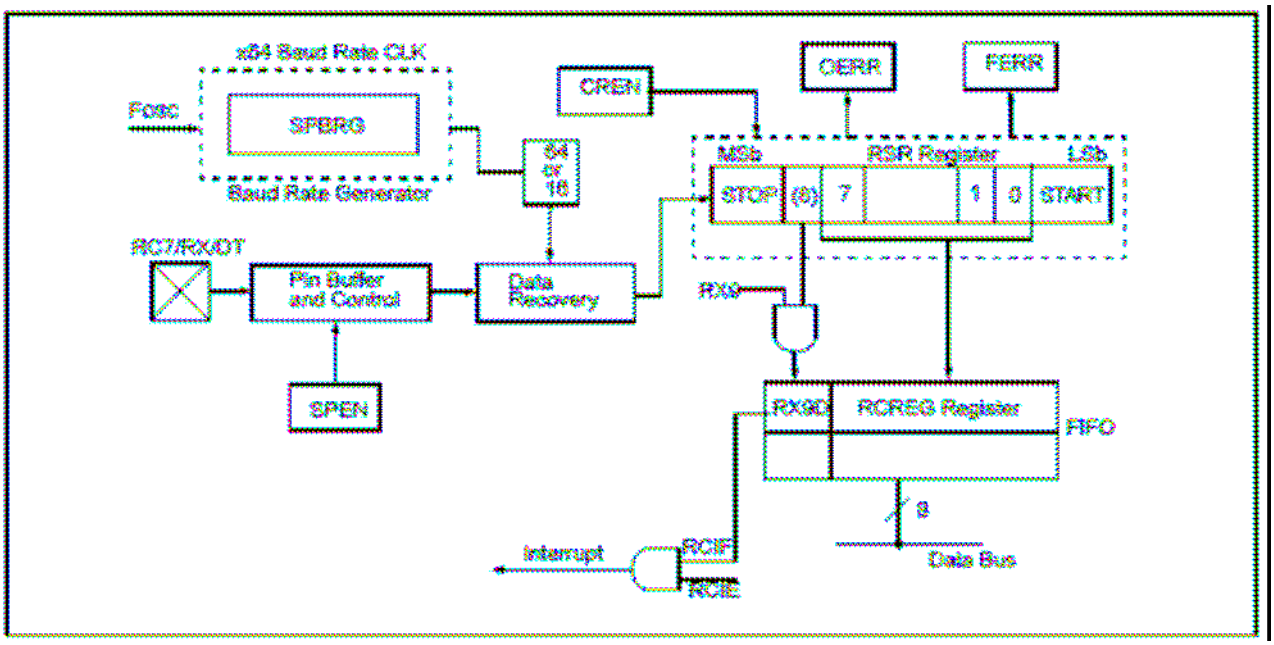
bit 0 TX9D: 9th bit of Transmit Data, can be parity bit

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

11.3. La réception :

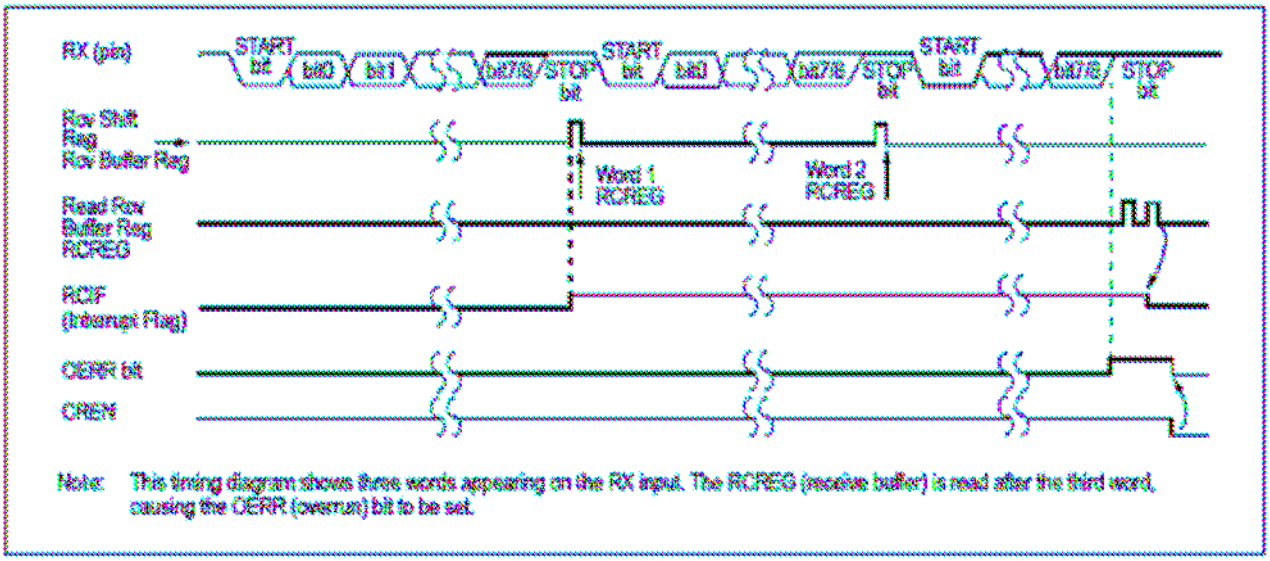
11.3.1) Présentation :

USART RECEIVE BLOCK DIAGRAM



11.3.2) Fonctionnement :

ASYNCHRONOUS RECEPTION



Cette fonction utilise un registre **RSR** à décalage pour les **8** ou **9** bits de l'information à recevoir, une fois la réception terminée la valeur est stockée dans le registre **RCREG**.

Pour que cette fonction soit opérationnelle, il faut que la broche **RC7** du **PORTC** soit configurée en entrée et positionnée à **1** le bit **SPEN** du registre **RCSTA.**

Dans le cas ou l'on utilise une réception sur **9** bits, il faut autoriser la réception sur **9** bits via le bit **RX9** du registre **RCSTA**, la valeur du neuvième bit est récupérée dans le bit **RX9D** du registre **RCSTA**.

Avant de lire une information dans le registre **RCREG**, il faut s'assurer que l'information est bien reçue en testant le bit **RCIF** (**1** Donnée reçue), ce bit est remis à **0** lors de la lecture du registre **RCREG**.
Les bits **FERR** et **OERR** peuvent indiquer respectivement une erreur de format et une erreur **« over run »**.

11.3.3) Configuration et registres associés :

La configuration de la transmission de la **SCI** passe par les registres : **PIR1** (adresse **0Ch**), **PIE1** (adresse **8Ch page 1**), **RCSTA** (adresse **18h page 0**), **RCREG** (adresse **1Ah page 0**), **TXSTA** (adresse **98h page 1**), **SPBRG** (adresse **99h page 1**) et **INTCON** (adresse **0Bh** : toutes les pages).

REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.
Note1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

Le registre le plus important pour la transmission c'est le registre **RCSTA**.

RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 SPEN: Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 RX9: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 SREN: Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode - master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode - slave:
Don't care

bit 4 CREN: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 ADDEN: Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as parity bit

bit 2 FERR: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 OERR: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 RX9D: 9th bit of Received Data (can be parity bit, but must be calculated by user firmware)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

11.4. Le générateur d'horloge:

11.4.1 Présentation et fonctionnement:

C'est lui qui fixe la vitesse de la réception et transmission de **l'USART**.
Il faut utiliser la formule ci-dessous pour calculer la valeur à mettre dans le registre **SPBRG**. Si la fréquence du quartz du microcontrôleur est supérieure à **10MHz**, il est conseillé de positionner le bit **BGRH** à **1**.

BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

X = value in SPBRG (0 to 255)

Tableaux de vitesses :

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3,6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

11.4.2) Configuration et registres associés :

La configuration du générateur d'horloge de la **SCI** passe par les registres : **TXSTA** (adresse **98h page 1**), **RCSTA** (adresse **18h page 0**) et **SPBRG** (adresse **99h page 1**).

REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend : x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

12. Les interruptions.

12.1. Présentation.

Le **µC** dispose de plusieurs sources d'interruptions.

- Une interruption externe, action sur la broche **INT/RB0**.
- Débordement du **TIMER0**.
- Changement d'état logique sur une des broches du **PORTB (RB4 à RB7)**.
- Une interruption d'un des périphériques (**PEIE**).
  - Fin de programmation d'une case mémoire de l'**EEPROM**.
  - Changement d'état sur le **PORTD** (**PSPIE**).
  - Fin de conversion analogique numérique (**ADIE**).
  - Réception d'une information sur la liaison série (**RCIE**).
  - Fin d'émission d'une information sur la liaison série (**TXIE**).
  - Interruption **SPI** ou **I2C** du module **MSSP** (**SSPIE**).
  - Interruption du registre de capture et/ou de comparaison 1 (**CCPI1E**).
  - Interruption du registre de capture et/ou de comparaison 2 (**CCPI2E**).
  - Débordement du **TIMER1** (**TMR1E**).
  - Débordement du **TIMER2** (**TMR2E**).
  - Collision de **BUS** (**BCLIE**)

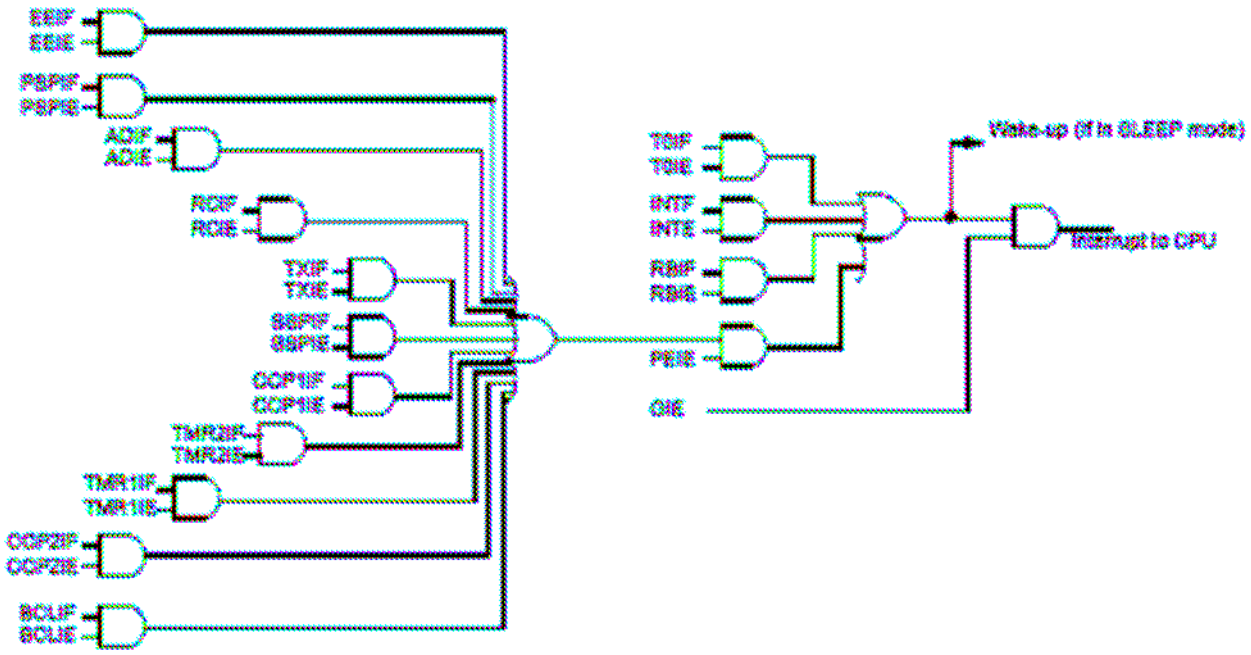
12.2. Fonctionnement.

