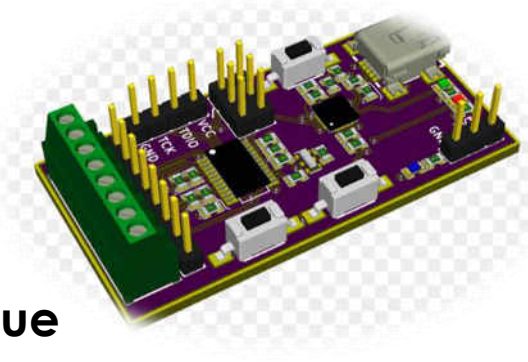
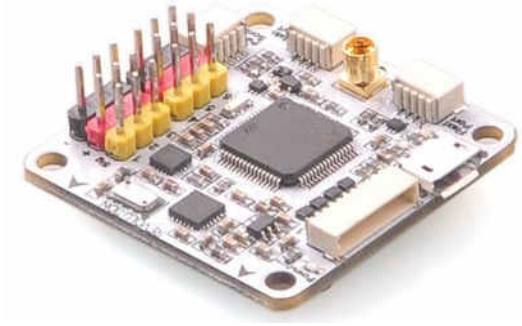
Ministre de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Seddik BENYAHIA - Jijel
Faculté des Sciences & de la Technologie
Département d'Electronique



Polycopié Pédagogique

Systèmes à MicroContrôleurs

- Cours & Exercices -

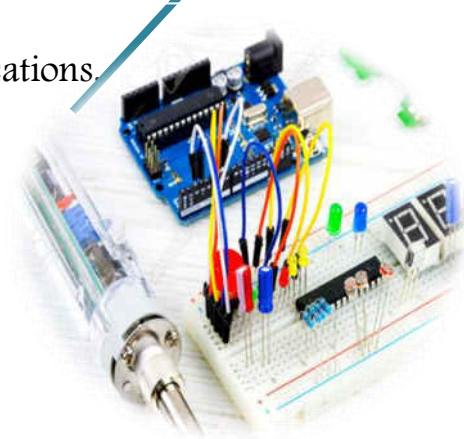


Rédigé par :

Dr. Ammar SOUKKOU

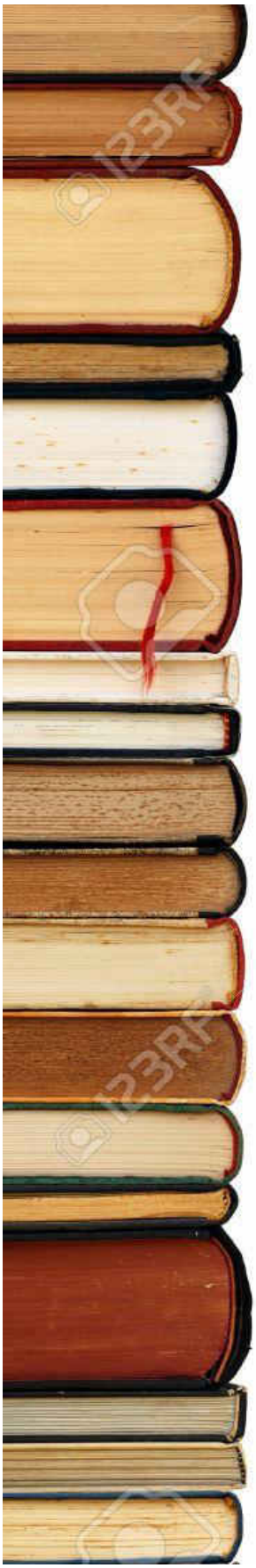
soukkou.amr@gmail.com

- **Manuscrit élaboré selon le programme officiellement agréé et confirmé par le CPNDST.**
- ~~**Public ciblé :**~~
  - **Troisième année** Licence Electronique.
  - **Troisième année** Licence Systèmes de Télécommunications.
  - **Master I**: Electronique des Systèmes Embarqués.
  - **Master I**: Systèmes de Télécommunications.



Année Universitaire
2018/2019

Déclaration



Je sous signé Monsieur Ammar SOUKKOU, déclare que:

1. L'effort rapporté dans ce travail, sauf indication contraire, est mon propre travail original.
2. Ce travail n'a pas été soumis pour l'obtention d'un diplôme ou pour la participation à un examen dans une autre université.
3. Ce travail ne contient pas des données, des images, des graphiques ou autres informations provenant d'autres sources, à moins qu'il ne le soit explicitement reconnu.
4. Ce travail ne contient pas du provenant à partir d'autres références, à moins qu'il ne le soit explicitement reconnu. Lorsque d'autres sources du texte ont été citées, alors:
    a. Leurs mots ont été réécrits mais les informations générales qui leurs ont été attribuées ont été référenciées;
    b. Lorsque leurs mots exacts ont été utilisés, le texte a été placé entre guillemets et référencée.
5. Ce travail ne contient pas du texte, des graphiques ou des tableaux copiés et collés à partir d'Internet. Dans le cas échéant, les sources des copiés sont explicitement reconnus et référenciées.

Dr. Ammar SOUKKOU

Université de Jijel,

Algérie.

soukkou.ammar@univ-jijel.dz

soukkou.amr@gmail.com

CONTENU DU POLYCOPIE

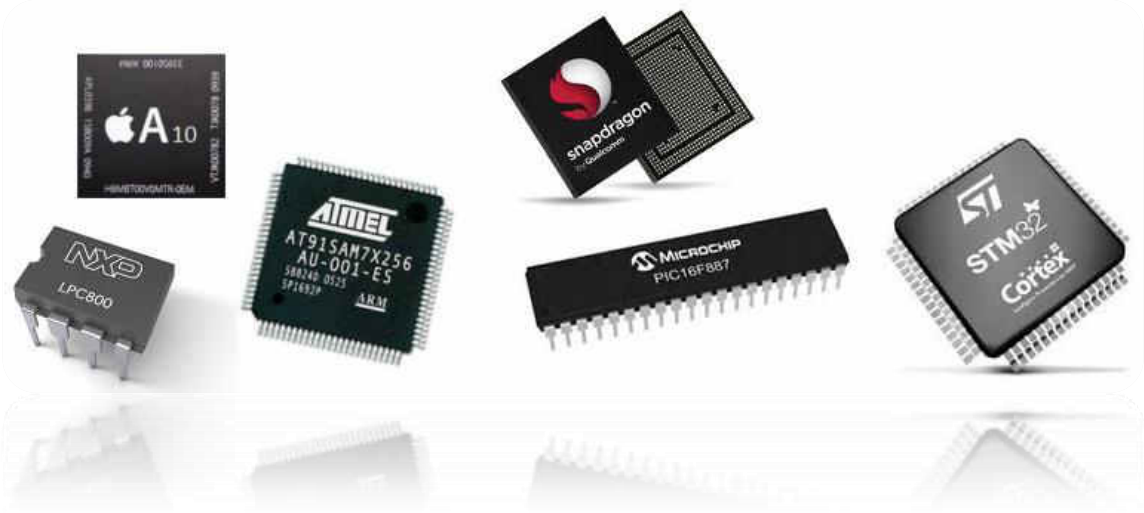
a. Objectifs

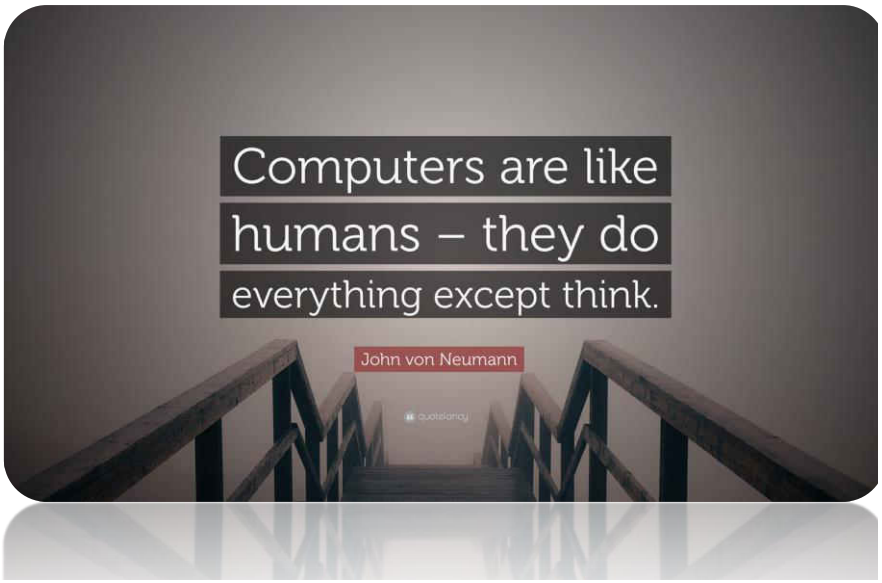
Comprendre le fonctionnement d'un microcontrôleur et son interaction avec ses principaux organes d'Entrées/Sorties (Timer, Convertisseur, …). Se familiariser avec les outils de développement et la programmation du microcontrôleur pour le contrôle des périphériques.

b. Connaissances préalables recommandées

Système à microprocesseurs. Conception des systèmes à microprocesseurs.

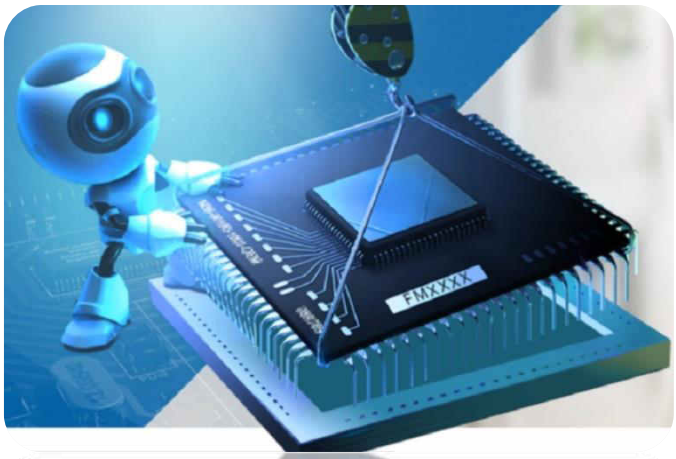
c. Sommaire

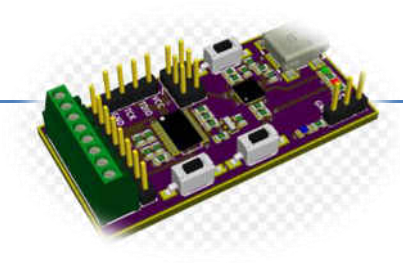




CHAPITRE 1

Du microprocesseur au microntrôleur





CHAPITRE 1

Du microprocesseur au microcontrôleur

Objectifs

L'objectif est de

- ✓ Présenter la structure générale des systèmes microprogrammés et les concepts associés.
- ✓ Donner les notions et des potentialités permettant d'analyser et de comprendre des éléments matériels (Architecture) ou logiciels (programmes) de systèmes microprogrammés.
- ✓ Définir les éléments génériques des systèmes à base de microcontrôleurs, les familles ainsi que les critères intervenants dans le choix des microcontrôleurs pour des applications spécifiques.

Avant Propos

Par définition, un système logique (circuit numérique) est un système qui traite l'information d'une manière digitale. Il existe deux grands types de systèmes (*fonctions ou circuits*) logiques :

- ✓ Les systèmes logiques *combinatoires* résultent de l'analyse combinatoire des variations des grandeurs d'entrées uniquement.
- ✓ Les systèmes logiques *séquentielles* ou les systèmes à base des bascules, qui résultent de l'association de plusieurs fonctions logiques combinatoires et qui supposent l'existence d'une horloge (**un paramètre additionnel qui est le temps**) où l'état de la sortie (**futur**) dépend, non seulement de l'état des entrées en cours (**présent**) mais aussi de l'état précédent de la sortie (**passé**).
- ✓ D'une manière générale, on peut classer les systèmes logiques selon leurs caractères fonctionnels et structurels de sa constitution interne comme illustre le diagramme de la figure 1.1.

En savoir +

- Pour réaliser un système logique, il faut disposer de composants convenables à sa fonction (éléments de base).

- **Systèmes logiques combinatoires :** Portes logiques, Décodeurs, Encodeur, Circuits de transcodage, Multiplexeurs et Démultiplexeurs, Générateurs de parité, Circuits d'aiguillage des informations, etc.
- **Systèmes logiques séquentiels :** Bascules, registres, compteurs.
- **Systèmes à fonctionnement programmable :** Microprocesseurs, Microcontrôleurs, mémoires, etc.

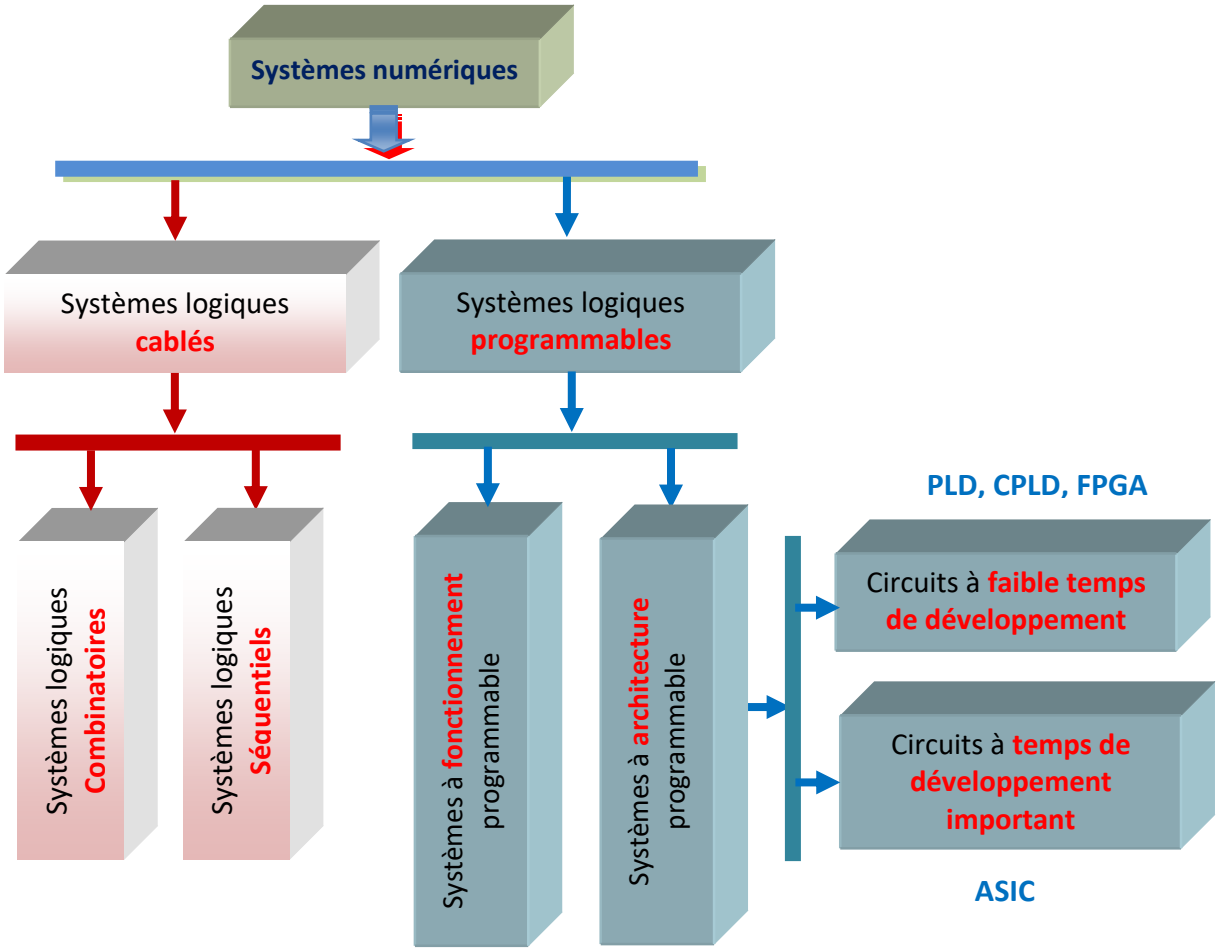


Figure 1.1 : *Classement des systèmes numériques.*

Le choix du type de logique pour résoudre un problème, dépend de plusieurs critères :

- Complexité ;
- Coût ;
- Evolutivité :
  - ✓ En logique câblée, la moindre modification du problème entraîne la mise au point d'un nouveau circuit.
  - ✓ Alors qu'en logique programmée on pourra parfois se contenter d'une modification du programme.
- Rapidité.