Lors d'un événement dans un ou plusieurs des circuits périphériques (**ADC, EEPROM, USART-SCI, MSSP-I2C-SPI, TIMER1, TIMER2**) comme par exemple : la fin de conversion, la fin de programmation d'un octet dans **l'EEPROM**, la réception d'une information, la détection d'un front, etc… et si le bit de l'interruption concernée a été autorisée (**EEIE, PSPIE, ADIE, RCIE, TXIE, SSPIE, CCP1IE, TMR2IE, TMR1IE, CCP2IE ou BCLIE : Registres PIE1 et PIE2**) alors une interruption périphérique est déclenchée. Pour que celle-ci soit prise en compte il faut que le bit d'autorisation des interruptions périphériques soit positionné à **1** (**PEIE**) ainsi que le bit **GIE** d'autorisation globale des interruptions du registre **INTCON**.

Pour qu'une interruption du type **TIMER0** ou **INT/RB0** ou **PORTB** soit prise en compte il suffit que le bit local d'autorisation d'interruption soit positionné à **1** (**T0IE** ou **INTE** ou **RBIE**) ainsi que le bit **GIE** d'autorisation globale des interruptions du registre **INTCON**.

Dans ces conditions le programme en cours d'exécution est interrompu et le microcontrôleur exécute le programme d'interruption à partir de l'adresse **0x0004**. Au début de celui-ci il faut que le logiciel vérifie quel périphérique a déclenché l'interruption.

INTERRUPT LOGIC

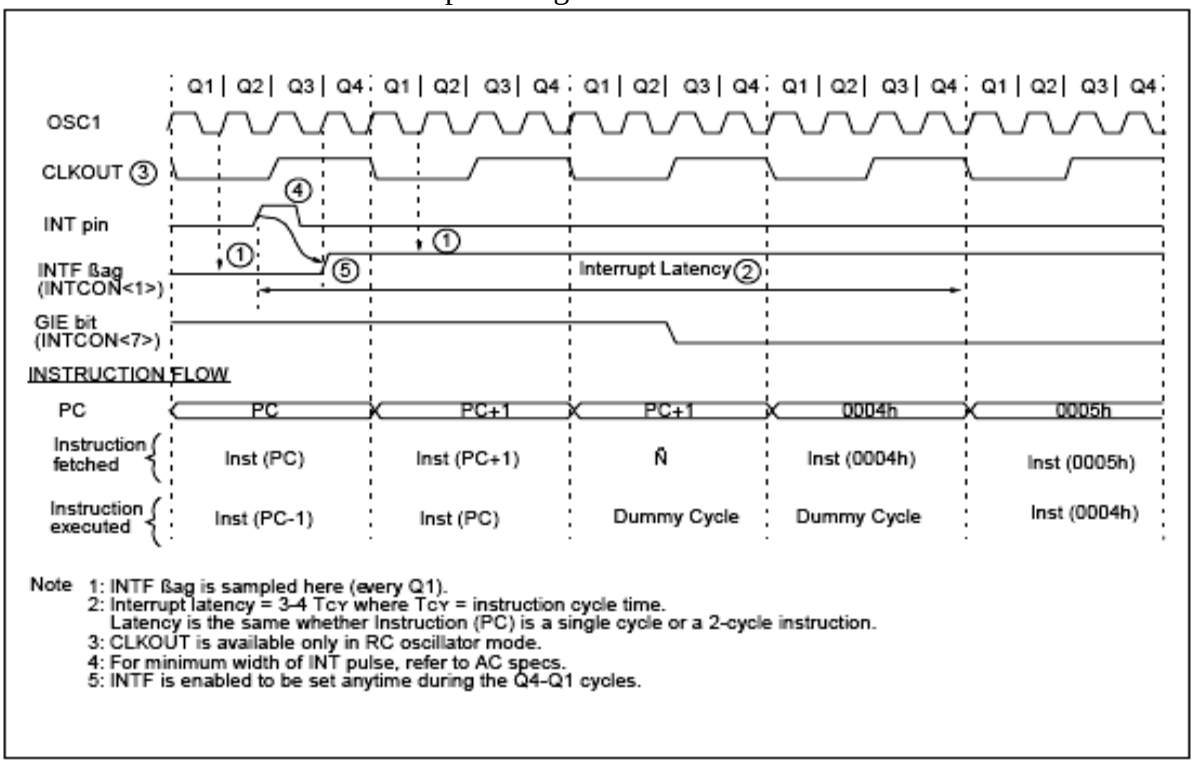


Device	T0IF	INTF	RBIF	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	EEIF	BCLIF	CCP2IF
PIC16F876/873	Yes	Yes	Yes	—	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PIC16F877/874	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

12.3. Déroulement d'une interruption.

Chronogramme de la prise en compte d'une interruption.

INT Pin and Other External Interrupt Timing



Note 1: INTF flag is sampled here (every Q1).
2: Interrupt latency = 3-4 TCY where TCY = instruction cycle time.
Latency is the same whether Instruction (PC) is a single cycle or a 2-cycle instruction.
3: CLKOUT is available only in RC oscillator mode.
4: For minimum width of INT pulse, refer to AC specs.
5: INTF is enabled to be set anytime during the Q4-Q1 cycles.

Au début d'une interruption le sous programme d'interruption doit sauvegarder le contexte et le restituer à la fin, c'est à dire les valeurs des registres **W, PCLATCH** et **STATUS**.
Cela permet au processus interrompu de retrouver ses registres intacts.
Pour respecter ce principe il faut ajouter au début du sous programme d'interruption quelques instructions pour sauvegarder les registres **W, PCLATCH** et **STATUS**. A la fin du sous programme on ajoute des instructions pour restaurer ces valeurs.

SAVING STATUS, W, AND PCLATH REGISTERS IN RAM

```
MOVWF   W_TEMP          ;Copy W to TEMP register
SWAPF   STATUS,W        ;Swap status to be saved into W
CLRF    STATUS          ;bank 0, regardless of current bank, Clears IRP,RP1,      RP0
MOVWF   STATUS_TEMP     ;Save status to bank zero STATUS_TEMP register
MOVF    PCLATH, W       ;Only required if using pages 1, 2 and/or 3
MOVWF   PCLATH_TEMP     ;Save PCLATH into W
CLRF    PCLATH          ;Page zero, regardless of current page
;
: (ISR)                 ;(Insert user code here)
;
MOVF    PCLATH_TEMP, W  ;Restore PCLATH
MOVWF   PCLATH          ;Move W into PCLATH
SWAPF   STATUS_TEMP,W   ;Swap STATUS_TEMP register into W
                        ;(sets bank to original state)
MOVWF   STATUS          ;Move W into STATUS register
SWAPF   W_TEMP,F        ;Swap W_TEMP
SWAPF   W_TEMP,W        ;Swap W_TEMP into W
```

12.4 Configuration et registres associés :

- Le registre **OPTION** permet de choisir le type de front pour l'interruption **INT/RB0**.

OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 6 INTEDG: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

- Le registre **INTCON** permet d'autoriser les interruptions globales (**GIE**), les interruptions des périphériques (**PEIE**), L'interruption **TIMER0** (**T0IE**), l'interruption extérieure (INT/RB0), l'interruption de changement d'état du **PORTB** (**RBIE**) et les indicateurs associés des interruptions (**TIMER0**, **INT/RB0** et du changement d'état du **PORTB** : **RBIF**).

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 GIE: Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 PEIE: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 T0IE: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 INTE: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 RBIE: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 T0IF: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 INTF: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 RBIF: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

- Les registres d'autorisations des interruptions périphériques **PIE1** et **PIE2**.

PIE1 REGISTER (ADDRESS 8Ch)

bit 7 PSPIE(1): Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 ADIE: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 RCIE: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 TXIE: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 SSPIE: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 CCP1IE: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 TMR2IE: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 TMR1IE: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

PIE2 REGISTER (ADDRESS 8Dh)

bit 7 Unimplemented: Read as '0'
bit 6 Reserved: Always maintain this bit clear
bit 5 Unimplemented: Read as '0'
bit 4 EEIE: EEPROM Write Operation Interrupt Enable
1 = Enable EE Write Interrupt
0 = Disable EE Write Interrupt
bit 3 BCLIE: Bus Collision Interrupt Enable
1 = Enable Bus Collision Interrupt
0 = Disable Bus Collision Interrupt
bit 2-1 Unimplemented: Read as '0'
bit 0 CCP2IE: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

- Le registre des indicateurs d'événements **PIR1**.

PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 PSPIF(1): Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 ADIF: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 RCIF: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 TXIF: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 SSPIF: Synchronous Serial Port (SSP) Interrupt Flag
1 = The SSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI
  - A transmission/reception has taken place.
- I^2C Slave
  - A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated START condition was completed by the SSP module.
  - The initiated STOP condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A START condition occurred while the SSP module was idle (Multi-Master system).
  - A STOP condition occurred while the SSP module was idle (Multi-Master system).

0 = No SSP interrupt condition has occurred.

bit 2 CCP1IF: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 TMR2IF: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 TMR1IF: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: PSPIF is reserved on PIC16F873/876 devices; always maintain this bit clear.

Chapitre 4

Programmation des PICs

1. Ecrire un programme PIC (16F8XX)

Pour écrire un programme PIC ils sont nécessaires :

- un éditeur de texte
- un assembleur
- un compilateur
- un simulateur pour tester le programme sur le micro

Tout cela est mis à la disposition par Microchip. Cela s'appelle MPLAB, téléchargez-le et installez-le sur votre micro.

MPASM - L'assembleur

MPASM fait partie de MPLAB, c'est l'assembleur qui permet d'écrire le programme PIC.

Le meilleur moyen d'apprendre est d'analyser un programme simple, tel que clignotement d'une LED. Il est nécessaire de télécharger le guide de MPASM de chez Microchip et de l'imprimer.