Info

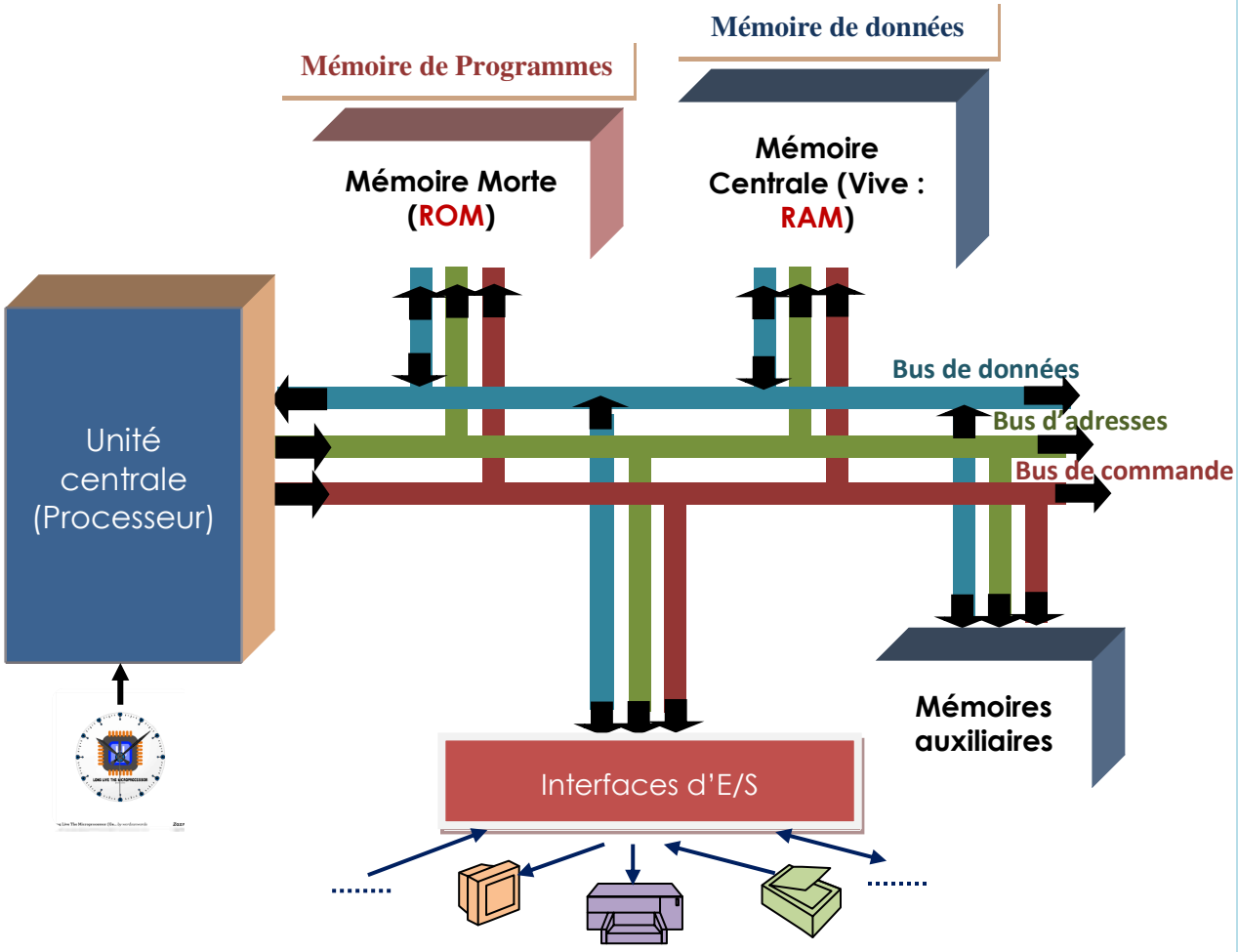
- Les fonctions logiques sont réalisées par voie matérielle (à base des portes logiques). Donc, la logique câblée qu'elle soit combinatoire ou

séquentielle exige un grand nombre de composants et rend les montages encombrants et chers.

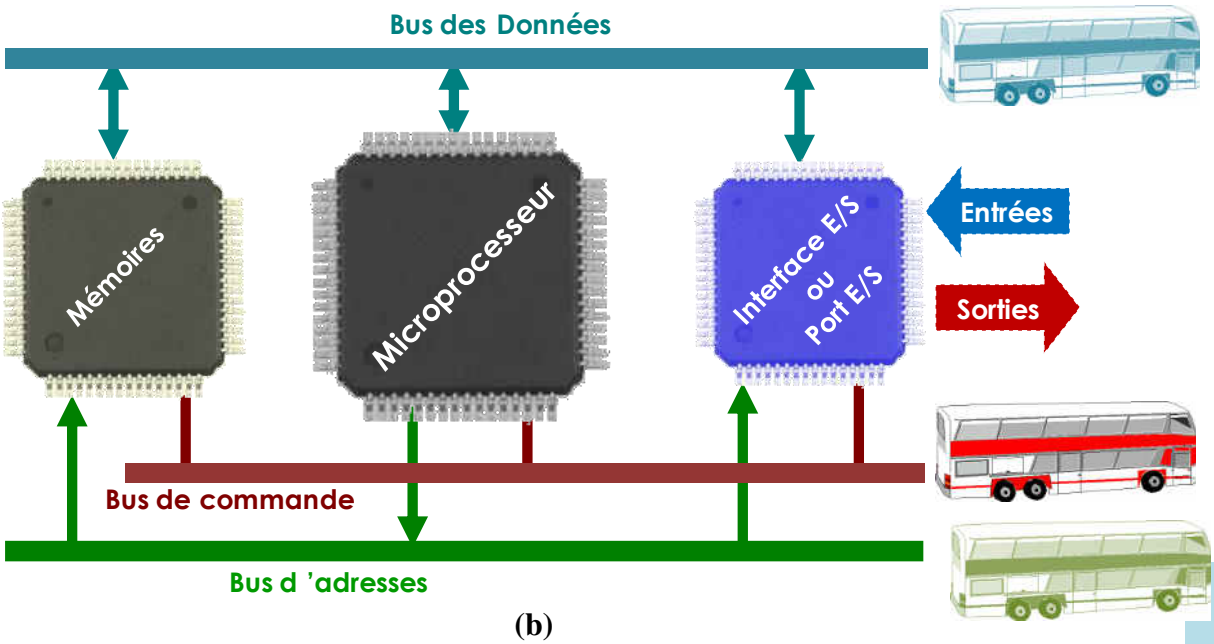
- Dans la logique programmé, les fonctions sont réalisées par voie logicielle, et sont modifiables à tout moment en un temps réduit. De plus, le nombre de composants est réduit à sa plus simple expression.
- Le schéma électrique est transcrit en une suite d'instruction qui constitue le programme. En cas de modification des équations avec les mêmes accessoires, l'installation ne comporte aucune modification de câblage seul le jeu d'instructions est modifié

1.1. Définition d'un système microprogrammé

Un système microprogrammé est un ensemble de circuits intégrés qui va pouvoir être programmé pour réaliser des tâches particulières. Le but étant de pouvoir en changer les tâches sans toucher à l'électronique. Sa structure est donnée par la figure 1.2.



(a)



(b)

Figure 1.2 : *Architecture schématique d'un système microprogrammé.*

L'unité centrale (processeur ou CPU) est le **Cerveau** du système microprogrammé (l'organe principal de la carte). Elle se charge de l'exécution des programmes qu'on lui a demandé et de la coordination entre les différents organes du système. Elle est composée de deux unités fondamentales :

- ✓ Unité de traitement ou Unité Arithmétique et Logique (**UAL**).
- ✓ Unité de commande.
- ✓ Des registres généraux (**RG**).

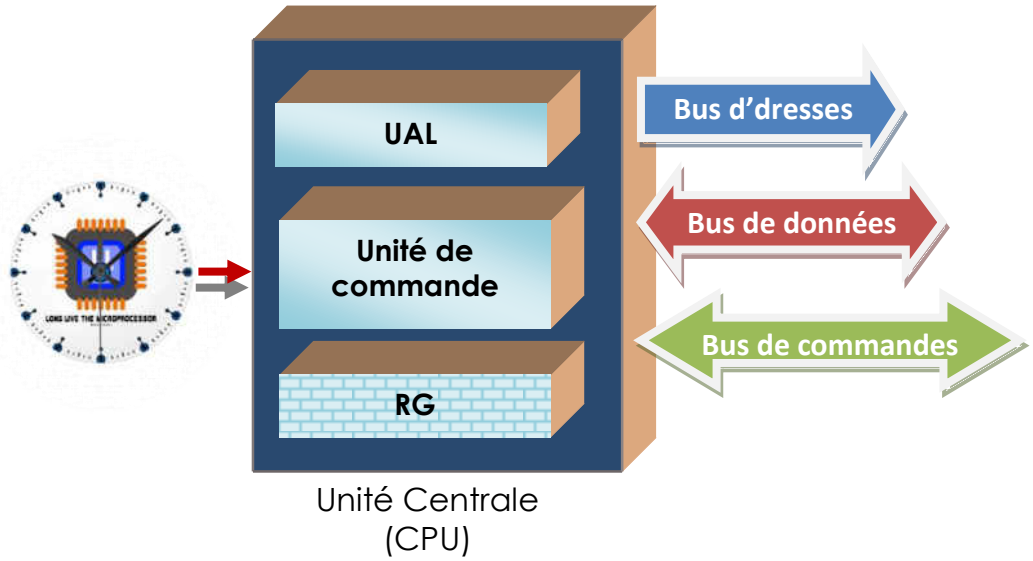
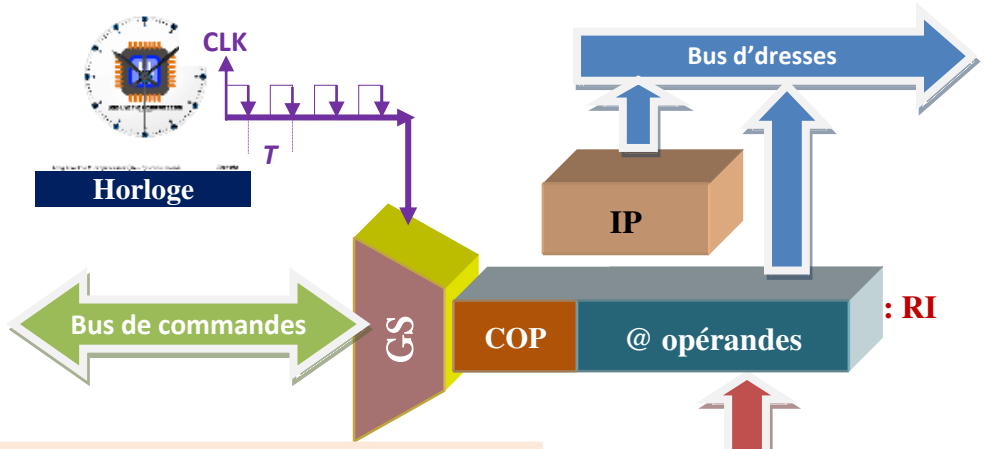


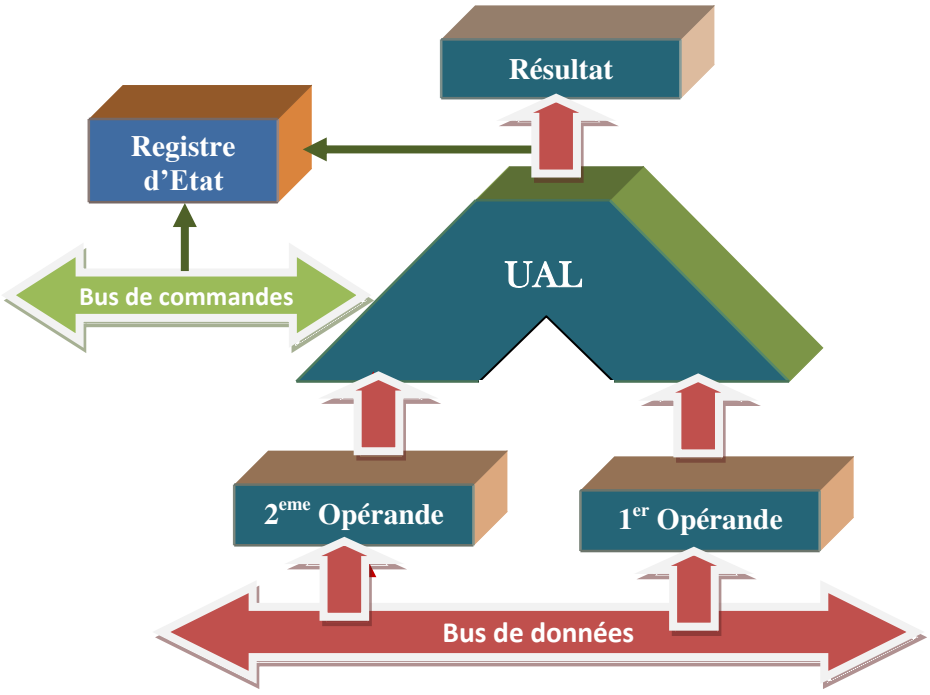
Figure 1.3 : *Structure de l'unité centrale de traitement.*



RI : Registre d'Instruction ;
PC : Pointeur d'instruction ;
COP : Code Opération ;
@opérandes : Adresses des opérandes ;
GS : Gérateur de séauences

- ✓ L'unité de commande extrait de la mémoire centrale la nouvelle instruction à exécuter.
- ✓ Elle analyse cette instruction et établi les connexions électriques correspondantes fans l'unité de traitement.
- ✓ Elle extrait de la MC les données sur lesquelles porte cette instruction.
- ✓ Elle déclenche le traitement des ces données dans l'unité de traitement.
- ✓ Eventuellement, elle range le résultat dans la MC.
- ✓ Puis, elle recommence avec l'instruction suivante.

Figure 1.4 : *Structure de l'unité de commande.*



- ✓ L'UAL contient tous les circuits électroniques qui réalisent effectivement les opérations désirées.
- ✓ Ces opérations sont principalement les opérations arithmétiques (+, -, x, /) et les opérations logiques (Inversion des bits, ET, OU, OU exclusif).
- ✓ Des registres et des indicateurs sont associés à l'UAL :
  - Les registres servent à contenir les opérandes et les résultats intermédiaires.
  - Les indicateurs (**Registre d'état**) pour indiquer l'état du résultat : Résultat nul, >0, <0, débordement, …etc.
- ✓ Ces registres sont accessibles aux programmeurs (l'utilisateur).

Figure 1.5 : *Structure de l'unité arithmétique et logique.*

La mémoire c'est un organe (Dispositif électronique de nos jours), capable de contenir (d'accueillir), de conserver et de restituer à la demande sans les modifier de grandes quantités d'information. On peut aussi classer les mémoires par leur utilisation ou leur contenu: **Mémoire programme** (ROM ou PROM) et **mémoire de données** (RAM).

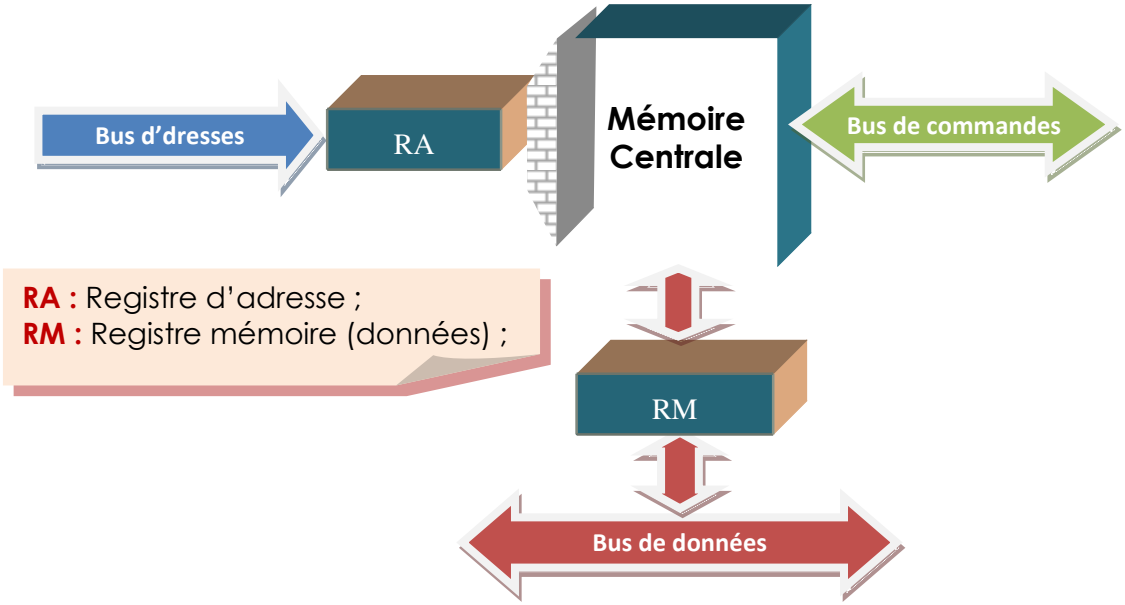


Figure 1.6 : *Structure de la mémoire centrale.*

Les blocs constitutifs du système microprogrammé sont reliés entre eux par trois **BUS**, comme le montre le schéma de principe de la figure 1.1. Ces trois bus ont pour nom :

- Le **bus de données** (bidirectionnel) est le bus sur lequel circulent les données.
- Le **bus d'adresses** (unidirectionnel) est le bus sur lequel circulent les adresses des données
- Le **bus de commandes** ou de contrôle (bidirectionnel) est le bus sur lequel circulent les commandes (horloge, ordres de lecture, ordres d'écriture, etc.).

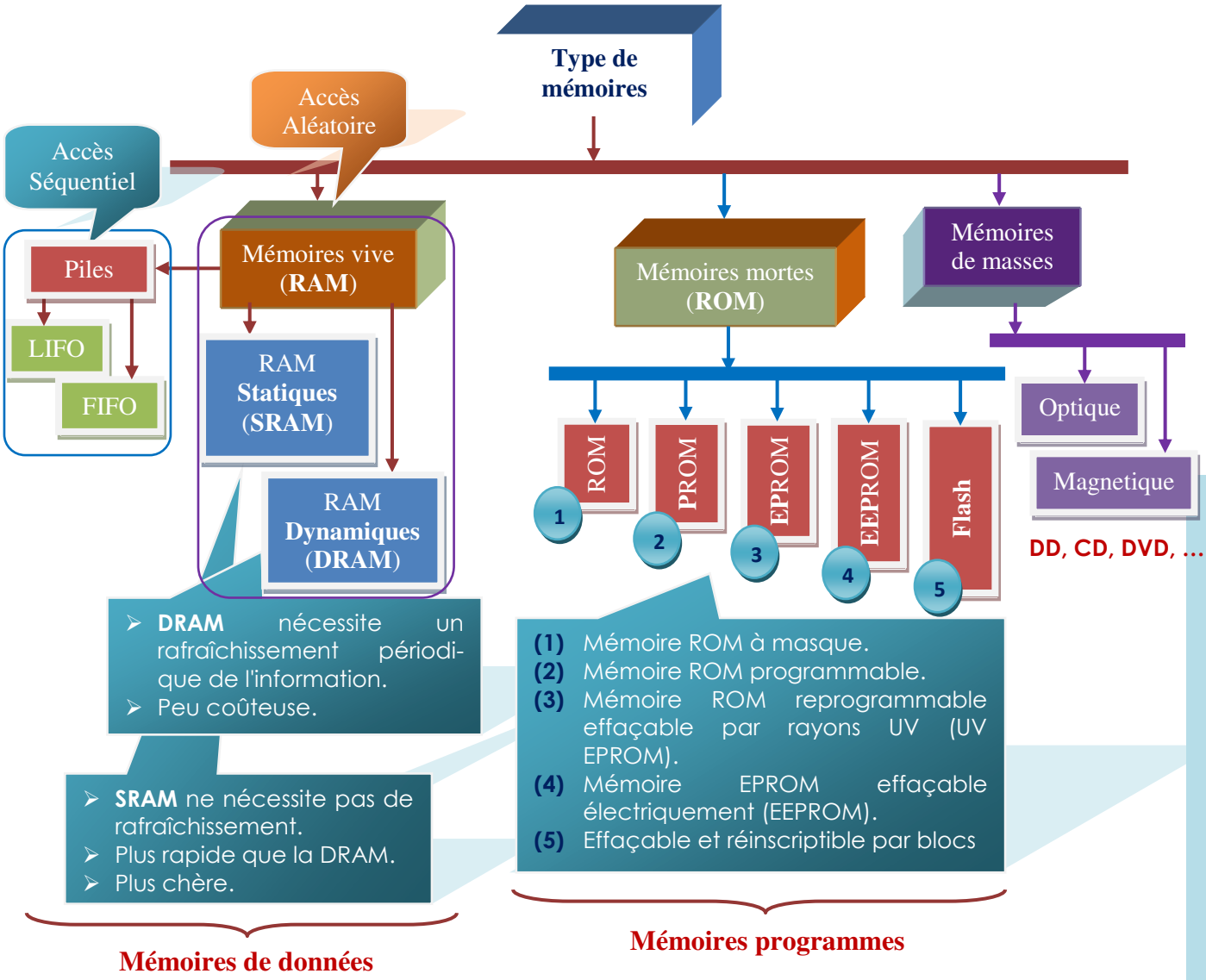


Figure 1.7 : *Familles des mémoires dans un système microprogrammé.*

Info

Le fonctionnement de l'unité centrale est schématiquement toujours le même, même si aujourd'hui il est de plus en plus complexe. Il suit les étapes représentées par la figure 1.8.

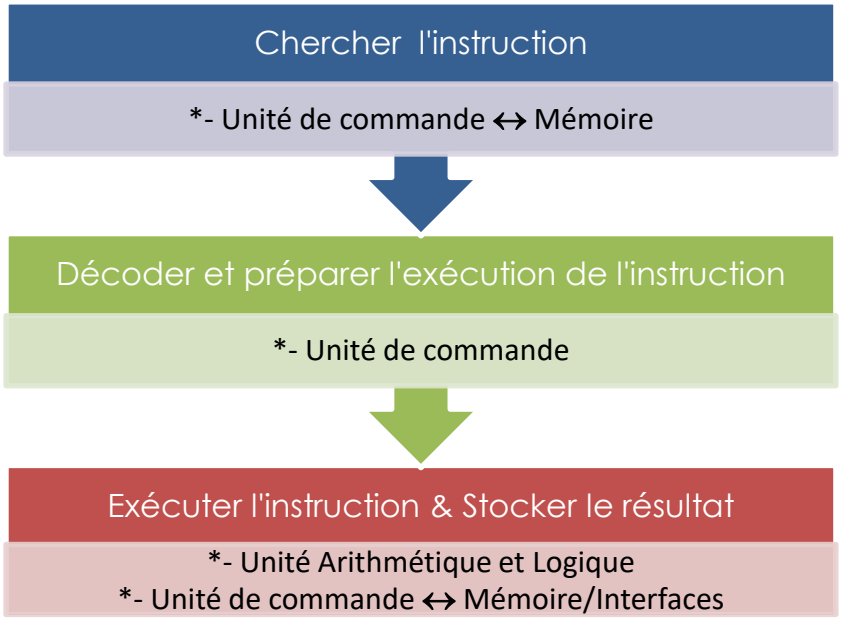
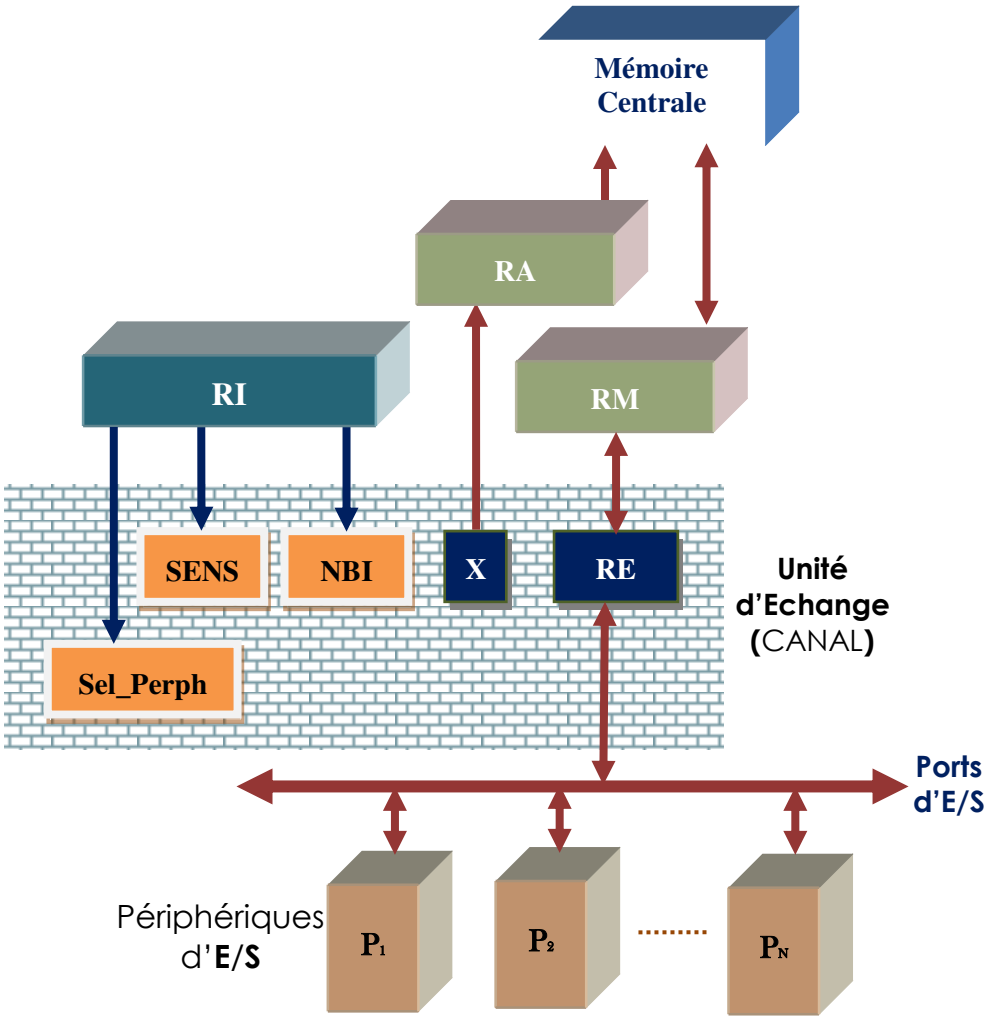


Figure 1.8 : *Etapes d'exécution d'une instruction dans un système microprogrammé.*

Les Ports d'entrées-sorties (interfaces d'E/S) permettent une interface avec l'extérieur (périphériques), elle gère les entrées (clavier, capteurs, etc.) et les sorties (écran, imprimantes, etc.) du système microprogrammé.



RI : Registre d'Instruction ;
RM : Registre Mémoire ;
RA : Registre d'Adresse ;
RE : Registre d'Echange ;
X : Adresse de rangement dans la MC ;
NBI: Nombre d'informations à transférer ;
SENS : Sens de transfert des données ;
Sel_Perph : Sélection du périphérique ;

Figure 1.9 : *Fonctionnement de l'interface d'E/S.*

- Pour piloter les périphériques, l'unité d'échange (canal) disposera.
  - D'un registre mémorisant l'adresse du périphérique (**X**),
  - Le registre de sélection du périphérique (**Sel_Perph**), et
  - D'un registre permettant l'échange d'informations entre unité centrale et les périphériques, le registre d'échange (**RE**) à la manière du registre mot de l'unité de mémoire.

La mise en œuvre des systèmes microprogrammés s'appuie sur deux modes de réalisation distincts :

- Le matériel (**Hardware**) correspond à l'aspect concret du système : unité centrale, mémoire, organes d'entrées-sorties, etc.
- Le logiciel (**Software**) correspond à un ensemble d'instructions, appelé programme, qui sont contenues dans les différentes mémoires du système et qui définissent les actions effectuées par le matériel :
  - Logiciels de gestion ou le système d'exploitation.
  - Logiciels d'applications.

Les applications à systèmes microprogrammés sont multiples et variées :

- ✓ Ordinateur,
- ✓ Console de jeux,
- ✓ Télévision,
- ✓ Téléphone portable,
- ✓ Distributeur automatique d'argent,
- ✓ Robotique,
- ✓ Automobile,
- ✓ Instrumentation,
- ✓ … etc.

Système d'Exploitation est l'interface (**Soft**) entre l'utilisateur et le système microprogrammé (**Hard**). Plus formellement, c'est un ensemble de programmes dont la fonction est de :

- Gérer les ressources physiques (processeur, mémoire, disques, etc.) et logiques (fichiers et bases de données, etc.).
- Contrôler les entrées-sorties.
- Ordonnancer les travaux.
- Gérer les erreurs.

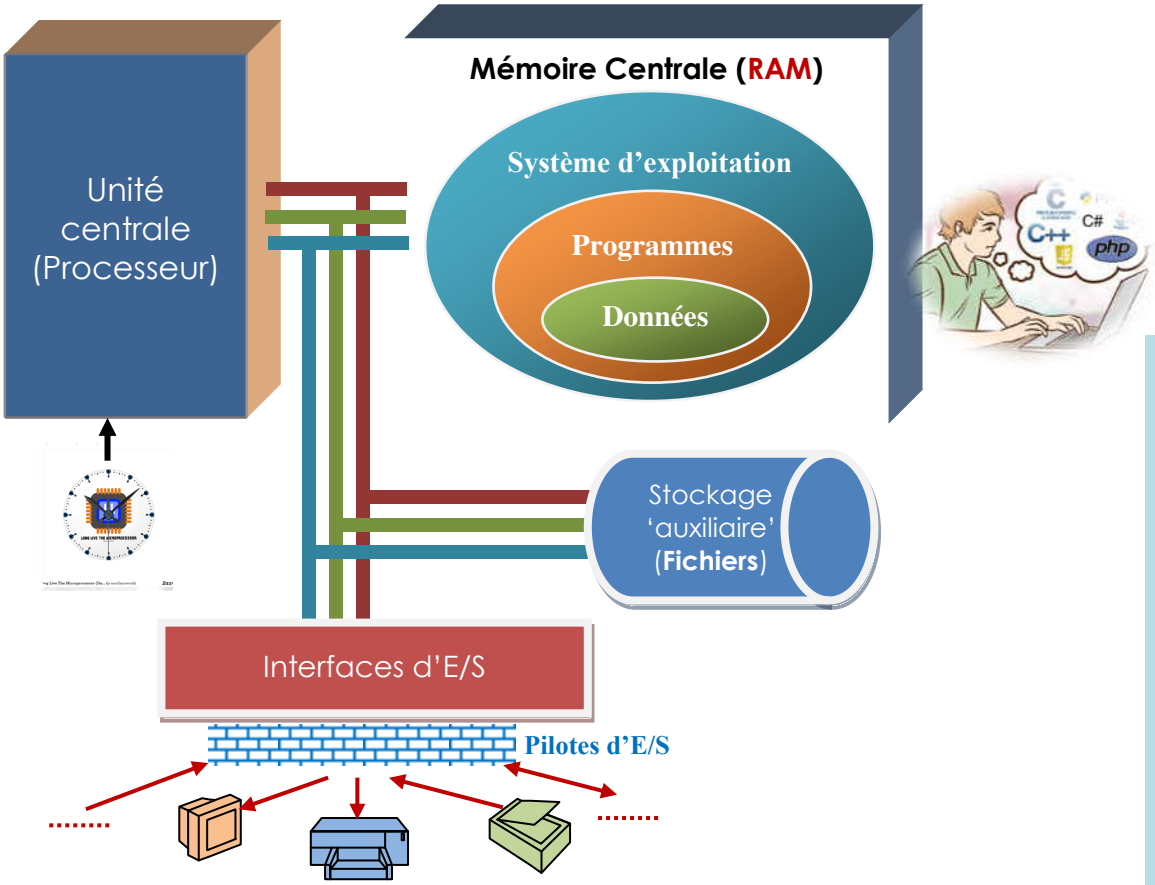


Figure 1.10 : *Structure logicielle et matérielle d'un système microprogramé.*

1.2. Architectures de Von Neumann et de Harvard

Pour l'organisation des différents blocs constituant un système microprogrammé, il existe deux architectures possibles:

- Architecture de Von Neumann.
- Architecture de de Harvard.