Le set d'instructions pour PIC 16F8XX. est donné sur annexe 1

Pour vous donner un aperçu voici un petit bout de programme qui allume une LED lorsqu'on appui sur le bouton poussoir. La LED est connectée à la **voie 0** et le B.P. à la **voie 1** du port B.

toto btfss PORTB,1

b toto

bsf PORTB,0

toto est une étiquette

btfss est une instruction de test de bit (bit test file skip if set), elle test le bit 1 du registre PORTB, si ce bit est à 1 (Bouton poussoir appuyé) elle saute l'instruction suivante, c'est à dire qu'elle arrive à l'instruction *bsf,* si le bit est à 0 elle exécute l'instruction suivante

b toto branchement à l'étiquette *toto,* le programme boucle

bsf PORTB,0 bit set file, c'est à dire qu'elle positionne à 1 la voie 0 du PORTB (ce qui allume la LED)

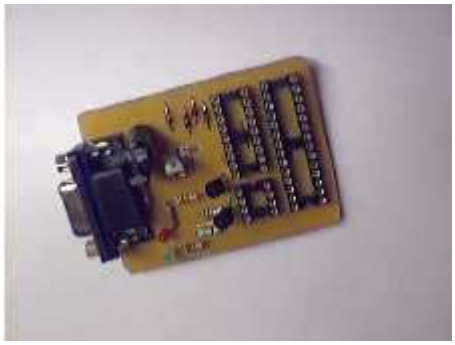
Le programme complet (la LED change d'état à chaque appui sur le BP)

```
;*****************************************************************
; bp_led - Commande d'une LED à l'aide d'un BP
;         La LED change d'état à chaque appui sur le BP
;
; RB
; 0      LED (cathode sur RB0, anode via résistance 1.2K au +5V)
; 1     BP  (contact à la masse via résistance 1K , brancher une résistance
;          10K de polarisation entre RB1 et +5V)
;
;*****************************************************************

        list p=16f84,f=inhx8m

      __config  B'00000000000001'    ; WDT Enabled, RC Oscillator
;
      #include "p16f84.inc"

;**** Constantes *********
led     equ       0
bp      equ       1

;
        org       0
        goto       Start
;


;
        org       10
Start
        clrf       PORTB
        bsf        STATUS,RP0     ; select bank1
        movlw    B'11111110'
        movwf    TRISB              ; seul rb0 en sortie
        bcf        STATUS,RP0
        bsf        PORTB,led         ; éteint la led

        b       Main

;************** Main **********************************

Main
        btfsc      PORTB,bp        ; saute l'instruction suivante si bp appuyé
        b          $-1             ; saut à l'instruction précédente
        comf       PORTB,F         ; bascule etat de la led
        btfss      PORTB,bp        ; saute l'instruction suivante si bp relaché
        b          $-1             ; saut à l'instruction précédente
        b          Main

        end
```

Le programmateur

Une fois le programme compilé, il faut le transférer dans la mémoire du microcontrôleur. Pour cela il vous faut:

Une petite interface matérielle



Ci-dessus, l'interface de programmation par le port série du PC. C'est une des meilleures interfaces de programmation, ne nécessite pas d'alimentation externe.
Ses caractéristiques:

- tension d'alimentation fournie par le port série : 4.5V
- tension de programmation : 13.5V
- connexion au port série par un câble droit 9 points mâle / femelle
- permet de programmer les PIC 12Cxx, 16Cxx, 16Fxx, 16Fxxx etc.... ainsi que les EEPROM 24Cxx

Le kit comprend:

- l'interface de programmation ci-dessus
- un cordon de connexion au port série du PC
- le logiciel de programmation ICPROG sur disquette
- une documentation en français

Un logiciel

Un logiciel qui assure le transfert des données entre le PC et le microcontrôleur (il est fourni avec le programmateur) ou bien il se trouve sur le Web.

Il est conseillé ICPROG , c'est le meilleur logiciel de programmation de PIC à l'heure actuelle, et il a l'avantage d'être disponible en freeware.

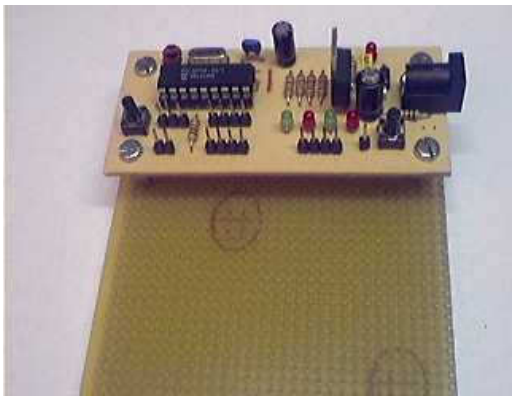
Un premier programme : **Clignotement d'une LED**

2 approches différentes:

- un programme qui utilise le timer : led_tim.zip (ANNEXE 2)
  - Le timer génère une interruption toutes les 256µs. Dans la routine d'interruption, l'on décrémente la variable TIME.
  - Dans le programme principal, on change l'état du PORTB (sur lequel est connectée une LED) lorsque TIME=0, et on recommence le cycle.

- un autre qui utilise le chien de garde : led_wdt.zip (ANNEXE 3)

Carte d'expérimentation



Pour tester de nouveaux montages à base de PIC, il peut s'avérer utile de disposer d'une platine de développement. Pour ma part j'utilise la carte d'expérimentation pour PIC16F84 ci-dessus. Ce système permet de tester rapidement un PIC que vous avez réussi à programmer.

La platine dispose de sa propre tension d'alimentation 5V, d'un oscillateur quartz 4, 10 ou 20 Mhz, d'une zone pastillée permettant d'implanter des composants d'E/S, de 4 LED et un bouton poussoir pouvant être reliés aux E/S par des straps.

2. Programmation en utilisant « EDITALGO »

A. Implanter un algorigramme avec le logiciel EditAlgo

Quelques règles

Le logiciel peut fonctionner de deux manières différentes.

L'option Programme permet d'écrire un petit programme monobloc, avec des variables globales.

L'option Procédure permet d'analyser le contenu d'une procédure et une seule. Les variables utilisées sont alors globales, locales ou paramètres d'entrée sortie ou paramètres d'entrée.

Dans les deux cas, le bloc d'instructions commence par DEBUT et se termine par FIN

Chaque nouvelle action ou expression, lorsqu'elle n'est pas encore formalisée, est notée entre <...>.

Les boucles ou itérations doivent commencer par un niveau général <BOUCLE> et terminer par <FinBOUCLE>. Le type de la boucle ne pourra être donné qu'au niveau suivant. Dans la mesure ou BOUCLE n'est pas une primitive du processeur, il figure entre '<' et '>' comme toute autre action non primitive.

Ainsi dans l'ordre on doit :

- Décrire l'action
- Si nécessaire, préciser que cette action est incluse dans une boucle (on commence par l'intérieur)
- Préciser le type de boucle
- Préciser les bornes ou les conditions.

Définir la fonction de l’algorithme.

Exemple a : On souhaite faire clignoter une diode à la période 2s.

Exemple b : On souhaite compter les impulsions produite par le mouvement d’une roue d’entée entre les bras d’un photocoupleur en fourche (codeur incrémental)

Exemple c : On souhaite réaliser un dé magique : l’utilisateur appuie sur un bouton un instant puis le lâche. Aussitôt un résultat s’affiche sur un afficheur 7 segments.

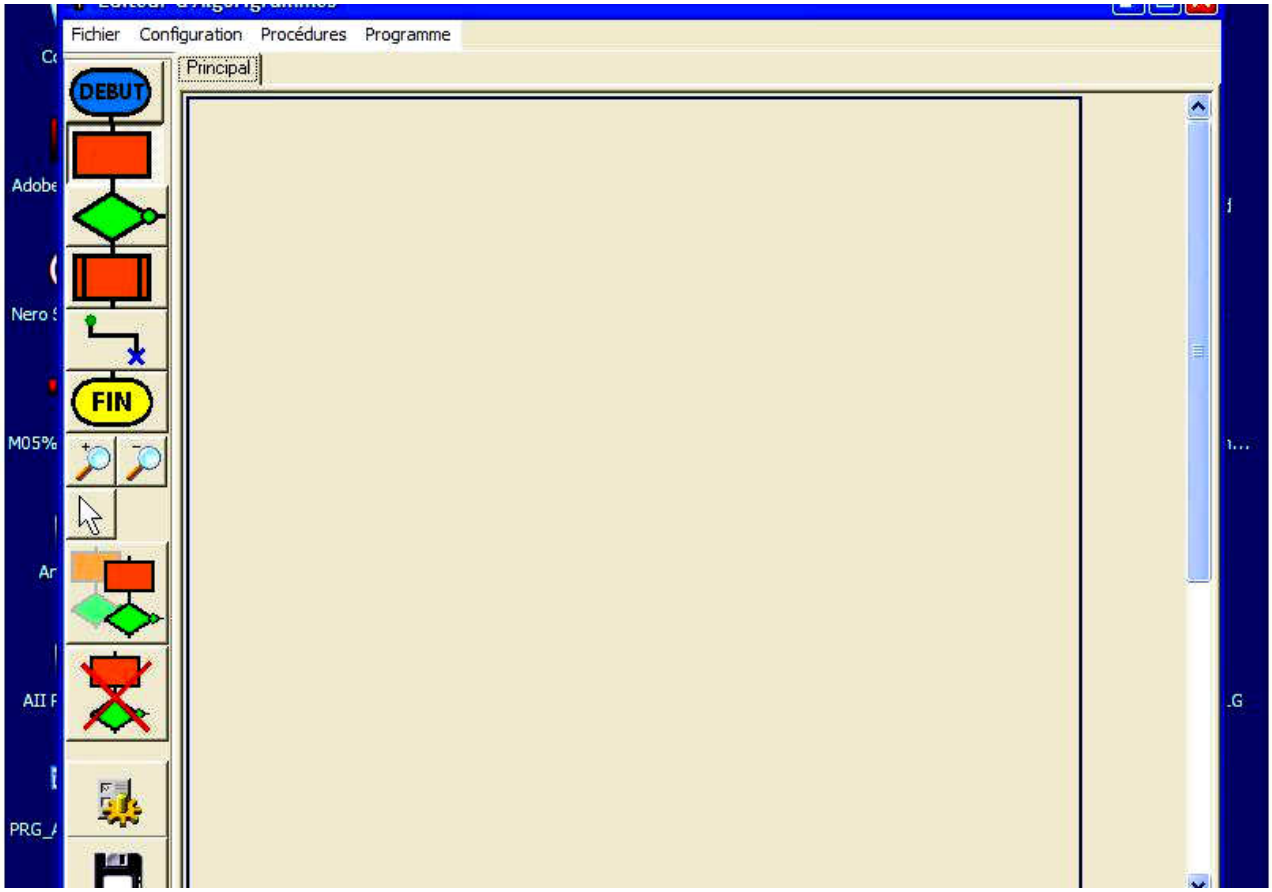
Définir les entrées sorties du microcontrôleur et les variables internes utiles.

Exemple a: Une seule sortie, B0 qui sera relié directement à la diode. B0=1 : Diode allumée. B0=0 : Diode éteinte.

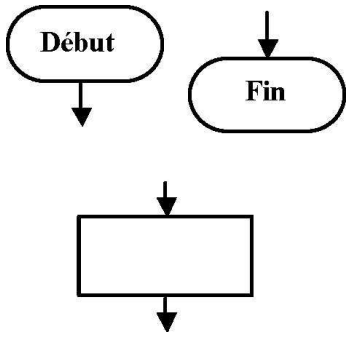
Exemple b : A0 en entrée, reçoit l'information du capteur. Bien que ce ne soit pas demandé on peut sortir sur le port B la valeur courante du comptage. Il s'agit donc d'un nombre de 8 bits. On a intérêt a stocker la valeur courante dans une variable interne COMPTEUR.

Exemple c : A0 en entrée reçoit l'information sur l'état du bouton : 1 pour appuyé et 0 pour pas appuyé. L'afficheur 7 segments peut être piloter via un transcodeur BCD 7 segments. Il faut donc seulement 4 bits pour indiquer la valeur du dé de 0 à 6. On utilisera pour cela les bits B4 (poids fort) à B0 du Port B. On a intérêt à utiliser un variable interne DE pour stocker la valeur courante du dé.

Dessiner l'algorigramme sur feuille.

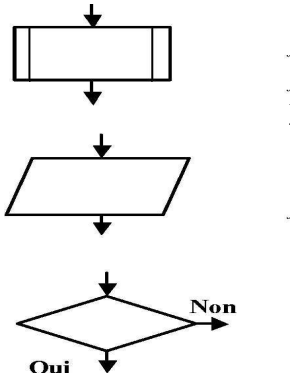


La représentation des algorigramme est normalisée. Elle comporte différents éléments de base. Nous n'utiliserons que les éléments suivants.



Entrée et sortie d'un algorigramme. Le programme commence toujours par le Début. Un algorigramme a forcément un début et un seul. Par contre il peut n'avoir aucun élément Fin ou plusieurs. Début et Fin peuvent être utilisés pour définir un sous programme. Ils sont alors les points d'entrée et de sortie du sous programme.

Symbole général pour tout type d'action : faire uns opération arithmétique (+, -) (remarque : * et / ne sont pas prévus dans EditAlgo) ou logique (OU, XOR, NON ? ET) sur des variables, incrémenter un nombre, le déclaer à droite ou a gauche…

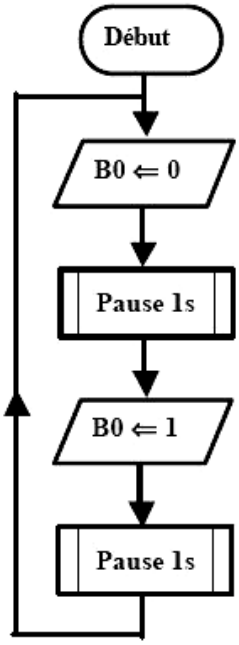


Appel de sous programme. Lors d'une procédure souvent répétée, par exemple utiliser des compteurs comptant 'dans le vide' pour faire passer le temps et faire ainsi une temporisation. Au lieu de réécrire sans arrêt le programme effectuant cette temporisation, on en fait une fonction que l'on appelle quand on en a besoin. Dans EditAlgo cet élément est représenté par le symbole général.

Entrée – Sortie. Dans la mesure du possible, on récupère les informations en entrées, on les stocke dans des variables internes, on fait tous les calculs sur des variables internes, puis on met à jour les sorties en fonction des résultats obtenus. Cet élément représente les lectures en entrée et écriture en sortie. Cependant, dans le cas de variables simples (informations TOR) de telles complications sont inutiles. Dans EditAlgo, ce symbole est représenté par le symbole général.

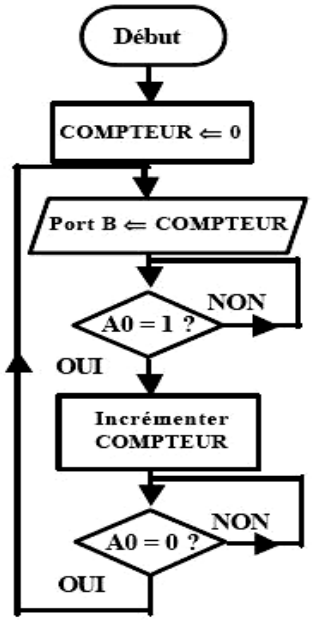
Condition. Cet élément représente un test effectué dont on peut dire que le résultat est vrai ou faux. Le test peut être effectué sur un bit (bit=1 ? ; bit=0 ?) ou sur des nombres (nombre 1<nombre2 ? nombre1=nombre 2 ?)

Exemple a :



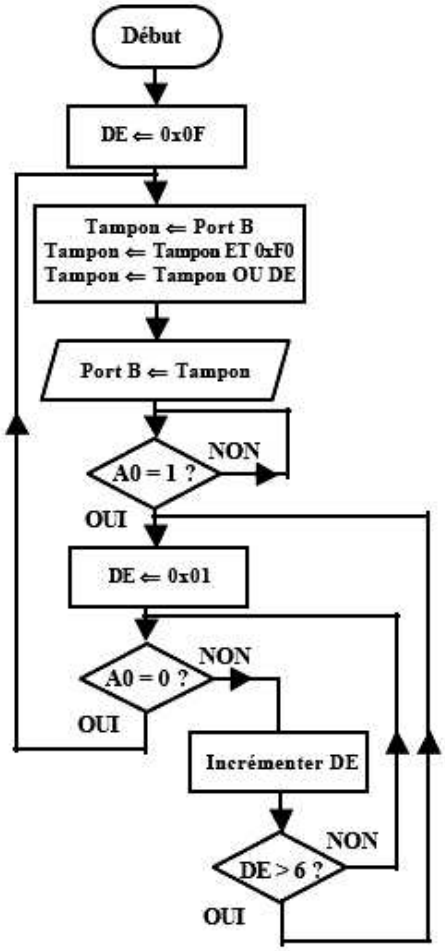
Cet exemple ne pose pas de difficultés. Remarquez seulement la notation ‘ B0 <-- 0’ qui signifie que l’on place la valeur 0 dans B0.

Exemple b *:*



Ici, remarquez les deux blocs de condition. Le premier attend que A0 passe à 1 et le second attend que A0 repasse à 0. On détecte ainsi les impulsions de A0. Il faut cependant être sûr que le microcontrôleur ait le temps d’incrémenter avant l’impulsion suivante.

Exemple c :

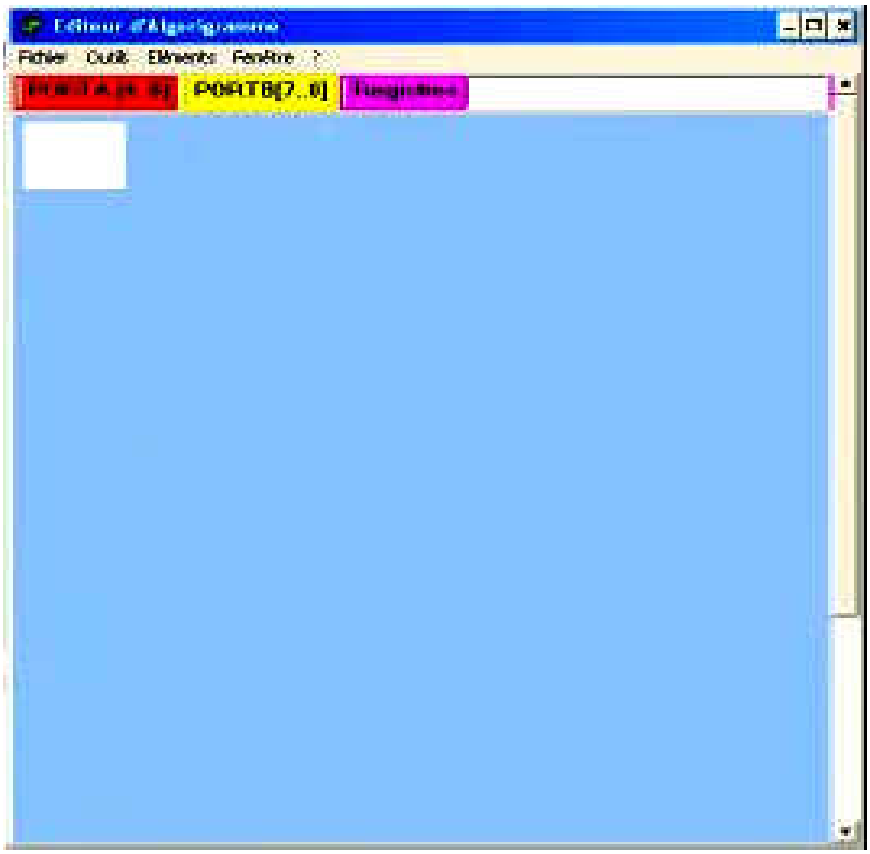


Cet algorigramme est plus complexe. -On place 0x0F dans DE. C'est une notation hexadécimale, elle est utilisée dans EditAlgo. Habituez-vous à n'utiliser que de l'hexadécimal lorsque vous programmez le microcontrôleur. -Seuls les 4 bits de poids faibles de DE ont une importance. Ils doivent être recopiés dans les 4 bits de poids faibles du port B si possible sans toucher aux 4 bits de poids fort du Port B. C'est la raison d'être de la variable Tampon. Tampon recopie Port B. Le ET 0xF0 est ce qu'on appelle un masque qui place les 4 bits de poids faibles de Tampon à 0 et laisse intact les 4 bits de poids fort. Ensuite le OU permet de recopier les 4 bits de poids faibles de DE sur les 4 bits de poids faibles de Tampon. Les 4 bits de poids forts de DE étant toujours à 0, cela ne pose pas de problème. Au démarrage, on envoie donc 1111 sur l'afficheur, il n'affiche donc rien, c'est ce qu'on veut. -Remarquez que l'on a mis plusieurs opérations dans un même bloc. On peut se le permettre pour simplifier l'organigramme à condition que cela ne nuise pas à la lisibilité. -On attend que l'utilisateur appuie (et donc que A0 passe à 1) -Dès qu'on appuie, DE est initialisé à 0x01 (c'est à dire que le dé est à 1) -Tant que A0 reste à 1 on incrémente la valeur de DE. Si DE devient supérieur à 0x06, on repasse par l'initialisation de DE à 0x01. -Quand l'utilisateur relâche le bouton, on retourne au moment de l'affichage.

Dans ce dé, le hasard est produit par le fait que l'utilisateur ne peut pas maîtriser suffisamment précisément la durée de sa pression pour décider de la valeur du dé.

B. Utilisation du logiciel EditAlgo.

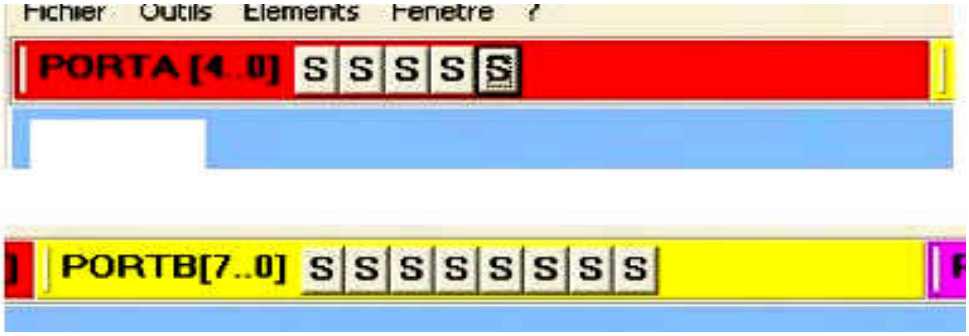
Pour lancer EditAlgo, double-cliquez sur EditAlgo.exe La fenêtre ci-contre s'affiche.