Dans l'architecture de Von Neumann, la mémoire programme, la mémoire données et les périphériques d'entrées/sorties partagent le même bus s'adresses et de données.

- Cette architecture a été introduite dans les années 50 à l'Institute for Advanced Study (IAS) de Princeton.
- L'architecture de Von Neumann nécessite plusieurs cycles d'horloge pour exécuter une instruction
  - ✓ Recherche d'instruction (Cycle Fetch ou initialisation),
  - ✓ Décodage et préparation de l'exécution,
  - ✓ Exécution effective.

- L'accès aux informations dans l'architecture de Von Neumann est assez facile puisque les programmes et les données peuvent être stockés dans la même zone mémoire.
- La majorité des structures microprogrammées utilisent l'architecture de Von Neumann.
- L'architecture de Von Neumann est maintenant principalement utilisée pour la conception des processeurs d'ordinateurs (PC, MAC).

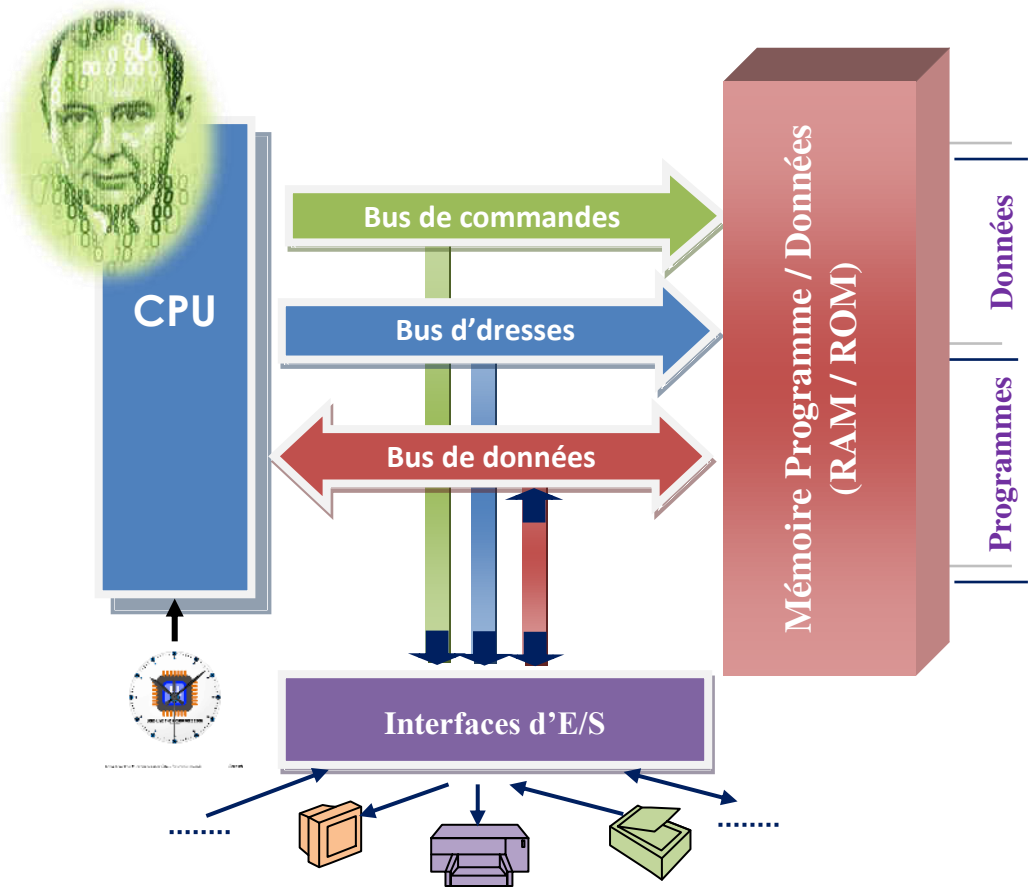


Figure 1.11 : *Architecture de Von Neumann.*

- L'architecture de type Harvard est une conception de systèmes microprogrammés qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chaque type de mémoires s'effectue via ses propres bus.

- L'architecture de Harvard est une évolution de l'architecture de Von Neumann.
- L'architecture de Harvard nécessite des instructions différentes pour accéder à la mémoire programme et à la mémoire de données.
- L'architecture de Harvard assure la rapidité de l'exécution des programmes (**PIPELINE**) puisque l'accès aux données et aux instructions est simultané malgré que l'architecture soit complexe.
- L'exécution d'une instruction ne fait plus appel qu'à un seul cycle machine.
- L'architecture de Harvard est très utilisée pour la conception des processeurs de traitement de signal (DSP) est de plus en plus pour les microcontrôleurs d'usage généraux qui ont connus un développement important ces dernières années.

Les microcontrôleurs PIC (Programmable Interface Controller) ainsi que bien d'autres structures sont construites autour de l'architecture de Harvard.

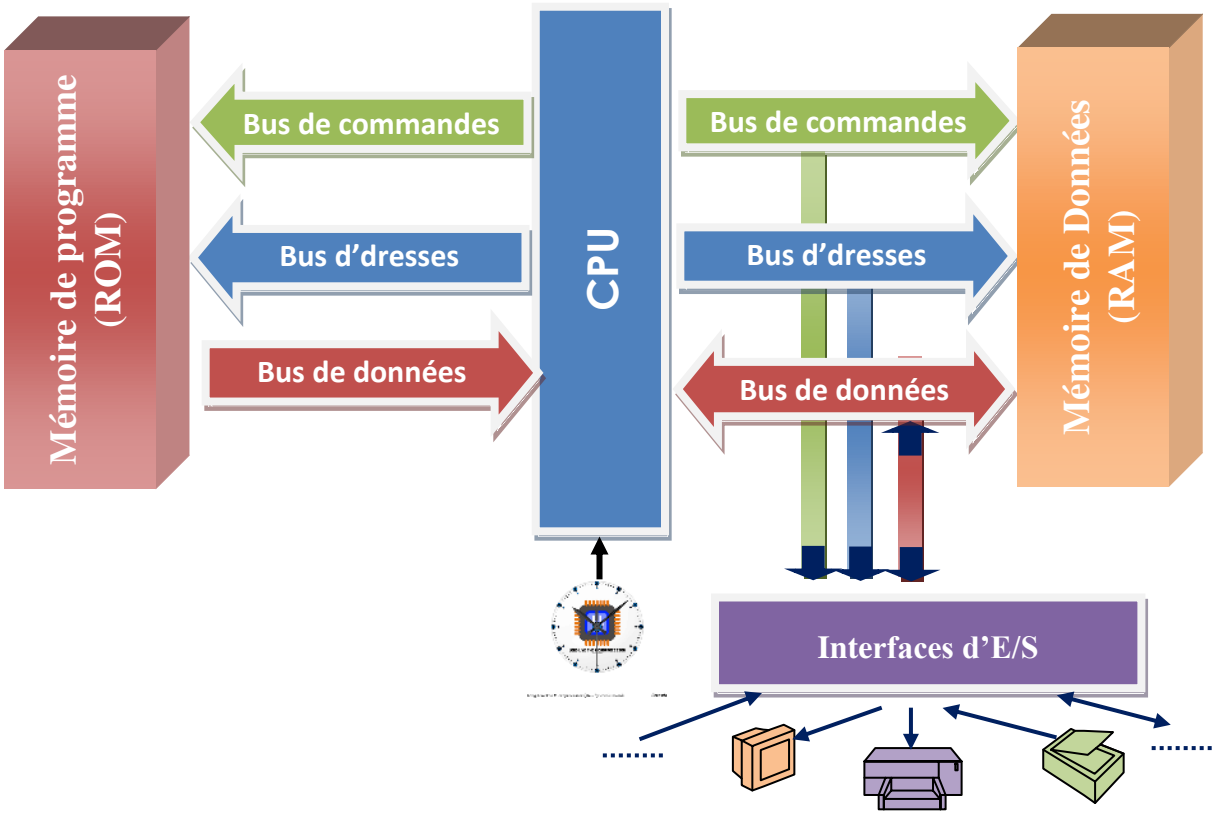


Figure 1.12 : *Architecture de Harvard.*

1.3. Processeurs de types CISC et RISC

Actuellement l'architecture des microprocesseurs se composent de deux grandes familles :

- ✓ Processeurs à jeu d'instruction étendu ou **CISC** (Complex Instruction Set Computing). Elle consiste à rajouter autant d'instructions que possible au système microprogrammé (processeur).
- ✓ Processeurs à jeu d'instructions réduit ou **RISC** (Reduced Instruction Set Computing). Elle consiste à minimiser le nombre d'instructions et à les simplifier dans un système microprogrammé (processeur).

Info

Dans les années 80, des chercheurs d'**IBM**, sous la direction de John Cocke se sont convaincus qu'un ensemble réduit d'instructions "**rapides/efficaces**" valait mieux qu'un plus **grand** ensemble d'instructions "**plus lentes et moins efficaces**".

Les processus RISC sont moins chers car moins complexes et ne comportent que des instructions très rapides toutes codées sur un même nombre d'octets. Elles sont plus simples et plus homogènes.

La technologie CISC domine le marché des **ordinateurs de bureau** et **des ordinateurs portables**.

- L'architecture RISC est, principalement, utilisée dans les imprimantes, routeurs domestiques, et les systèmes embarqués (smartphones, etc.).
- Les microprocesseurs RISC consomment moins d'énergie et donc ne sont pas soumis aux mêmes thermiques que les microprocesseurs CISC.
- L'architecture RISC a montré qu'elle était plus performante :
  - ✓ Tout les processeurs fonctionnent en RISC de nos jours.
  - ✓ Pour un processeur avec un jeu d'instruction CISC: Traduction interne des instructions CISC en micro-opérations de type RISC.

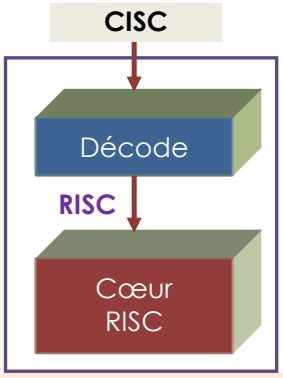


Tableau 1.1 : *Comparaison entre l'architecture RISC et CISC.*

Architecture RISC	Architecture CISC
Un nombre réduit d'instructions (~100) → Jeu d'instruction réduit.	Un nombre important d'instructions (~1000) → jeu étendu d'instructions complexes.
Format d'instructions fixe (toutes les instructions sont codées sur 4 octets).	Format d'instructions variable (Instructions codées sur 1 à 15 octets).
Instructions rapides (1 instruction = 1 cycle machine).	Certaines instructions sont parfois lentes (> 1 cycle).
Décodeur simple (câblé).	Décodeur complexe (microcode).
Trop de registres.	Peu de registres.
Peu de modes d'adressage. Opérations arithmétiques et logiques sur les registres *uniquement*.	Beaucoup de modes d'adressage. Opération arithmétiques et logiques à la fois sur des registres et de la mémoire.
seuls des tests, dont le résultat va dans des registres, sont utilisés lors des branchements conditionnels. **U**tilisation uniquement des registres pour les arguments des fonctions ainsi que pour l'adresse de retour.	Des drapeaux (registre d'état) sont positionnés et utilisés lors des branchements conditionnels. Utilisation intensive de la pile pour les arguments et pour l'adresse de retour.
Seules les instructions load et store ont accès à la mémoire.	Toutes les instructions sont susceptibles d'accéder à la mémoire.
Compilateur complexe.	Compilateur simple.

Info

- Quelques constructeurs d'architecture RISC :
  - ✓ Alpha (DEC),
  - ✓ ARM,
  - ✓ PowerPC (Motorola),
  - ✓ MIPS,
  - ✓ PA-RISC (Processeur RISC de Hewlett-Packard),
  - ✓ SPARC,
  - ✓ Microchip (8 à 16 bits),
  - ✓ … etc.

- Quelques constructeurs d'architecture CISC :
  - ✓ S/360 (IBM),
  - ✓ VAX (DEC),
  - ✓ 68xx, 680x0 (Motorola),
  - ✓ x86, Pentium (Intel),
  - ✓ … etc.

Info

- Deux paramètres sont souvent utilisés pour mesurer la performance d'un processeur :
  - ✓ **Latence: Le temps de réponse (temps d'éxécution d'une instruction) → Le temps qui s'écoule entre le début et la fin d'exécution de l'instruction.**
  - ✓ **Débit: Quantité total de travail réalisé en fonction du temps → Nombre d'opérations (tâche, instruction) exécutées par unités de temps.**

- L'amélioration du temps de réponse implique toujours une amélioration du débit. Par contre, le contraire n'est toujours vrai.
  - ✓ **Par exemple, augmenter le nombre de processus (instruction) augmentera le débit mais pas le temps d'exécution.**

1.4. Notions de pipeline

La notion de pipeline, autrement dit la chaîne de traitement des instructions, est l'élément d'un processeur dans lequel l'exécution des instructions est découpée en plusieurs étapes, c'est-à-dire le processeur peut commencer à exécuter une nouvelle instruction sans attendre que la précédente soit terminée. C'est une sorte **d'exécution parallèle** des instructions (Parallélisme).

- Le pipeline est inspiré des chaînes industrielles d'assemblage (A titre d'exemple, une chaîne d'assemblage d'automobiles);
- Dans une chaîne d'assemblage d'automobiles, il y a beaucoup d'étapes;
- Chaque étape contribue à une partie du montage de l'automobile;
- Chaque étape opère en parallèle avec les autres étapes, mais sur une autre voiture.

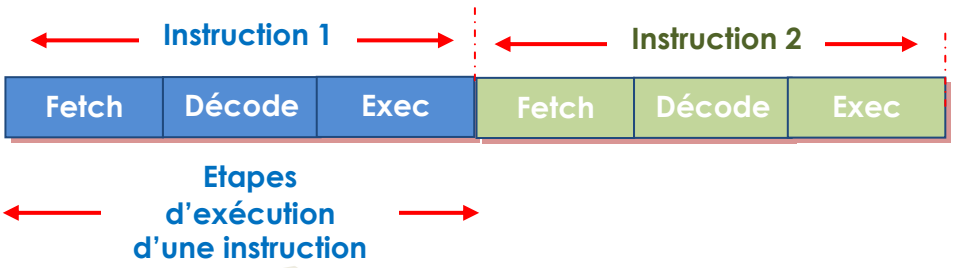
Info

- Dans un processeur conventionnel (sans pipeline), l'exécution des instructions s'effectue l'une après l'autre (une à la fois).
- Le pipeline est une technique utilisée pour optimiser le temps d'exécution d'un programme contenant plusieurs instructions (augmentation du nombre de tâches exécutées par unité de temps).
- Le pipeline est un mécanisme permettant d'accroître la vitesse d'exécution des instructions dans un micro-processeur.
- **Principe** : Ne pas attendre d'avoir finir toutes les étapes du traitement du 1er processus (instruction) pour commencer le 2ème, ainsi de suite.
- Le traitement des étapes s'effectue en parallèle (simultanément pour plusieurs instructions (processus).
- Les premières machines commerciales mettant en œuvre la notion de pipeline apparues avec l'IBM 360\91 en 1964.
- Chacune des étapes du pipeline est implémentée par un circuit intégré indépendant, appelé **étage**.
- Chaque étage correspond à une étape d'exécution.
- Le nombre d'étages d'un pipeline est appelé sa **profondeur**.

1.4.1. Processeur sans pipeline

Dans ce type de processeur, le traitement de l'ensemble des instructions (processus) s'effectue d'une manière normal non pipeliné ou séquentiel :

✓ Les instructions sont traitées les unes à la suite des autres.



✓ **Phase A = Cycle de recherche d'instruction (Fetch (IF));**
✓ **Phase B = Cycle de décodage et préparation de l'exécution (Décode (ID));**
✓ **Phase C = Cycle d'exécution effective de l'instruction (Exec (IE)) ;**
✓ **Instruction = 03 phases (A, B et C);**
✓ **Phase = ensemble des micro-opérations (µop) ;**
✓ **1 µop = 1 période d'Horloge ;**
✓ **1 Cycle machine = phase A + phase B + phase C**
= Temps d'exécution d'une instruction ;
✓ **L'enchaînement des phases est cadencé par les tops d'horloge.**
✓ **Chaque instruction possède son propre temps d'exécution ;**

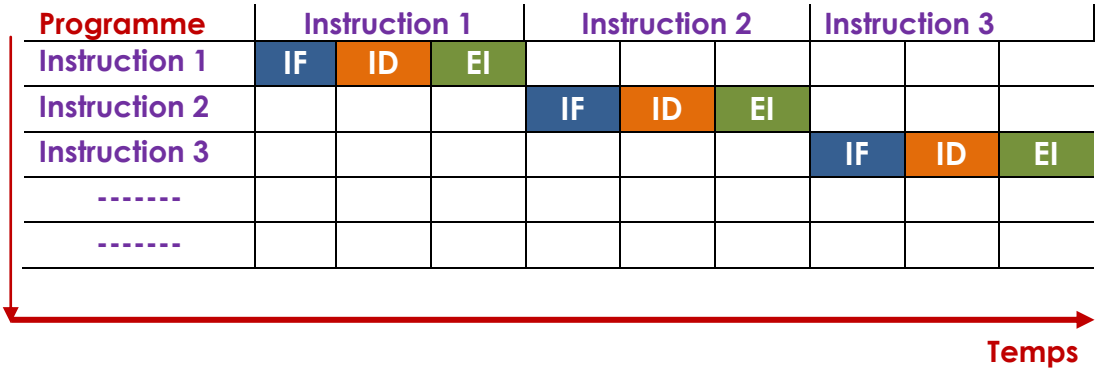
Tableau 1.2 : *Etapes d'exécution d'un processus (une instruction).*

Phase élémentaire	Micro opération	Description
(A)		
(A1)	**RA ← (CO)**	✓ L'adresse de l'instruction en cours d'exécution sera transférée du CO vers le registre d'adresse (RA).
(A2)	**CO ← (CO) + 1**	✓ Le CO s'incrémente et pointe vers la prochaine instruction à exécuter.
(A3)	**RM ← (RA)**	✓ Lecture de l'instruction à partir de l'adresse spécifiée par CO. Avant de l'accès, il faut passer par le registre RA. L'instruction récupérée sera transférée vers le registre de données RM associé à la MC.
(A4)	**RI ← (RM)**	✓ L'instruction récupérée sera transférée du registre de données RM vers le registre RI dans l'unité de commande.

Phase élémentaire	Micro opération	Description
(B)		
(B1)	**GS ← COP**	✓ Le code opération (COP) de l'instruction est transmis vers le générateur de séquences dans l'unité de commande qui s'occupe du décodage et de génération des commandes adaptées à cette instruction.
(B2)	**RA ← @ OP**	✓ La partie adresses des opérandes (@ **OP**) de l'instruction qui donne l'adresse des opérandes est transférée vers le registre RA associé à la MC.
(B3)	**RM ← (RA)**	✓ Lecture de l'opérande à partir de la partie des données de la MC et le transfert vers l'UAL pour l'exécution de l'instruction.

Phase élémentaire	Micro opération	Description
(C)		
(C)	**ACC ← (ACC) COP (RM)**	✓ Le code opération (COP) de l'instruction est transmis vers le générateur de séquences dans l'unité de commande qui s'occupe du décodage et de génération des commandes adaptées à cette instruction.

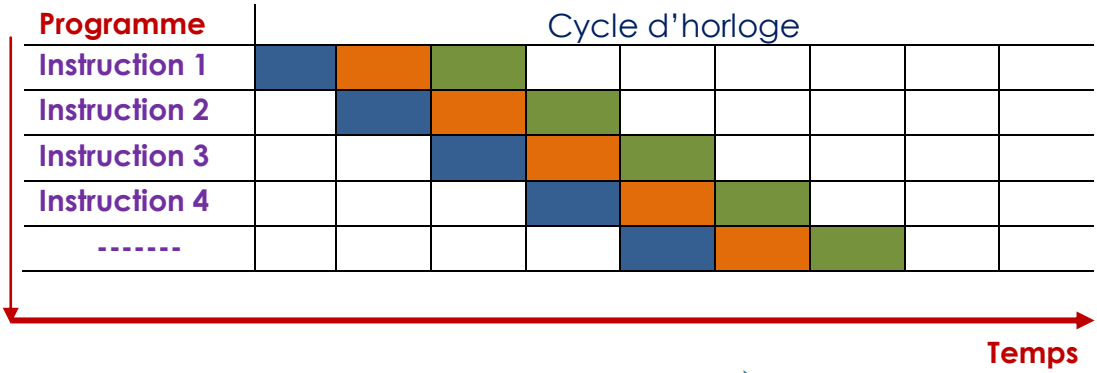
- ✓ Lorsque le traitement se trouve à une étape quelconque, les autres éléments sont inutilisés.
- ✓ Pas de recouvrement.

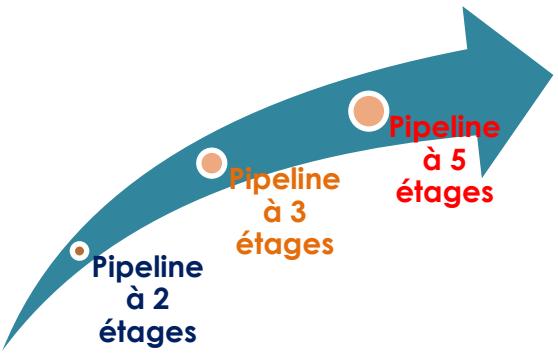


1.4.2. Processeur avec pipeline

Pour optimiser le temps d'exécution d'un programme contenant plusieurs instructions (processus) et d'augmentation du nombre de tâches exécutées par unité de temps, la technique de pipeline est utilisée.

- ✓ A chaque étape du processus est affectée une ressource indépendante, de façon à pouvoir exécuter plusierurs processus en parallèle où chacun à une étape différente.
- ✓ Exécution simultannée de différentes étapes sur des données distinctes.

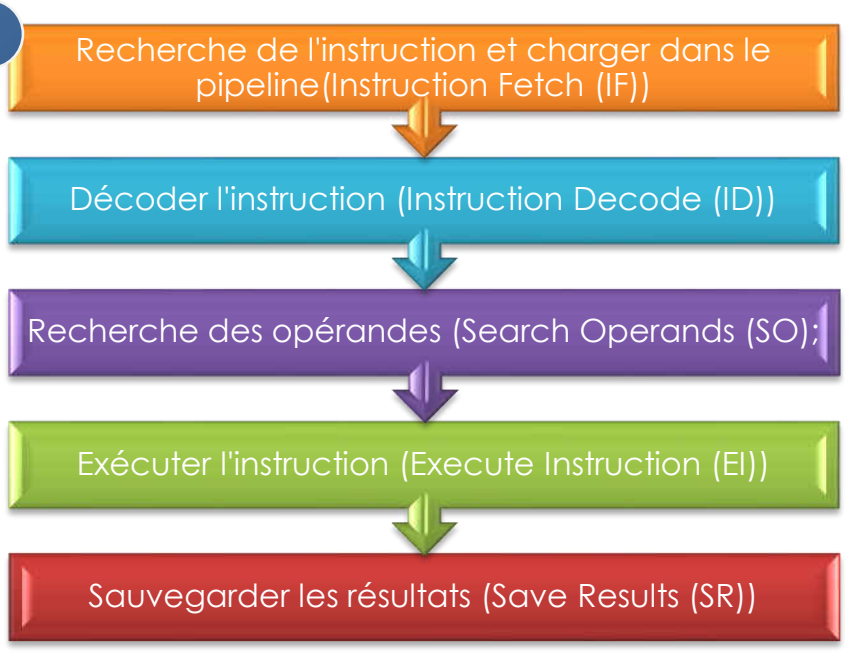




1.4.2.1. Pipeline à 5 étages

Ce type est appelé pipeline RISC classique (Classic RISC pipeline) crée par **David Patterson**, inventeur des processeurs RISC et du concept de pipeline. Avec ce pipeline, **5 étapes** (étages) sont nécessaires pour l'exécution ou le traitement d'une instruction.

David A. Patterson est un scientifique américain, professeur émérite de l'université de Californie à Berkeley en génie électrique et informatique. Lui et John Hennessy sont les co-lauréats du prix Turing 2017 pour avoir été les pionniers d'une approche systématique et quantitative de la conception et de l'évaluation d'architectures informatiques ayant un impact durable sur l'industrie des microprocesseurs.



Programme	Cycle d'horloge								
	1	2	3	4	5	6	7	8	9
Instruction 1	IF	ID	SO	EI	SR				
Instruction 2		IF	ID	SO	EI	SR			
Instruction 3			IF	ID	SO	EI	SR		
Instruction 4				IF	ID	SO	EI	SR	
Instruction 5					IF	ID	SO	EI	SR

- On remarque que pour 5 instructions, le temps d'exécution est réduit de 25 unités de temps à 9 unités de temps.
- Dans certains cas les 5 étages ne sont pas obligatoires, particulièrement dans le cas de chargement en mode immédiat qui ne nécessite pas l'étage de recherche de l'opérande en mémoire (SO).
- Pour simplifier le hardware du pipeline, le temps d'exécution est calculé en supposant que chaque instruction nécessite les cinq étages.

1.4.2.2. Pipeline à 2 étages

Une organisation alternative au pipeline RISC classique (à 5 étages) serait de découper l'exécution des instructions en deux étapes :

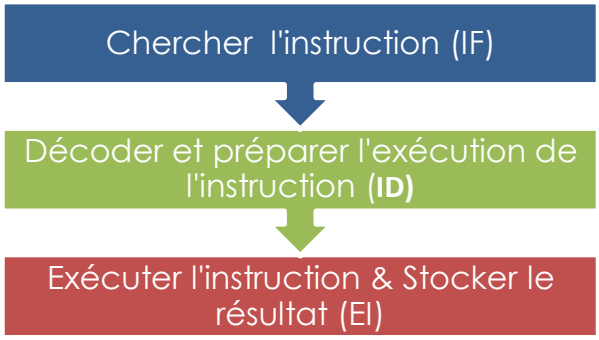
- ✓ Cycle Fetch (IF), qui charge et décode une instruction, et qui met à jour du pointeur d'instruction (Program Counter), c'est à dire pointer vers la prochaine instruction à exécuter.
- ✓ Exécution effective (EI) de l'instruction en cours avec stockage des résultats dans la mémoire ou dans les registres.

Programme	Cycle d'horloge					
	1	2	3	4	5	6
Instruction 1	IF	ID				
Instruction 2		IF	ID			
Instruction 3			IF	ID		
Instruction 4				IF	ID	
Instruction 5					IF	ID

Ce type de pipeline est notamment utilisé sur certains **microcontrôleurs Atmel AVR et PIC**.

1.4.2.3. Pipeline à 3 étages

Ce type de pipeline est celui de l'IBM Stretch project composé des trois étapes :



En supposant que chaque étape met 1 cycle d'horloge pour s'exécuter, il faut normalement 3 cycles pour exécuter une instruction, 9 pour 3 instructions :

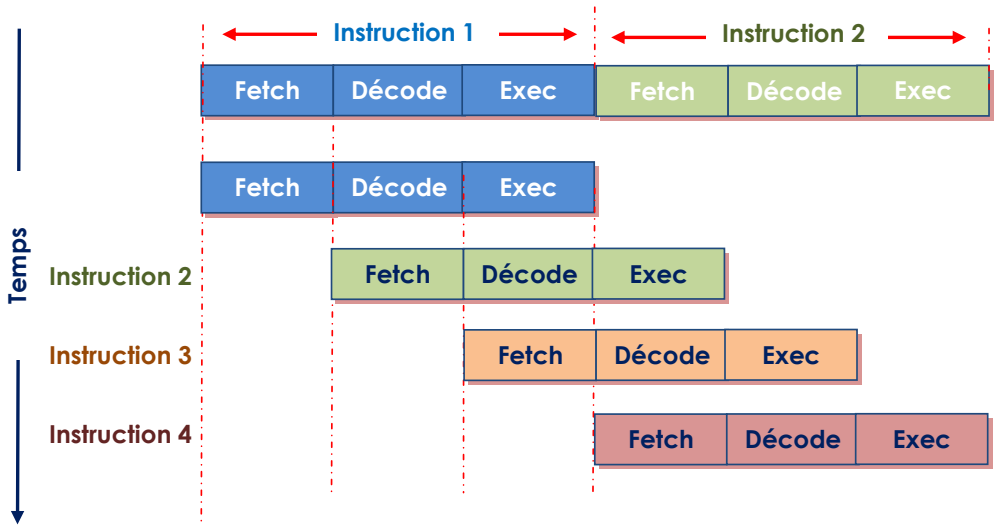


Figure 1.13 : *Chaine de traitement (Pipeline à 3 étages).*

Programme	Cycle d'horloge						
	1	2	3	4	5	6	7
Instruction 1	IF	ID	EI				
Instruction 2		IF	ID	EI			
Instruction 3			IF	ID	EI		
Instruction 4				IF	ID	EI	
Instruction 5					IF	ID	EI

Actuellement, tous les microprocesseurs commercialisés sont équipés par des étages de pipelines (profondeurs) comme illustre le tableau ci-dessous.

Tableau 1.3 : *Etages de pipelines (profondeurs) pour quelques microprocesseurs.*

Microprocesseur	*Profondeur*
Intel Pentium 4 Prescott	31
Intel Pentium 4	20
AMD K10	16
IBM POWER5	16
IBM PowerPC 970	16
Intel Core 2 Duo	14
Intel Pentium II	14
Sun UltraSPARC IV	14
Sun UltraSPARC IIi	14
AMD Opteron 1xx	12
AMD Athlon	12
IBM POWER4	12
Intel Pentium III	10
Intel Itanium	10

MIPS R4400	8
Motorola PowerPC G4	7

- Lorsque le nombre d'étages (profondeur) devient important (pouvant aller jusqu'à 31 pour la microarchitecture Prescott d'Intel), cette architecture sera appelée **superpipelinée**.

Les principaux problèmes rencontrés dans les architectures pipelines sont liés :

- Aux accès à la mémoire (ex : accès au même bus)
- Aux conflits de dépendance entre instructions (ex : lorsque le contenu d'un même registre est requis par deux instructions successives).
- Aux JUMP (si) et aux traitements des interruptions et exceptions (ex : il faut finir le traitement avant avant d'exécuter le « si », voir vider le pipeline)
- Tous ces problèmes peuvent être résolus soit matériellement, soit par un logiciel.

Afin d'améliorer la structure interne d'un microprocesseur, différentes procédés ont été mis en place :

- Les architectures C.I.S.C. et R.I.S.C.
- La mémoire cache.
- L'architecture Pipeline.
- L'architecture Superscalaire.
- L'architecture Hyperthreadind.
- L'architecture Bicoeur.
- L'architecture NetBurst.
- L'architecture EPIC.
- Le concept du « PC on a chip ».

1.5. Microprocesseur ou microcontrôleur ?

Les microprocesseurs et les microcontrôleurs sont des puces électroniques programmables typiques utilisées à des fins différentes.

- La différence principale entre eux est qu'un microprocesseur est un moteur de calcul programmable constitué d'une unité arithmétique et logique, d'un processeur et de registres, capable d'effectuer des calculs et de prendre des décisions.
- Par contre, un microcontrôleur est un microprocesseur spécialisé considéré comme un ordinateur sur une puce (**System On Chip**). Il contient
  - Un CPU,
  - De la RAM (mémoire de données),
  - De la ROM (mémoire programme) et
  - Des ports d'E/S.