La fenêtre se compose d'une barre de menu ; d'une grille de 8x15 éléments sur laquelle vous pourrez représenter votre algorigramme ; Une fenêtre et une commande en bas de la fenêtre dont la signification sera donnée plus loin.

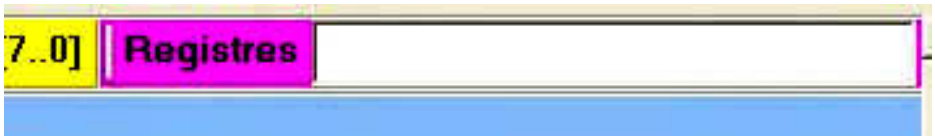
Les étapes suivantes peuvent être effectuées dans n'importe quel ordre.

Première étape : Décider des entrées / sorties.



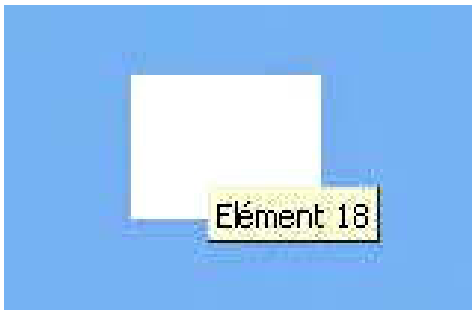
Sur la bande du haut, cliquez sur PortA . Cliquez simplement sur les boutons pour changer les bits du portA en entrée (E) ou en sortie (S) Le bit de poids fort est à gauche. Idem pour le PortB .

Deuxième étape : Définir les variables utilisateur.



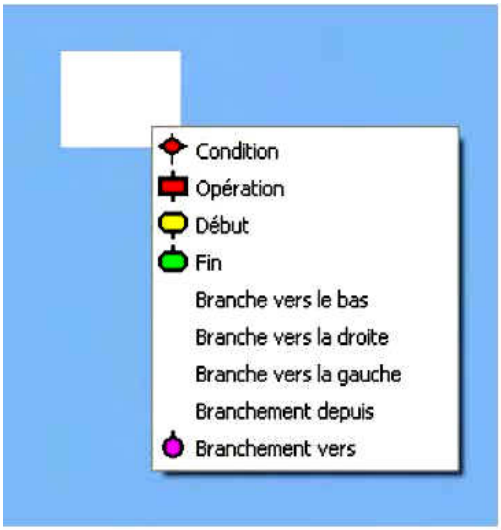
Sur la troisième partie de la bande supérieure, vous pouvez définir des variables utilisateur. Tapez simplement le nom de votre variable, sans espace puis tapez Entrée. Vous pouvez également supprimer des variables. Ne laissez pas de ligne vide entre les variables.

Troisième étape : Dessiner l'algorigramme.

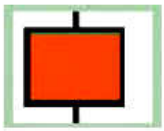


Vous pouvez choisir n'importe quelle case en cliquant dessus avec le bouton gauche de la souris. Chaque cas a un numéro qui apparaît si vous laissez la souris immobile un instant au-dessus.

En cliquant-droit, vous faites apparaître ce menu. Il vous permet de choisir entre les différents types d'éléments possibles. Vous avez le choix entre



Condition

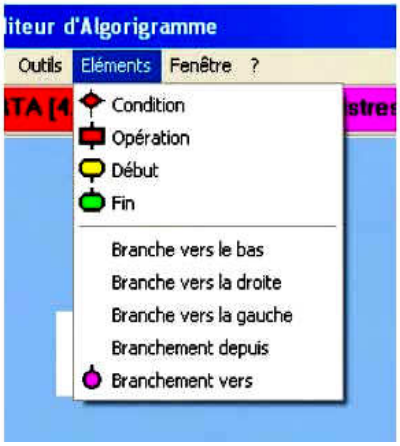


Opération

Début

Fin

Vous avez aussi accès à différents branchements.

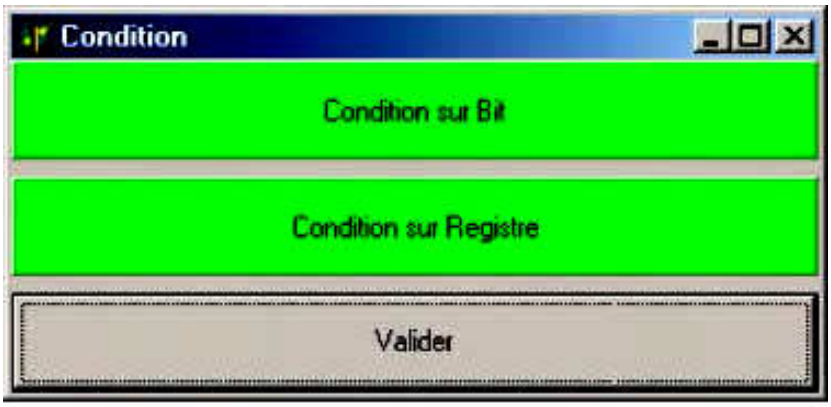


Vous pouvez aussi accéder à ce choix grâce au menu ***Eléments***.

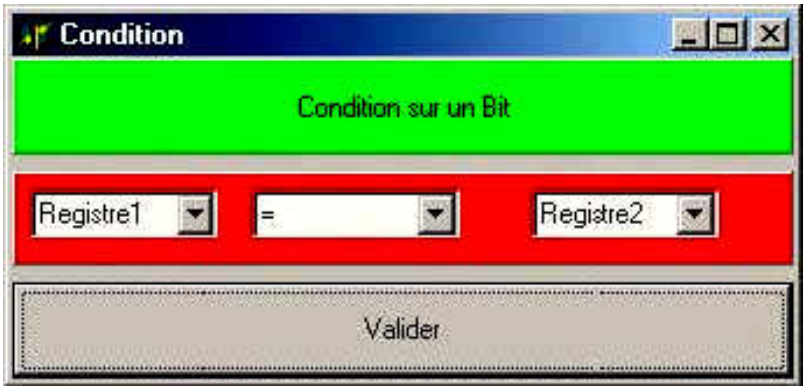
Elément condition.

Le choix de *condition* fait apparaître la fenêtre cicontre. Vous pouvez laisser cette fenêtre ouverte ou pas. Vous pourrez toujours la rouvrir grâce au menu *Fenêtre*.

Quand vous appuyez sur le bouton du haut, les options ci-contre apparaissent. C'est une condition sur un bit. Il faut préciser le nom du registre et le numéro du bit en question. Vous pouvez choisir l'un des registres système (dont Port A et Port B) ou un registre utilisateur. Vous devez préciser quel état vous souhaitez tester : 0 ou 1. Quand vous avez terminé, appuyez *Valider*.



Quand vous appuyez sur le bouton du bas, les options ci-contre apparaissent. C'est une condition sur un registre. Il faut préciser le nom du registre à tester. Vous pouvez choisir l'un des registres système (dont Port A et Port B) ou un registre utilisateur. Vous devez choisir le test (=, <>, <, >) et la valeur de comparaison (à droite), soit un registre soit un nombre. Attention : Si vous entrez un nombre à droite, utilisez une notation hexadécimale. Exemple : pour 28, tapez 0x1C. Quand vous avez terminé, appuyez *Valider*.



Remarque : Chaque fois que vous tapez *Valider*, vous écrasez l'ancienne programmation de l'élément par la nouvelle.

Elément opération.

Le choix d'Opération fait apparaître la fenêtre cicontre. Vous pouvez laisser cette fenêtre ouverte ou pas. Vous pourrez toujours la rouvrir grâce au menu *Fenêtre*.

Très important : Pour gagner de la place, on peut mettre plusieurs opérations en un seul élément. Le bouton *Ajouter* permet d'ajouter des opérations. Si vous avez fait une erreur il faut recommencer avec *Effacer* ou bien utiliser la fenêtre code (voir plus loin)

Le bouton du haut donne accès aux opérations sur un bit, c'est à dire Mise à 1 ou Mise à 0.

Le deuxième bouton permet de faire des opérations à deux opérandes, c'est à dire +, -, ET, OU, XOR. Attention : Si vous entrez un nombre à droite, utilisez une notation hexadécimale. Exemple : pour 28, tapez 0x1C.

Le troisième bouton donne accès à des opérations à une opérande : NOT, OUI, SHIFTL et SHIFTR (décalage à gauche et à droite) Incrémenter, Décrémenter, SWAP (inversion des 4 bits de poids forts avec les 4 bits de poids faibles) Attention : Si vous entrez un nombre à droite, utilisez une notation hexadécimale. Exemple : pour 28, tapez 0x1C. Remarque : Pour affecter une valeur, par exemple x=0x35, il suffit de faire x=OUI 0x35.

Le bouton du bas donne accès à des commandes particulières :

- Pause 0,1s ?? : Permet de faire une pause de ?? · 0,1s où ?? est un nombre hexadécimal (noter 0x..)
- Pause 5ms ?? : Permet de faire une pause de ?? · 5ms où ?? est un nombre hexadécimal (noter 0x..)
- CALL et RETURN sont des instructions assembleur.
- ROUTINE permet d'inclure un sous programme réalisé au préalable : Réalisez le sous programme et enregistrez-le en tant que programme. Le Début et le Fin du

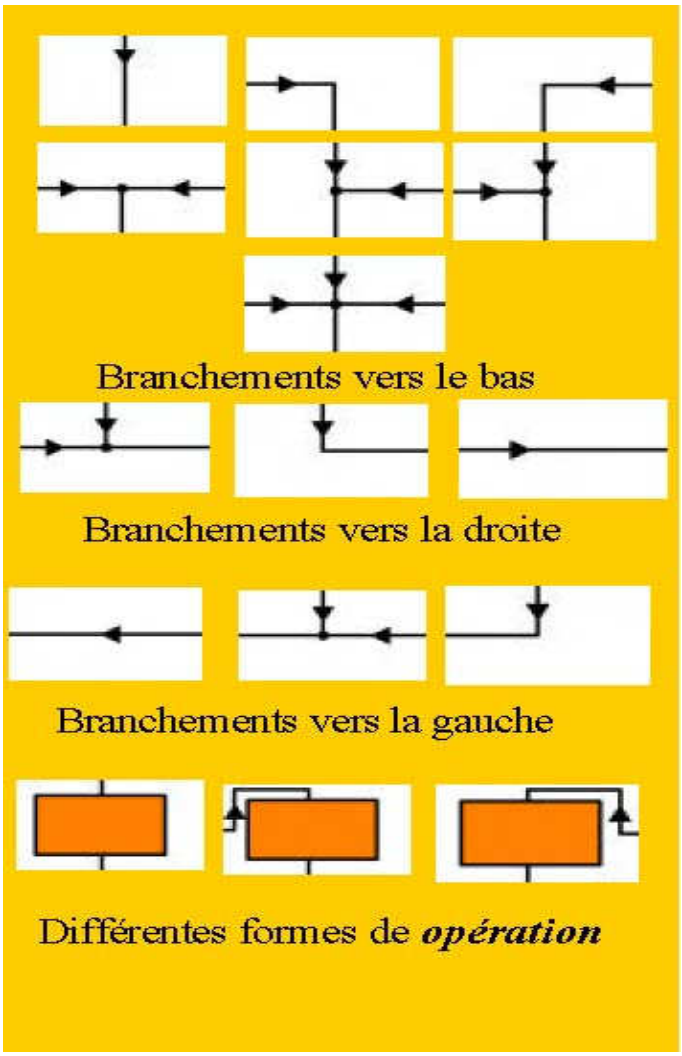
sous programme seront sont entrée et sa sortie. Les entrées/sorties du sous programme doivent être les mêmes que celles du programme principal. Les variables utilisateur du sous programme sont automatiquement ajoutés au programme principal.