  Il comporte aussi des fonctions spécifiques (modules annexes) comme
  - Des compteurs programmables pour effectuer des mesures de durées (Timers),
  - Des convertisseurs analogiques/ numériques (CAN) (voir des CNA),
  - … etc.

Microcontrôleur = Microprocesseur + RAM + ROM + Interfaces E/S

Circuits intégrés dans le même boîtier

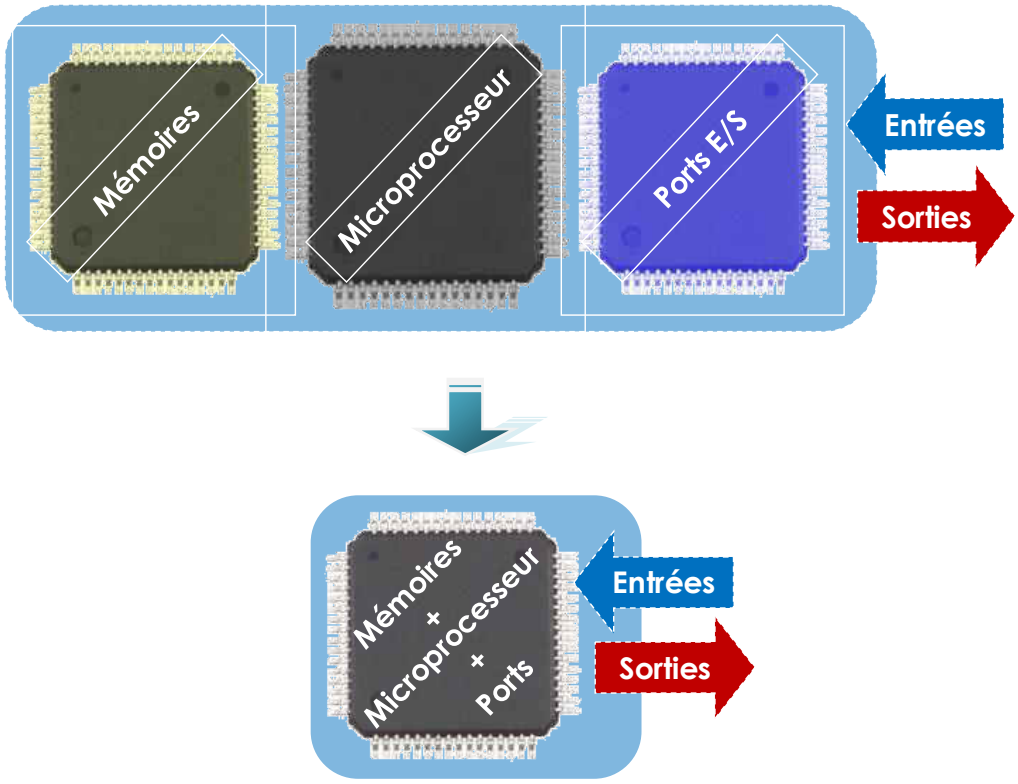
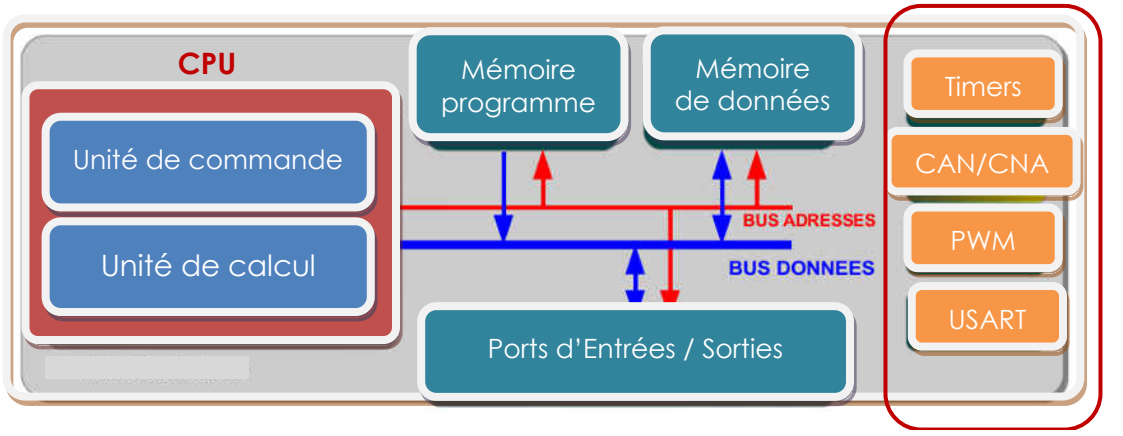
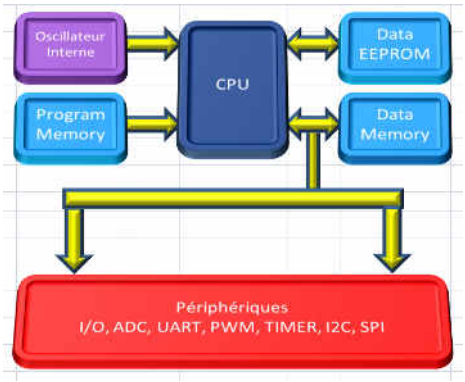


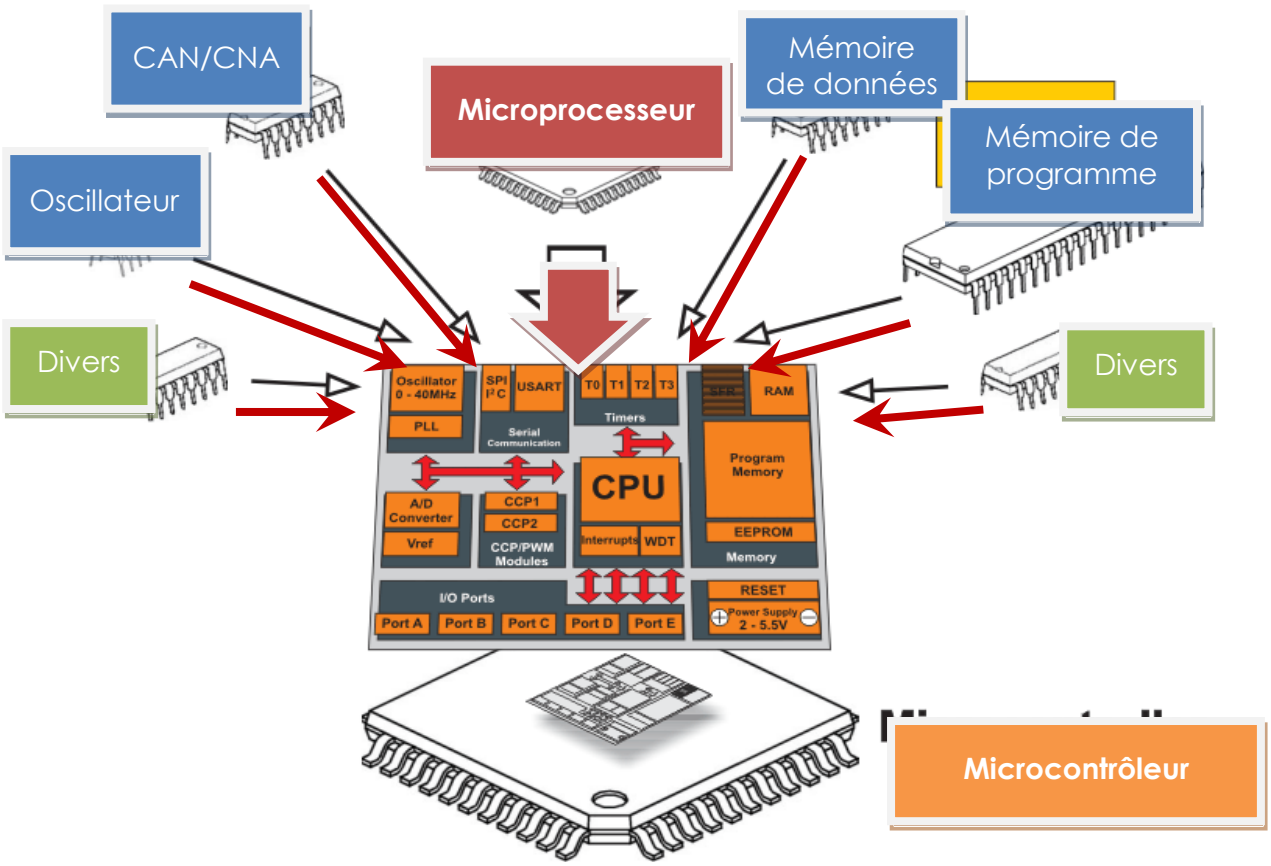
Figure 1.14 : *Différence entre microprocesseur et microcontrôleur.*



(a)



(b)



(c)

Figure 1.15 : *Différences structurelles entre microprocesseur et microcontrôleur.*

Info

Le microcontrôleur est comme un mini-ordinateur doté d'un processeur, de RAM, de ROM, de ports série, de timers et des périphériques E/S, le tout intégré sur une seule puce (System On chip). Un microcontrôleur est donc un composant autonome, et qui peut fonctionner sans l'addition de circuits externes. En termes simples, il s'agit d'un processeur entièrement fonctionnel sur un seul circuit intégré utilisé par un système informatique pour effectuer son travail.

Carte mère – Ordinateur
(Microprocesseur + Mémoires + Circuits électroniques)

Microcontrôleur
Regroupe dans une seule puce (Microprocesseur + Mémoires + Circuits électroniques)

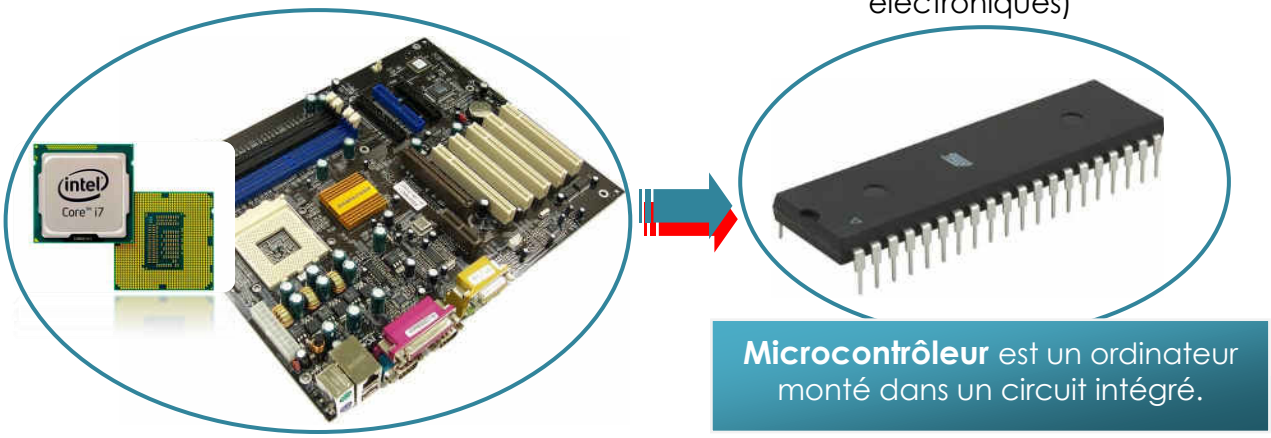


Figure 1.16 : *Différence entre microprocesseur et microcontrôleur.*

Tableau 1.2 : *Comparaison entre microprocesseur et microcontrôleur.*

Microprocesseur	Microcontrôleur
Le microprocesseur est le cœur d'un système informatique.	Le microcontrôleur est le cœur d'un système embarqué (Embedded System).
C'est juste un processeur. Les composants de mémoire et d'E/S doivent être connectés en externe.	Le microcontrôleur dispose d'un processeur externe ainsi que de composants de mémoire interne et d'E/S.
Étant donné que la mémoire et les E/S doivent être connectés en externe, le circuit devient volumineux.	Puisque la mémoire et les E/S sont présents en interne, le circuit est petit.
Ne peut pas être utilisé dans des systèmes compacts et donc inefficace.	Peut être utilisé dans des systèmes compacts et constitue donc une technique efficace.
Le coût de l'ensemble du système augmente.	Le coût de l'ensemble du système est faible.
Principalement utilisé dans les ordinateurs personnels.	Utilisé principalement dans des applications spécifiques, à savoir, les machines à laver, les lecteurs MP3, etc.
Les microprocesseurs sont basés sur l'architecture von **Neumann** dans laquelle le programme et les données sont stockés dans le même module de mémoire.	Les microcontrôleurs sont basés sur l'architecture de **Harvard** où la mémoire du programme et la mémoire de données sont séparées.
Le microprocesseur a moins de registres, donc plus d'opérations sont basées sur la mémoire.	Les microcontrôleurs ont un grand nombre de registres, ce qui facilite l'écriture des programmes.
La plupart des microprocesseurs ne disposent pas des fonctions d'économie d'énergie.	La plupart des microcontrôleurs ont des modes d'économie d'énergie comme le mode veille et le mode économie d'énergie. Cela aide à réduire encore plus la consommation d'énergie.
Les systèmes à microprocesseur sont plutôt réservés pour les applications demandant beaucoup de traitement de l'information et assez peu de gestion d'entrées / sorties. Les ordinateurs sont réalisés avec des systèmes à microprocesseur.	Les microcontrôleurs sont plutôt dédiés aux applications **spécifiques** qui ne nécessitent pas une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées/sorties. C'est le cas de contrôle de processus

Les applications à base du microcontrôleur offrent les avantages suivants:

- ✓ Diminution de l'encombrement du matériel et du circuit imprimé.
- ✓ Simplification du tracé du circuit imprimé.
- ✓ Fiabilité ''assuré'' du système.
- ✓ Consommation réduite de l'énergie.
- ✓ Coûts réduit de main d'œuvre (câblage) : Conception et montc
- ✓ Intégration dans un seul boîtier.
- ✓ Environnement de développement et de simulation évolués.

Inconveniants

Les applications à base du microcontrôleur souffrent des inconvénients suivants:

- ✓ L'intégration de nombreux périphériques, de RAM, de ROM limite la puissance de calcul et la vitesse de ces circuits (les transistors intégrés sont destinés aux périphériques et non plus au calcul).
- ✓ Mise en œuvre et approche du composant d'apparence complexe.

Application

Parmi les applications à base des microcontrôleurs, on trouve :

- ✓ Informatique et télécommunication (souris, modems, etc.).
- ✓ Contrôle de processus industriels (régulation, poursuite).
- ✓ Vidéo (Appareil photos et caméscopes numériques, et.).
- ✓ Multimédia (téléviseur, carte audio, carte vidéo, MP3, etc.).
- ✓ Systèmes embarqués (contrôleurs des moteurs automobiles (ABS, injection, GPS, airbag)).
- ✓ Systèmes de télécommandes.
- ✓ La téléphonie mobile.
- ✓ … etc.

1.6. Différentes familles des microcontrôleurs

De nos jours, il existe plusieurs types de microcontrôleur. Les fabricants les plus connus sont mentionnés dans le tableau ci-dessous.

Tableau 1.3 : *Différentes familles des microcontrôleurs.*

Fabricant	Logo	Série	Famille
Microship	MICROCHIP	PIC	PIC10xxx, PIC12xxx, PIC16xxx, PIC24xxx, PIC32xxx

Atmel		AVR	Atmega, AT90, AVR32
Intel		MCS-51	80C50, 80C31
Toshiba			TX19A
Zilog			Z180, Z80
Analog Devices			ADuC
STMicroelectronics		STX	ST6, ST7, STR7, STR9
Philips			LPC21xx ARM7-TDMI
Texas Instruments			MSP430
NEC			V800, KO
cypress			V800, KO

1.7. Critères de choix du microcontrôleur

Il existe un très grand nombre de modèles de microcontrôleurs, où de nombreux fabricants proposent chacun plusieurs familles (comptant parfois de centaines de modèles). Ces fabricants proposent aussi des microcontrôleurs allant de petits circuits à 6 pattes (faible prix) jusqu'à des circuits comptant de centaines de pattes.