Les branchements ordinaires.

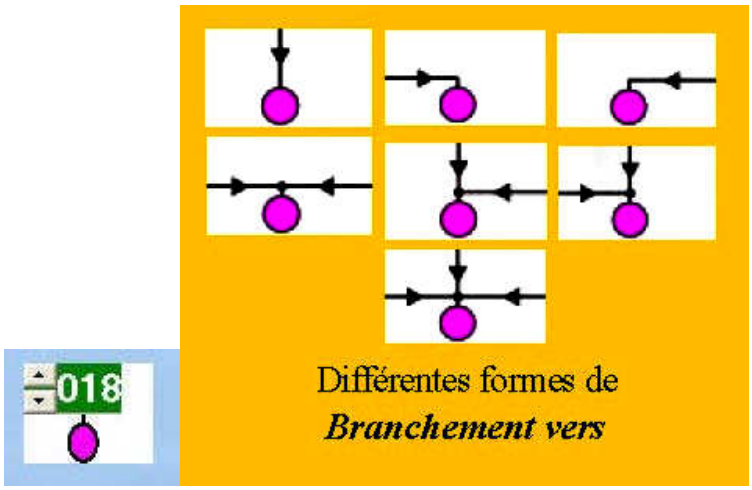
Ils vous permettent de relier entre eux les éléments. Vous devez d'abord sélectionner la direction du branchement (vers le haut, vers la droite, vers le bas)

Quand vous avez sélectionné le branchement, doublecliquez sur l'élément pour lui donner la courbure souhaitée.

Remarque : Vous pouvez aussi double-cliquer sur un élément *opération* pour changer sa forme.



Les branchements vers.



Lorsque vous avez besoin de remonter ou bien que dessiner les branchements complexifie trop l'algorigramme, vous pouvez représenter un saut, c'est à die un branchement vers une destination.

Quand vous avez sélectionné Branchement vers, choisissez le numéro de la cellule de destination. Sélectionnez le numéro de l'élément sur lequel vous souhaitez vous brancher, puis appuyez sur *Valider* . Remarque : Branchez vous directement sur l'élément souhaitez. Inutile de représenter une arrivée pour le saut (élément *Branchement depuis* . Cet élément est inutile)

Remarque 2 : Tout comme pour les branchements ordinaires, vous pouvez double cliquer sur l'élément *Branchement vers* pour lui donner la forme souhaitée.

Modification du code.

A chaque élément correspond une portion de code (c'est à dire de programme) Vous pouvez l'afficher grâce à la commande *Code* du menu *Fenêtre* .

Vous pouvez modifier ce code. Cliquez sur *Modifier*, l'écran devient bleu. Faites vos modifications. Quand vous en avez fini, Cliquez sur *Valider* ou bien *Annuler*.

Remarque : Un fichier résumé est créé automatiquement quand vous enregistrez le programme. Il porte l'extension *.RSU. Pendant la programmation vous pouvez aussi, en laissant le curseur immobile sur un élément, voir un résumé de ce qui a été programmé dans l'élément (le carré représente une tabulation) Par contre, si vous utilisez la fenêtre code pour modifier un élément, la fonction résumé indiquera *'code modifié'* pour cet élément.

Quatrième étape : Fichier *.ASM.

Grâce à la commande Programme du menu Fenêtre , vous pouvez visualiser le programme assembleur qui sera généré. Vous pouvez réactualiser ce code assembleur grâce à la commande Fichier Asm du menu Programme.

Outils ou bien en fermant et en rouvrant la fenêtre

Vous pouvez voir le résultat grâce à la fenêtre programme. Dans le menu *fenêtre* , cliquez sur *Programme* .

Vous pouvez modifier ce code. Cliquez sur *Modifier*, l'écran devient bleu. Faites vos modifications. Quand vous en avez fini, Cliquez sur *Valider* ou bien *Annuler*. Vous pouvez sauvegarder ce fichier en cliquant *Sauvegarder*.

Cinquième étape : Sauvegardes.

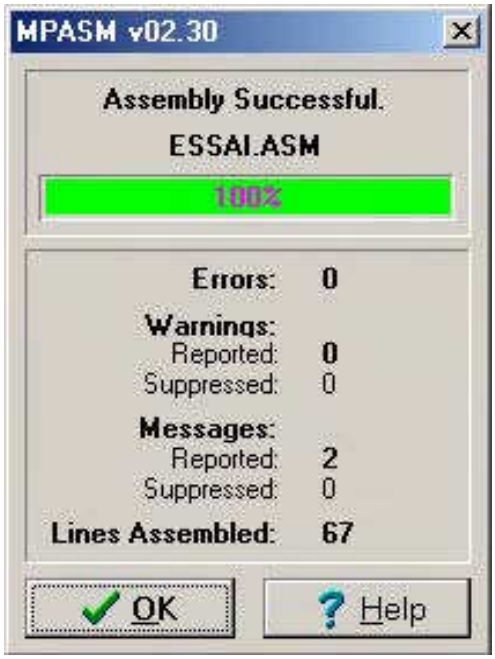
Dans le menu *Fichier* vous devez sauvegarder le fichier *.ASM. Attention : Le fichier *.ASM n'enregistre pas la structure graphique de votre algorigramme. Si vous voulez la conserver, enregistrez-là avec *Enregistrer programme sous…* Vous pourrez ensuite la rouvrir avec *Ouvrir programme* ou même l'utiliser comme sous programme grâce à la commande *Routine* de l'élément *opération*.

Sixième étape : Compilation.

Dans le menu Outil, choisissez la commande MPASMWIN. Il s'agit d'un utilitaire Microchip.

La fenêtre ci-contre apparaît. Cliquez sur *Browse*. Choisissez votre fichier Asm. Cliquez sur *Assembl* .MPASMWIN lance la compilation.

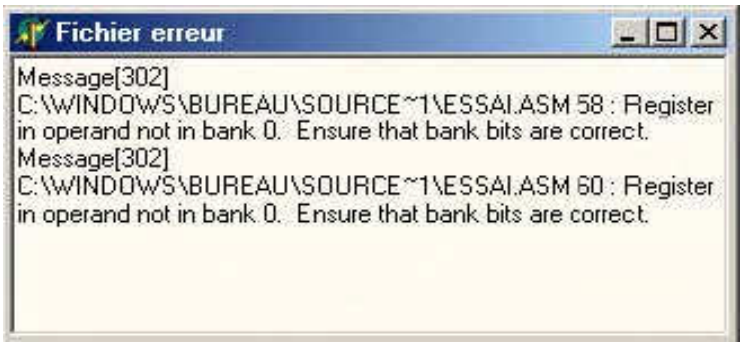
Si vous obtenez ce genre de message, alors votre programme ne contient pas d'erreur. Il a été compilé avec succès sous forme d'un fichier *.HEX. Vous pouvez passer à l'étape suivante.



Si vous obtenez ce genre de message, alors la compilation a échoué. Votre programme contient des erreurs et il vous faut les corriger.

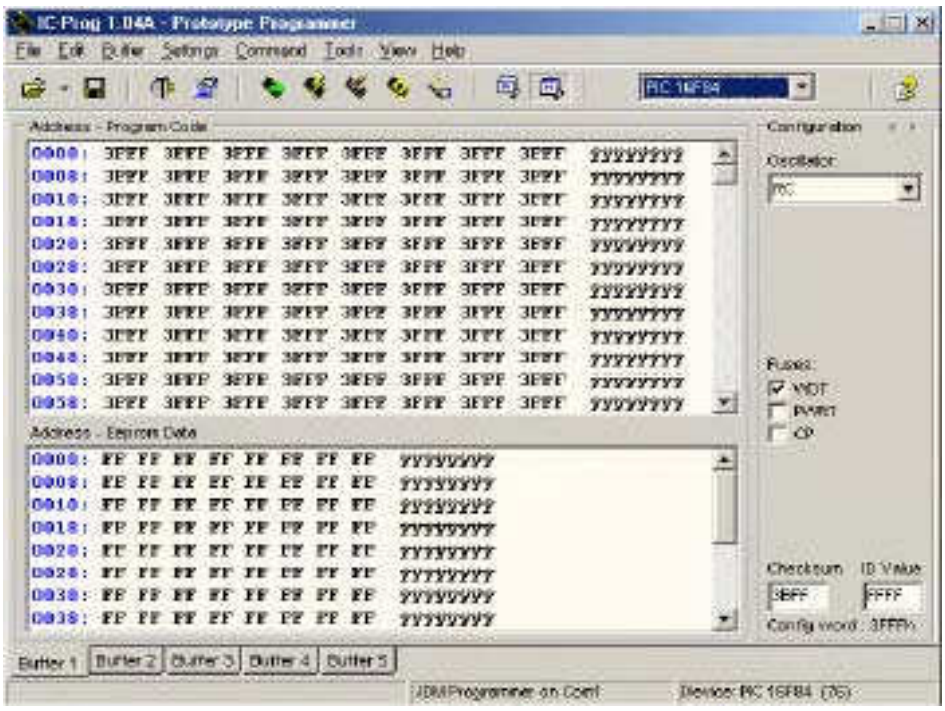
Pour connaître les erreurs que MPASMWIN a rencontré, dans le menu *Fenêtre* de EditAlgo, cliquez sur *Erreur* et sélectionnez le fichier *.ERR correspondant à votre programme. La fenêtre, en bas à gauche, apparaît. Généralement, les erreurs sont clairement explicitées. Il suffit de lire attentivement le fichier *.ERR.

Remarque : Les deux messages indiqués ci-contre sont seulement des avertissements et n'empêchent pas la compilation. En fait vous aurez toujours ces deux messages. Ne vous en préoccupez pas.



Quand vous avez corrigé votre programme, enregistrez de nouveau le fichier *.ASM puis réessayez de le compiler.

Septième étape : Transfert du programme sur le PIC16F84 (Microcontrôleur)



Tout d'abord, reliez un programmateur au port série COM1 de votre ordinateur et branchez le programmateur à une carte Microcontrôleur PIC. Le PIC ne doit pas être alimenté et les bits 6 et 7 du Port B doivent être débranchés.

Quand le branchement est fait, lancez I*CPROG*. ICPROG est un logiciel gratuitement mis à disposition par Bonny Gijzen.

La fenêtre ci-contre apparaît. Vérifiez que en haut à droite, le microcontrôleur sélectionné est bien le PIC 16F84.

Dans *Fichier*, sélectionnez *Open File* et choisissez le fichier *.HEX correspondant à votre programme. Si à droite, Oscillator n'est pas XT, il y a un problème. Peut-être vous êtes vous trompé de fichier.

Pour lancer le transfert cliquez sur l'icône représentant un circuit frappé par un éclair. Confirmez votre choix en cliquant OK. Le transfert commence.