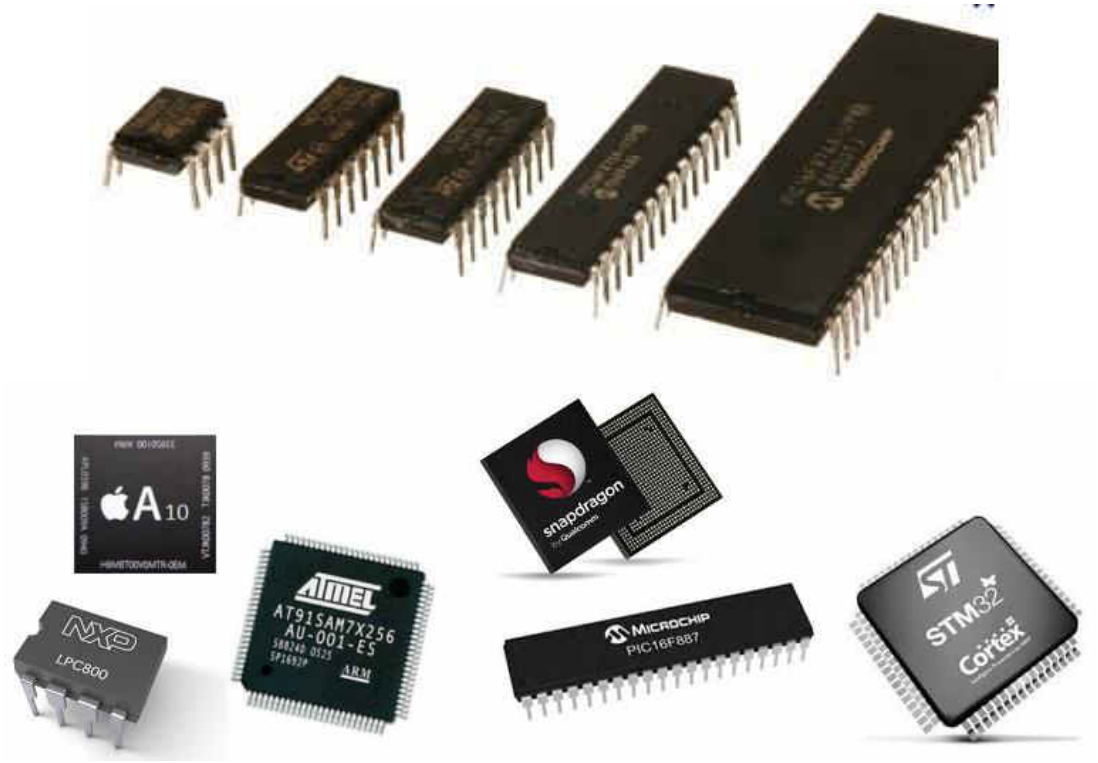


Figure 1.17 : *Quelques modèles de microcontrôleurs.*

Tableau 1.3 : *Critères de choix des microcontrôleurs.*

Critère	Caractéristiques
Critères techniques	Nombre de broches du circuit.
	Taille de la mémoire de programme.
	Taille de la mémoire de données.
	Besoin en puissance de calcul : famille 8, 16 ou 32 bits.
	Type de boîtier et caractéristiques de consommation (Tension d'alimentation, courant de consommation).
	Périphériques embarqués.
	Taille de bus de données (capacité de traitement de système, format du mot traité).
	Taille de bus d'adresses
	Vitesse d'horloge.
Autres critères	Coût et disponibilités : ▪ Coût de composant. ▪ Coût de système de développement. ▪ Disponibilité des ressources internet.
	Environnement de développement –Outils - (Matériel et logiciel): ▪ Compilateurs dédiés, ▪ Simulateurs,

	▪ Debuggers en circuits, ▪ Émulateurs). Connaissance du système. L'expérience. Type d'applications : ▪ Utilisation pédagogiques. ▪ Utilisation professionnelle.

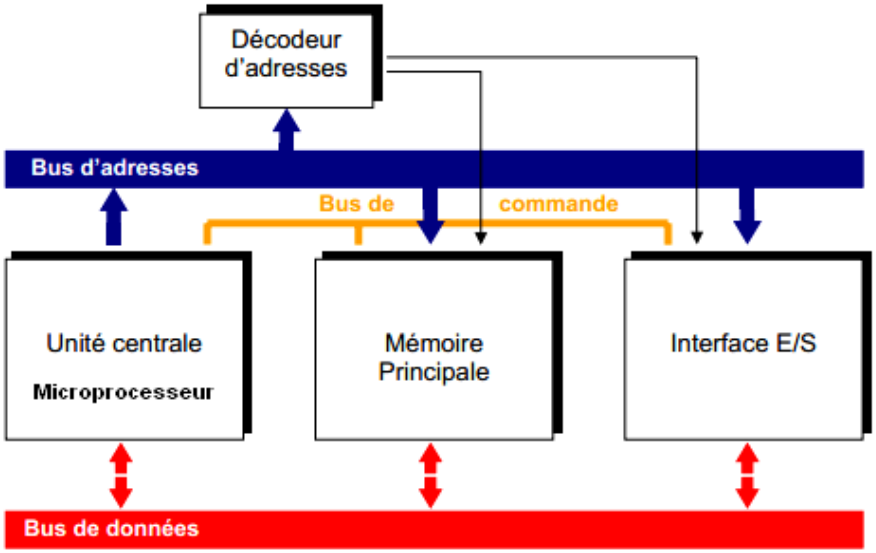
EXERCICE 1.1

- ✓ Qu'est-ce qu'un bus ?
- ✓ Quelles sont les différences fondamentales entre les langages machine et les langages évolués ?
- ✓ Décrire les cycles de recherche et d'exécution d'une instruction.
- ✓ Quelle est la différence entre un séquenceur câblé et un séquenceur micro-programmé?

EXERCICE 1.2

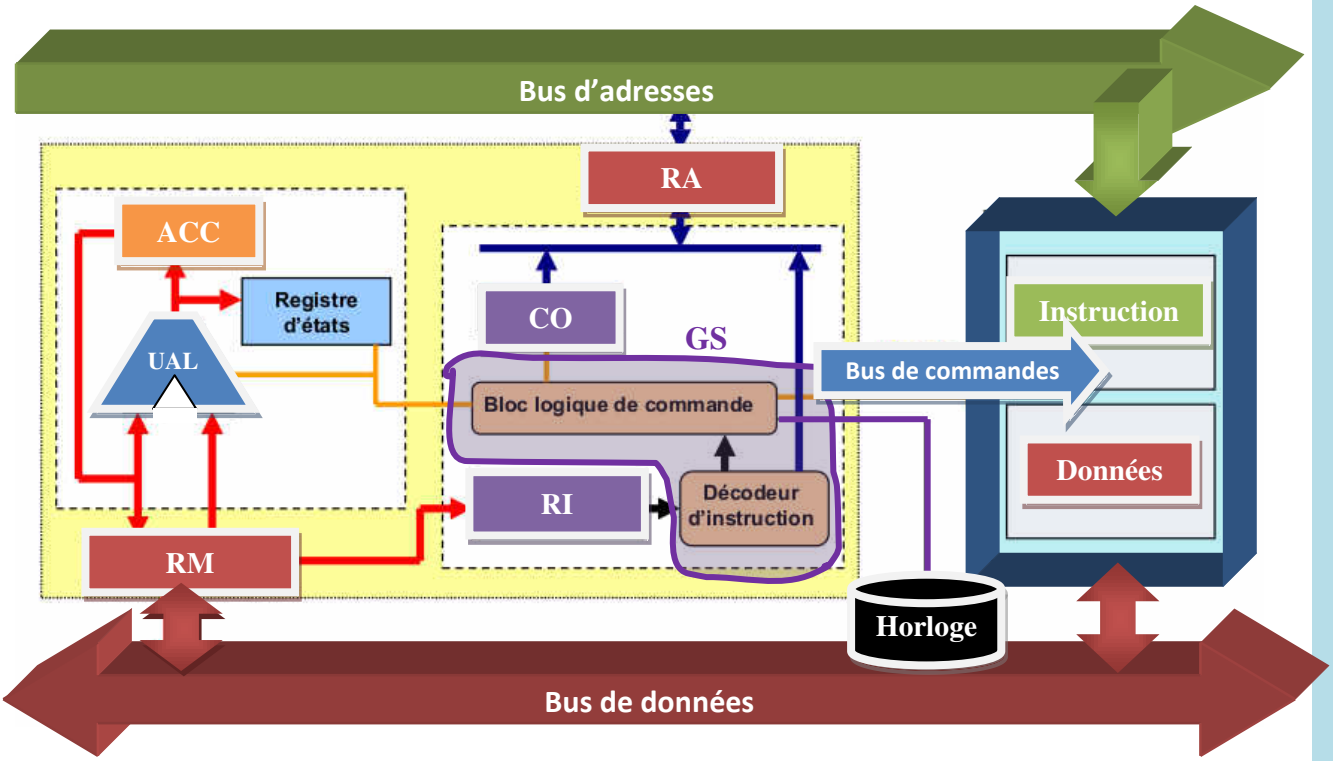
On considère le système de la figure 1.

1. Que représente ce schéma ?
2. Expliquer le rôle de chaque partie.
3. Que peut-on conclure quant à la taille du bus de données ?
4. Que peut-on conclure quant à la taille du bus d'adresses ?



EXERCICE 1.3

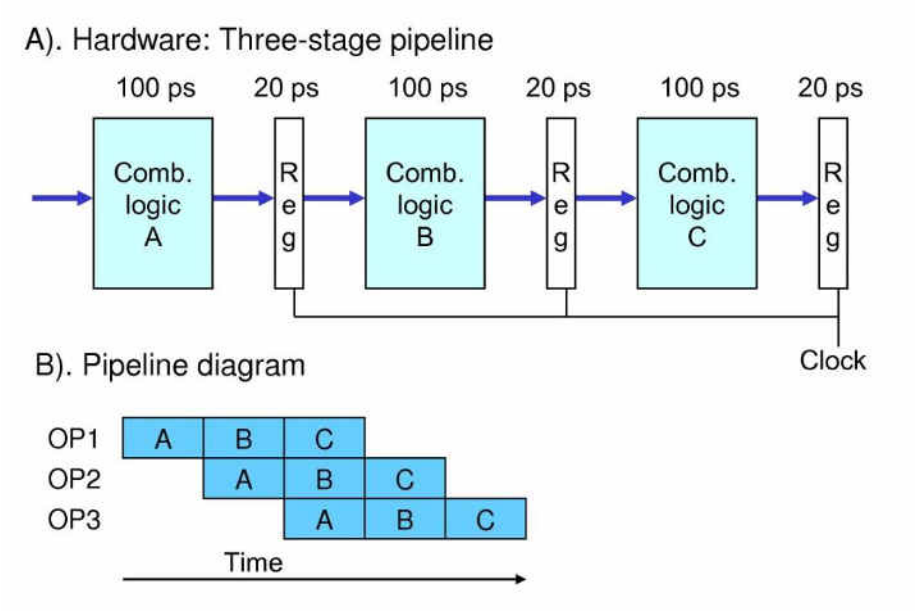
On considère le schéma synoptique ci-dessous.



- ✓ Quel est le rôle de chaque unité ?
- ✓ Donner les différentes étapes nécessaires pour l'exécution d'une instruction. Discuter…

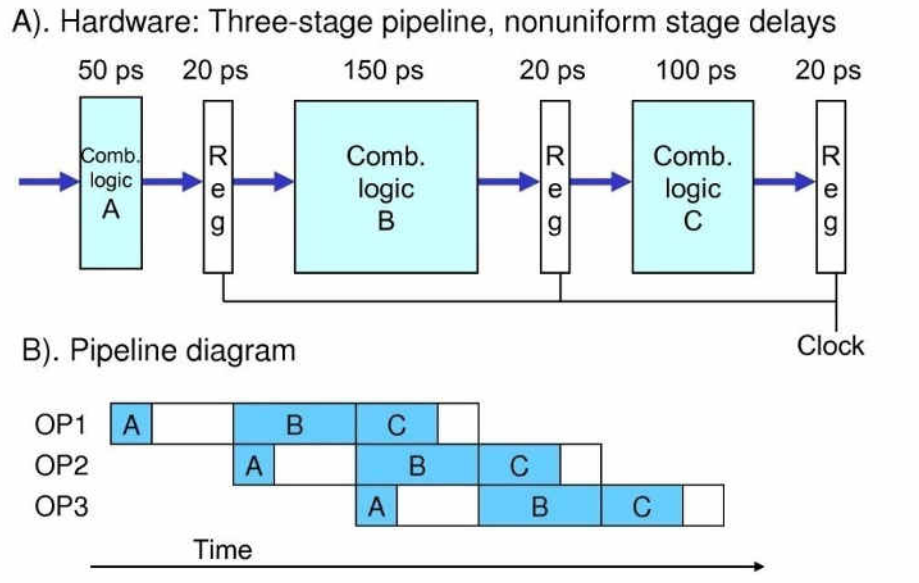
EXERCICE 1.4

1. On considère le schéma synoptique d'un pipeline à 3 étages où l'unité de temps est la pico-seconde (ps) qui vaut 10^{-12} seconde.



- ✓ Quelle est la durée minimale du cycle d'horloge ?
- ✓ Quel est le débit maximal de ce pipeline, en Gops (Giga-opérations par seconde) ?
- ✓ Quelle est dans ce cas la latence, c'est-à-dire la durée d'exécution d'une instruction ?

2. Mêmes questions pour le schéma ci-contre (les durées de traitement des trois circuits combinatoires sont différentes).



EXERCICE 1.5

On suppose qu'on a découpé les circuits combinatoires qui composent une instruction en six blocs *A* à *F* de durées respectives 80, 60, 30, 50, 70 et 10 ps ; ces blocs doivent être exécutés l'un après l'autre dans cet ordre, après quoi on charge un registre au prochain front d'horloge. La durée de chargement d'un registre est de 20 ps.

- ✓ Insérer un seul registre intermédiaire fournit un pipeline de profondeur 2. Où faut-il insérer ce registre pour obtenir un débit maximal ? Calculer alors la durée du cycle d'horloge, le débit et la latence.
- ✓ Mêmes questions en insérant deux registres intermédiaires (pipeline de profondeur 3).

- ✓ Quel est le pipeline de profondeur optimale ? Fournir une description et une analyse des performances comme précédemment.
- ✓ Echanger les durées de *B* et *C* et traiter à nouveau la question précédente.

EXERCICE 1.6

Un processeur non pipeliné possède un temps de cycle de 10 ns.

- ✓ Quels seront les temps de cycle des versions pipelinées du processeur avec un pipeline de 2, 4, 8 et 16 étages, si la logique de chemin de données est répartie de manière égale entre les étages du pipeline (on considère que le temps de stabilisation après le passage dans chaque étage est de 0,5 ns) ?
- ✓ En outre quel est le temps d'exécution d'une instruction complète pour chacune des versions pipelinées ?

EXERCICE 1.7

Supposons qu'un processeur non pipeliné possède un temps de cycle de 25 ns et que le chemin de données est constitué de modules dont les temps d'exécution sont respectivement 2, 3, 4, 7, 3, 2 et 4 ns (dans cet ordre). Il est alors possible de mettre en place un pipeline à 7 étages correspondant à ces 7 modules. On suppose que le temps de stabilisation après le passage dans un étage est de 1 ns.

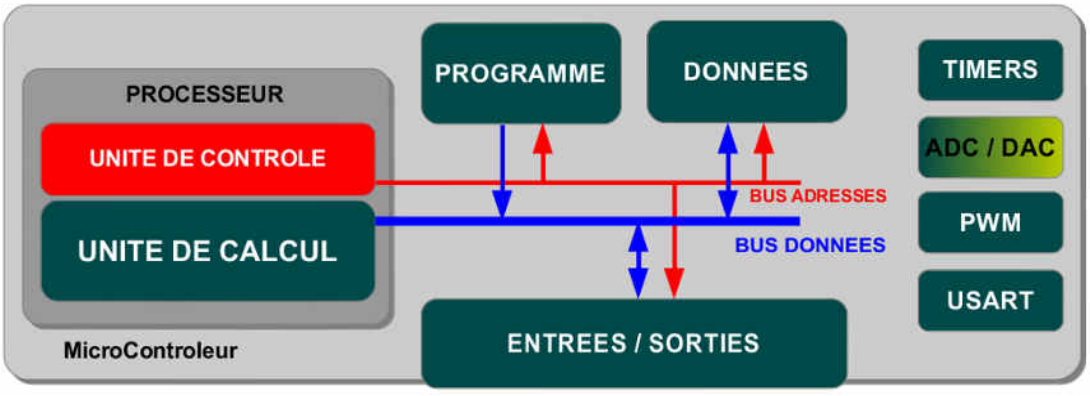
- ✓ Quel est le temps de cycle minimal qui peut être atteint en implémentant le pipeline sur ce processeur ?