Il ne vous reste qu'à tester votre programme avec le microcontrôleur !

ANNEXE 1

PIC16F8X

TABLE 9-2 PIC16FXX INSTRUCTION SET

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

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

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

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

ANNEXE 2

```
;  Clignotement d'une LED a l'aide du TIMER
;  Ce programme utilise une interruption générée par le timer toutes les 256µs .
;  A chaque interruption il execute la routine ServiceRtcc qui décremente la variable
;  TIME.
;  Le programme principal (dans ce cas BL1) est une boucle qui teste dans la routine ;
SecondOver si TIME est égal à 0. Si oui, il incrémente le PORTB. Une LED  branchée à
la voie 0 du PORTB changera d'état toutes les 2ms (256µs * 8). Sur la voie 1, toutes les 4
ms ...., sur la voie 7 toutes 256ms (2ms * 128)
;  Quartz = 4Mhz

        list p=16f84,f=inhx8m

      __config  B'00000000000001'    ; WDT Disabled, XT Oscillator
;
      #include "p16f84.inc"
;
TIME    equ     0x10
;
        org         0
        goto        Start
;
        org         4
        goto        ServiceRtcc
;
        org         10
Start
        movlw       0                   ;make port b outputs
        movwf       PORTB
;       movwf       PORTA
        tris        PORTB
        tris        PORTA
        call        InitRtcc
        clrf        PORTB               ;turn off leds
        clrf        PORTA
BL1
        call        SecondOver          ;wait for 1/2 second
        incf        PORTB,1             ;toggle leds
        goto        BL1
;
InitRtcc
        movlw       B'10000000'
        option
        clrf        TMR0                ;start time
        movlw       B'10100000'         ;enable interrupts
        movwf       INTCON              ;       /
        movlw       .8                  ;initialize time
        movwf       TIME
        return
```

```
;
ServiceRtcc
        btfsc   INTCON,T0IF     ;rtcc interrupt?
        decf    TIME,1          ;yes then dec time
        clrf    INTCON          ;clr all interrupts
        bsf     INTCON,T0IE     ;enable RTIE
        bcf     STATUS,Z        ; Z=0
        retfie                  ;not zero then return
;
SecondOver
        movf    TIME,1          ;check if time = 0
        btfss   STATUS,Z        ;       /
        goto    SecondOver      ;no then loop
        movlw   .8              ;load for 1/2 second
        movwf   TIME

        return

        end
```

ANNEXE 3

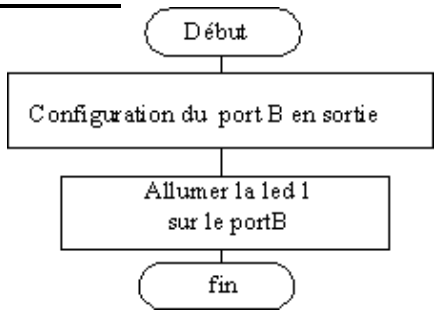
```
;  Clignotement d'une LED a l'aide du Watch Dog Timer (WDT)
;  En SIM, il faut attendre 30 sec avant que le WDT se reveille

;  LIGHTS.PIC: RC Oscillator version
;  This is as simple as it gets.
;  Port B outputs produce square waves if approxfollowing frequencies.
;  Pin   PB0  PB1  PB2  PB3  PB4  PB5  PB6  PB7
;  Hz    28   14    7  3.5  1.7  0.8  0.4  0.2
;
        list p=16f84,f=inhx8m
      __config  H'00ff'    ; WDT Enabled, RC Oscillator
;
      #include "p16f84.inc"
;
      org        0
      goto       Start
;
      org        10
Start
      bsf       STATUS,RP0 ; select bank1
      movlw   0       ; set portb to output
      movwf   TRISB   ; TRISB = 0
      movlw   B'00000111'
      movwf   OPTION_REG          ; OPTION = 0F  prescaler 128
      bcf       STATUS,RP0 ; select bank0
Loop
      sleep
      incf      PORTB,f
      goto      Loop
      end
```

TRAVAUX PRATIQUE

Pour réaliser les TP il est recommandable d'utiliser le logiciel EDITALGO

TP 1

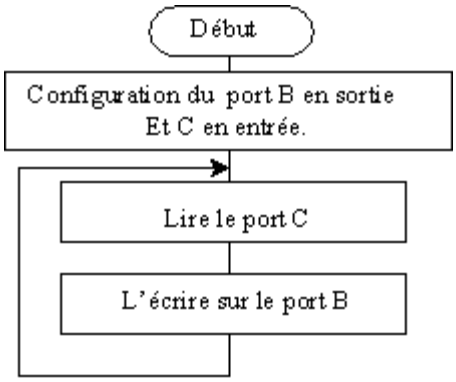
Allumage d'une LED en sortie.




```
* fichier led.asc
* Allumage d'une LED sur le portb

*Equivalences registres

portb       equ         $1004

*Début du programme implantation en EEPROM

            org         $F800

*********************
*   Configuration
*********************

* configuration du portb
* le portb en mode single est unidirectionnel et en sortie (pas de
DDRB)

*********************
*Programme principal
*********************

*Allumage de la led 1 sur le portb

start       ldaa        #$01
            staa        portb
            bra         *
****************************
* Vecteurs d'interruptions
****************************
*vecteur de reset

            org   $fffe
rstvect           fdb   start

* le vecteur est obligatoire pour faire demarrer le programme au
bon endroit

            End
```

TP2

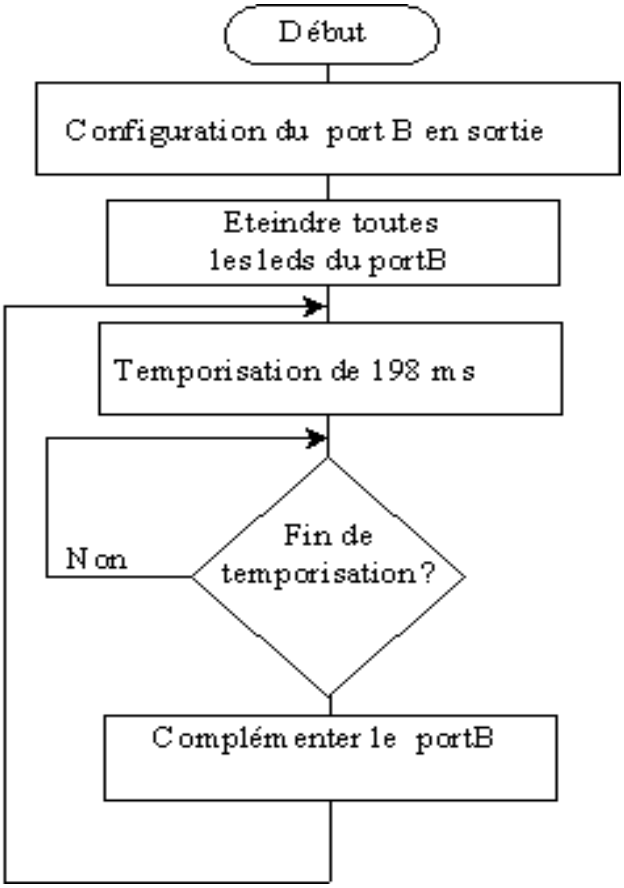
Recopie d'un port d'entrée en sortie




```
* fichier recop.asc
* recopie d'un port d'entree sur un port de sortie

*Equivalences registres

portb       equ         $1004
portc       equ         $1003
ddrc        equ         $1007

*Début du programme implantation en EEPROM

            org         $F800
*********************
*   Configuration
*********************
* configuration du portb  et C
* portc en entree pas de ddrb, b tjs en sortie

start       clr         ddrc

*
*********************
*Programme principal
*********************
*Allumage de la led 1 sur le portb

debut       ldaa        portc
            staa        portb
            bra         debut
*****************************
* Vecteurs d'interruptions
*****************************
*vecteur de reset

            org         $fffe
rstvect           fdb         start

            end
```

TP3

Clignotement de toutes les LED sur un port en sortie

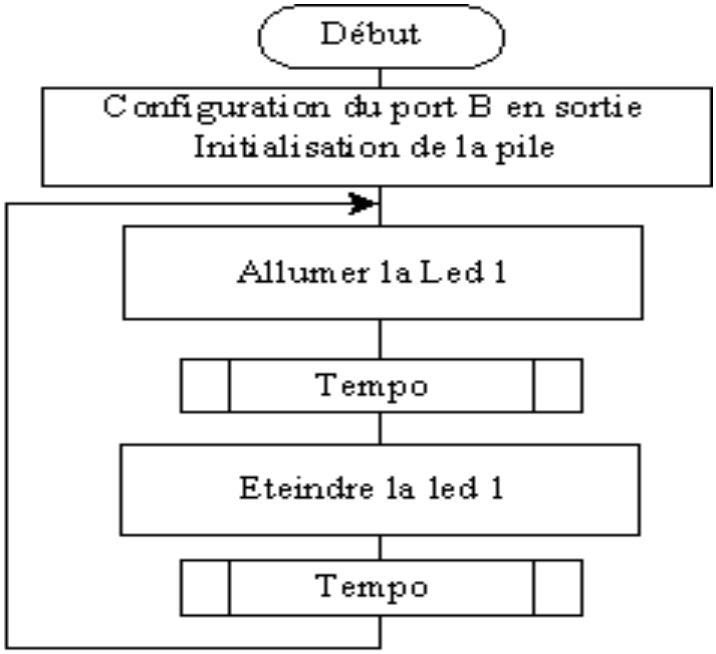


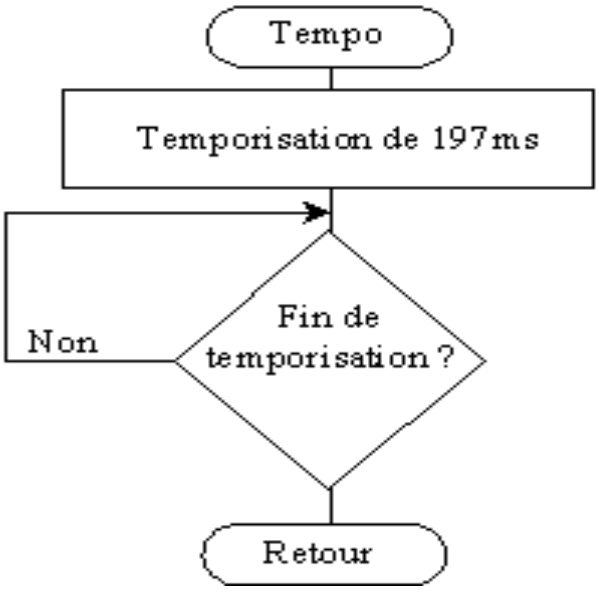

```
* fichier clignot.asc
* clignotement de LEDS d'un port en sortie

*Equivalences registres

portb        equ          $1004

*Début du programme implantation en EEPROM

             org          $F800

********************
*   Configuration
********************

********************
*Programme principal
********************


*Eteindre toutes les leds sur le portb.
```

```
start       clr         portb

*temporisation de 197ms

debut       ldx         #$ffff

*tempo basée sur le nombre de boucle à effectuer
* ici on execute 65535 fois les instructions dex et bne.
* elles durent 6 cycles d'horloge. Soit pour un µP
*cadencé à 8MHz Horloge =f/4=2Mhz
* Soit 0,5µs pour la periode.
*Donc la duree d'un bne et dex et de 3µs
*repeté 65535 fois donne 197 ms

temp        dex
            bne         temp
            com         portb
            bra         debut

****************************
* Vecteurs d'interruptions
****************************
*vecteur de reset

            org   $fffe
rstvect           fdb   start
            end
```

TP 4

Clignotement d'une seule LED sur un port en sortie à l'aide d'un tempo par boucle. Introduction d'un sou programme.






```
*fichier ledclign.asc
* Clignotement d'une seule led sur le portb

*Equivalences registres

portb         equ          $1004

*Début du programme implantation en EEPROM
```

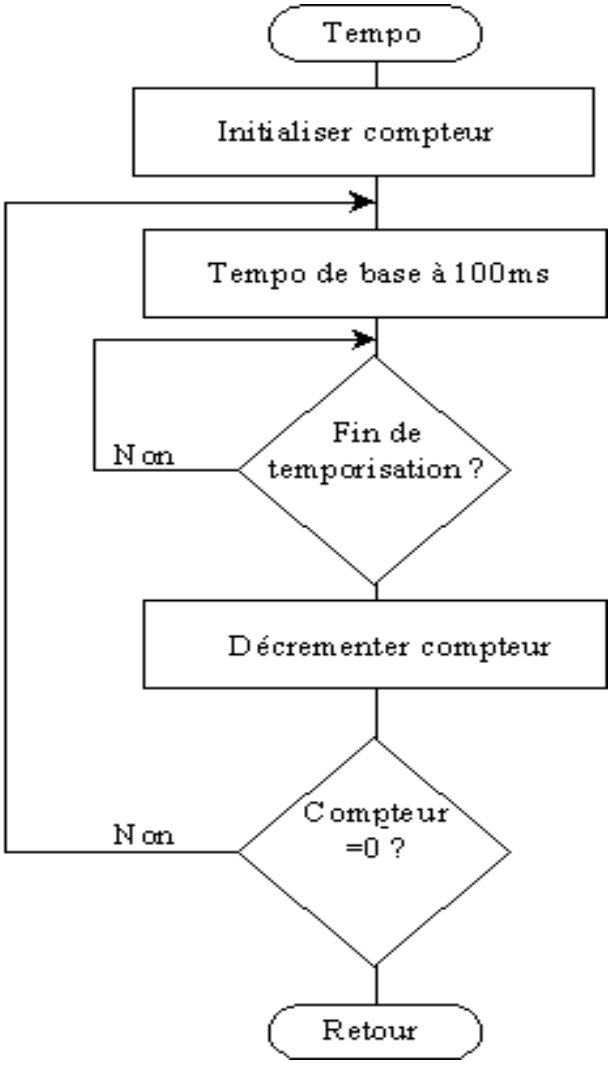
```
        org         $F800

********************
*   Configuration
********************
* L'initialisation de la pile est obligatoire lors de
l'utilisation d'un
sous programme.

start         lds         #$00ff


********************
*Programme principal
********************

*Allumer la led 1 sur le portb.

debut         ldaa        #$01
        staa        portb

* Attente de 197ms

        bsr         tempo

* Eteindre la led 1 sur le portb.

        ldaa        #$00
        staa        portb

* Attente de 197ms

        bsr         tempo
        bra         debut

********************
*  Sous Programmes
********************

*  SP Tempo
*****************



*temporisation de 197ms

tempo         ldx         #$ffff

*tempo basée sur le nombre de boucle à effectuer
```

```
temp            dex
        bne             temp
        rts


***************************
* Vecteurs d'interruptions
***************************
*vecteur de reset

        org     $fffe
rstvect         fdb     start
        end
```

TP 5

Clignotement d'une seule LED sur un port en sortie à l'aide d'un tempo par boucles imbriquées.

Même pgm principal qu'en TP 4.



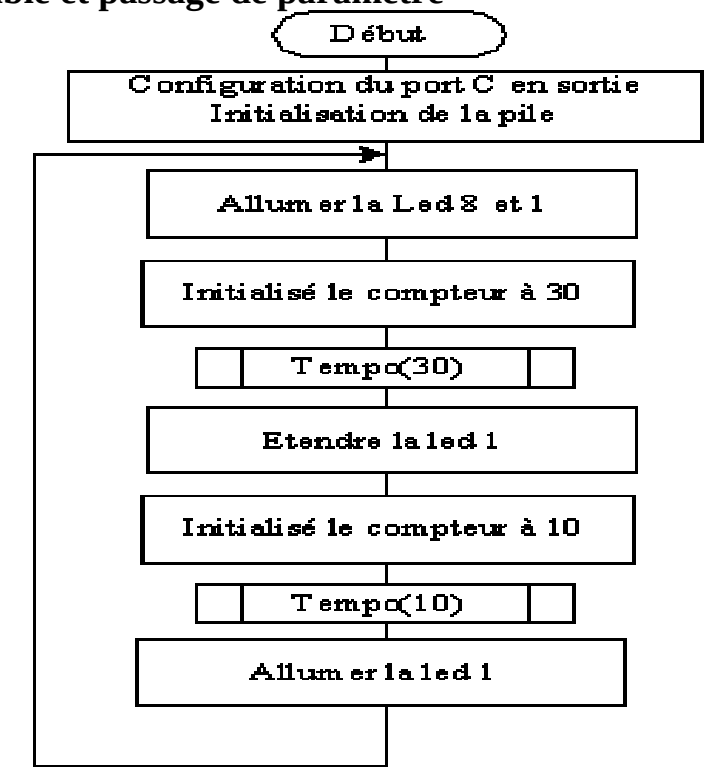

```
*fichier led1clig.asc
* Clignotement d'une seule led sur le portb
* Le temps de clignotement est de 1 s

*Equivalences registres

portb       equ         $1004

*Début du programme implantation en EEPROM

            org         $F800

*********************
*   Configuration
```

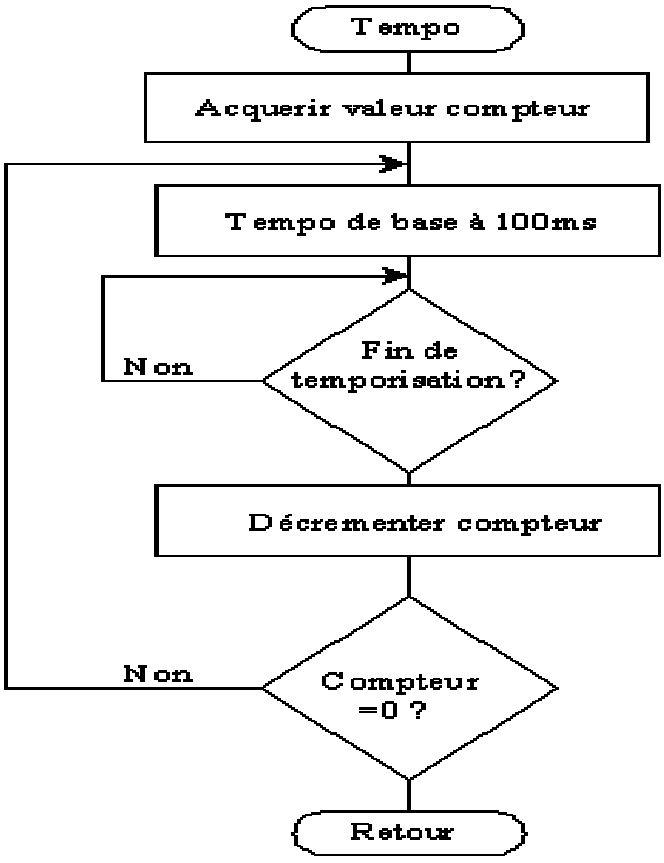
```
*********************
* L'initialisation de la pile est obligatoire lors de
l'utilisation d'un
sous programme.

start       lds         #$00ff


*********************
*Programme principal
*********************

*Allumer la led 1 sur le portb.

debut       ldaa        #$01
            staa        portb

* Attente de 197ms

            bsr         tempo

* Eteindre la led 1 sur le portb.

            ldaa        #$00
            staa        portb

* Attente de 197ms

            bsr         tempo
            bra         debut

*********************
*  Sous Programmes
*********************

*  SP Tempo
*****************


*temporisation de 1s

* chargement du compteur du nombre de boucle a effectuer

tempo       ldaa        #10

* Chargement de la valeur permettant de faire une tempo de 0,1s

t1s         ldx         #33333

*tempo basée sur le nombre de boucle à effectuer
```

```
t100ms      dex
            bne         t100ms

* fin de la boucle de 100ms

* décomptage du nombre de boucle

            deca
            bne         t1s
            rts



***************************
* Vecteurs d'interruptions
***************************
*vecteur de reset

            org   $fffe
rstvect           fdb   start
            end
```

TP 6

Clignotement d'une seule LED2 sur le même port en sortie.
La durée d'extinction et d'allumage étant différente
Introduction de variable et passage de paramètre





```
* Fichier hautbas.asc
* Led allumée durant un temps different de celui eteint.
*Equivalences registres

portb         equ          $04

* Constantes

dureeal       equ          30
dureeet       equ          10

*Variables

        org          $0000
duree         rmb          1

*Début du programme implantation en EEPROM

        org          $F800

*********************
*   Configuration
*********************

start         lds          #$00ff

*********************
*Programme principal
*********************

* préparation de l'adressage indexé pour instruction

        ldy          #$1000

*Allumer la led 1 et 8 sur le portb.

        ldaa         #$81
        staa         portb,y

* Attente de 3 secondes

debut         ldaa         #dureeal
        staa         duree
        bsr          tempo

* Eteindre la led 1 sur le portb.

        bclr         portb,y,#$01
```

```
* cette instruction permet d'eteindre la led 1 sans toucher
aux autres sorties

* Attente de 1 secondes

        ldaa        #dureeet
        staa        duree
        bsr         tempo

* Allumer la led 1 sur le portb.

        bset        portb,y,#$01


* cette instruction permet d'allumer la led 1 sans toucher
aux autres sorties

        bra         debut

********************
*  Sous Programmes
********************

*  SP Tempo
******************
*temporisation parametre par duree

* chargement du compteur du nombre de boucle a effectuer

tempo       ldaa        duree

* Chargement de la valeur permettant de faire une tempo de
0,1s

t1s         ldx         #$33333

*tempo basée sur le nombre de boucle à effectuer

t100ms      dex
        bne         t100ms

* fin de la boucle de 100ms

* décomptage du nombre de boucle

        deca
        bne         t1s
        rts

***************************
```

```
* Vecteurs d'interruptions
***************************
*vecteur de reset

        org    $fffe
rstvect         fdb    start
        end
```

BIBLIOGRAPHIE

Philippe Lebenneur : Les microcontrôleurs PIC 16F87X, Granville 2003

François de Dieuleveult, Hervé Fanet : Principes et pratique de l'électronique

Bigonoff : La programmation des pics Révision 6