EXERCICE 1.8

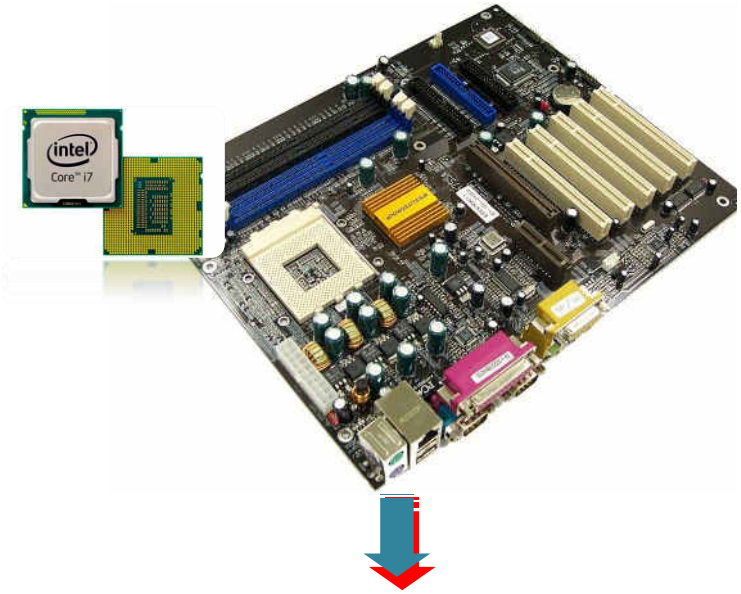
- ✓ Expliquer en quelques lignes le principe de pipelining ?
- ✓ On suppose que l'on dispose d'un processeur pipeliné à 5 étages dont le temps de cycle est de 0.4ns. Combien de temps faut-il pour exécuter les 101 premières instructions en supposant qu'il n'y a aucun aléa ?

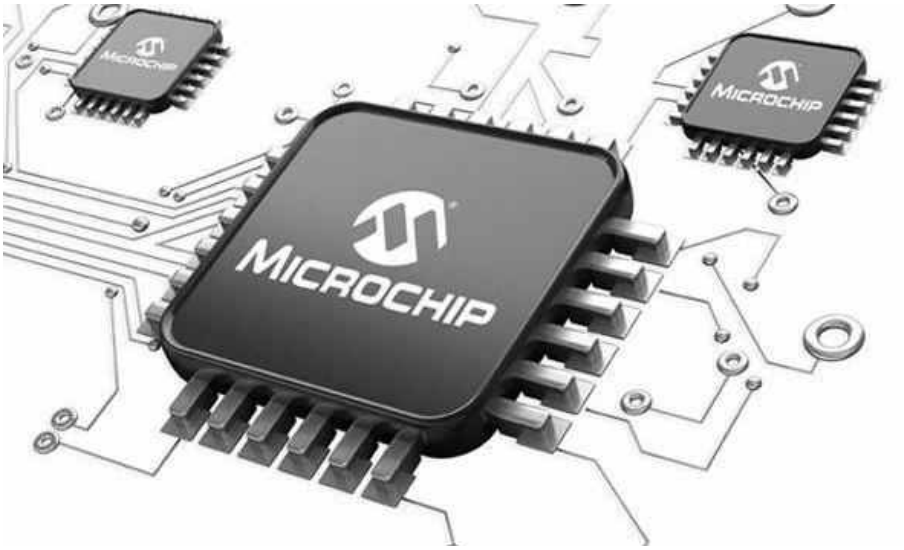
EXERCICE 1.9

On considère le schéma synoptique ci-dessous



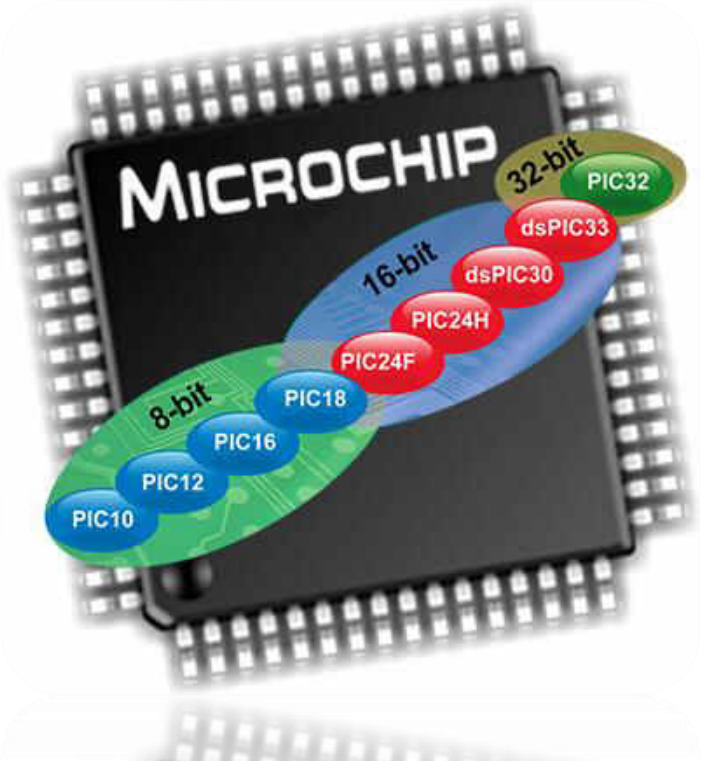
- ✓ Quel est le rôle de chaque unité ?
- ✓ Expliquer brièvement le fonctionnement de chaque unité de la figure ci-dessous. Citer quelques domaines d'applications. Discuter…
- ✓ Donner les avantages et les inconvénients. Expliquer…

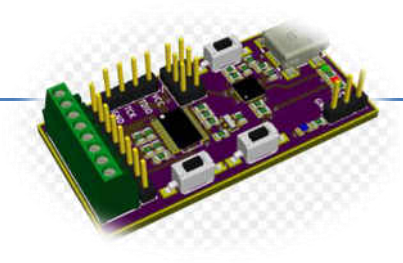

intel
Core i7



CHAPITRE 2

Architecture du microcontrôleur

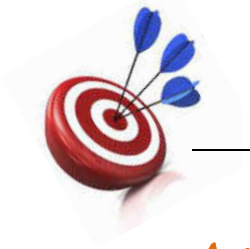




CHAPITRE 2

Architecture matérielle du PIC 16F84

Objectifs



- L'objectif est de
  - ✓ Présenter d'une manière profonde la structure interne du microcontrôleur PIC 16F84.
  - ✓ Décrire les fonctionnalités et les concepts associés au PIC 16F84.
  - ✓ Etudier la programmation du PIC 16F84.

Avant Propos

Dans ce chapitre, les microcontrôleurs PIC de Microship sont étudiés. Particulièrement le fameux PIC 16F84 de la famille Mid-Range. Une fois le PIC 16F84 est assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur. Plusieurs facteurs intervenant dans le choix du microcontrôleur PIC de la société Microship, à savoir :

- Le jeu d'instruction réduit et facile à maîtriser (Programmation facile).
- La disponibilité sur le marché avec des prix très bas.
- Grande utilisation professionnelle / pédagogique.
- Les versions avec mémoire flash présentent une souplesse d'utilisation et des avantages pratiques indéniables.
- Les outils de développement (soft) sont gratuits et téléchargeables sur le WEB : http://www.microchip.com/ (Offre logicielle (IDE, compilateur C)).
- Les performances sont identiques voir supérieurs à ses concurrents.
- Documentation riche sur le Web.

2.1. Introduction aux microcontrôleurs PIC

Un **microcontrôleur** est un circuit microprogrammé capable d'exécuter un programme et qui possède des circuits intégrés d'interface avec le monde extérieur.

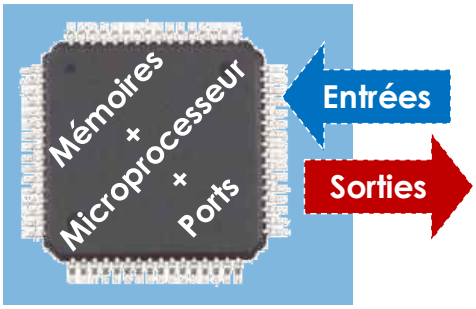


Figure 2.1 : *Equivalence Microcontrôleur-Microprocesseur.*

Un PIC (abréviation de l'anglais **P**eripheral **I**nterface **C**ontroller) est un microcontrôleur marque de la société ***Microchip***, construit selon l'architecture ***Harvard***. Le logo de la société est donné par :

Figure 2.2 : *Logo du constructeurs des microcontrôleurs PIC.*

Généralement, les PIC sont caractérisés par:

- ✓ Séparation des mémoires de programmes et de données (Architecture de Harvard).
- ✓ Communication avec l'extérieur seulement via des ports d'entrées/sorties.
- ✓ Utilisation d'un jeu d'instructions réduit RISC (Reduced instruction set computer).

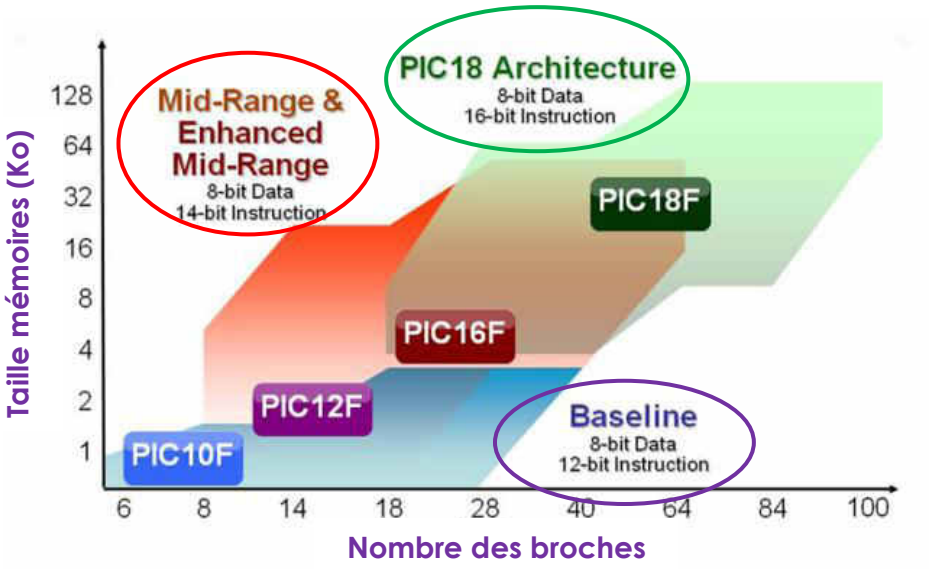
2.1.1. Classification des PICs de Microchip

Actuellement les modèles **Microchip 8 bits** (Tailles des registres internes de données est égale 8 bits), sont classés en 3 grandes familles :

- ✓ **Base-Line** : Les instructions sont codées sur 12 bits.
- ✓ **Mid-Line** : Les instructions sont codées sur 14 bits.
- ✓ **High-End** : Les instructions sont codées sur 16 bits.

Tableau 2.1 : *Familles des microcontrôleurs PIC.*

Famille	Taille du mot d'instruction	Nombre d'instructions	Exemples de circuits	Vecteurs d'interruptions
Base-Line	**12 bits**	**33**	**10F200, 12F508, 16F57**	**/**
Mid-Range	**14 bits**	**35**	**12F609, 16F84A, 16F631, 16F873A**	**1**
High-End	**16 bits**	**75**	**18F242, 18F2420**	**2**



(b)

Figure 2.3 : *Caractéristiques structurelles des microcontrôleurs PIC 8 bits.*

Malgré qu'il y a plusieurs modèles de famille de PIC, comme indique la figure ci-dessous, tous ces microcontrôleurs ont quelques caractéristiques en communs représentées. Par contre, certains microcontrôleurs présentent plus de caractéristiques illustrées dans le tableau suivant :

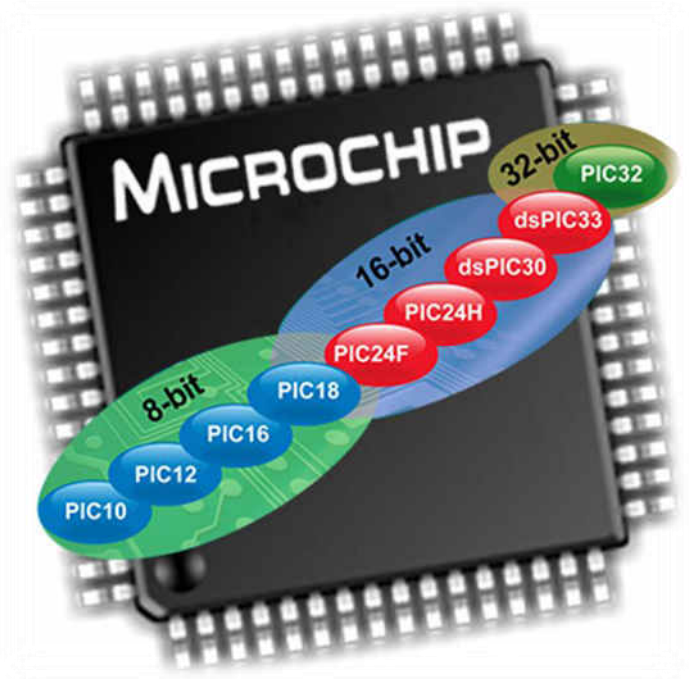


Figure 2.4 : *Familles des microcontrôleurs PIC 8, 16 et 32 bits.*

Tableau 2.2 : *Caractéristiques intrinsèques des microcontrôleurs PIC.*

Caractéristiques en communs	Caractéristiques additionnelles	Autres Caractéristiques
✓ Digital I/O ports. ✓ On-chip timer avec 8-bit prescaler. ✓ Power-on reset ✓ Watchdog timer. ✓ Power-saving SLEEP mode. ✓ Courant débité. ✓ External clock interface. ✓ Mémoire de données (RAM) . ✓ Mémoire EPROM or Flash program memory.	✓ Entrée analogique (Analog input channels). ✓ Comparateur analogique (Analog comparators). ✓ TIMER additionnel (Additional timer circuits). ✓ EEPROM data memory. ✓ Interrupteur (External and internal interrupts). ✓ Oscillateur intérieur (Internal oscillator). ✓ Modulation-largeur-impultion (Pulse-width modulated (PWM) output). ✓ USART serial interface (Universal Synchronous Asynchronous Receiver Transmitter).	✓ CAN bus interface. ✓ Direct LCD interface. ✓ USB interface. ✓ Motor control

Info

Program Memory (Flash) :
- ✓ Contient le programme à exécuter.
- ✓ Son contenu demeure conservé après mise hors tension (Type non volatile : ROM).

Data Memory (RAM) :
- ✓ Utilisée pour stocker les variables du programme (données temporaires).
- ✓ Pour pouvoir travailler normalement le µC doit stocker des données temporaires quelque part et c'est la qu'intervient la mémoire RAM.
- ✓ Type volatile où les données seront perdues en cas mise hors tension.

La mémoire EEPROM :
- ✓ Elle sert à stocker quelques données utilisées dans le programme comme les temporisations, les comptages, etc. Ces données sont conservées après une coupure de courant (non volatile) et sont très utiles pour conserver des paramètres semi-permanents.

Les registres spéciaux : SFR :
- ✓ SFR sont utilisés par le microcontrôleur pour contrôler ses opérations internes.
- ✓ Le nombre des SFR varie selon la complexité du µC.
- ✓ Parmi les SFR les plus importants/utilisés :
  - OPTION register.
  - I/O registers.
  - Timer registers.
  - INTCON register.
  - A/D converter registers.
  - … etc.

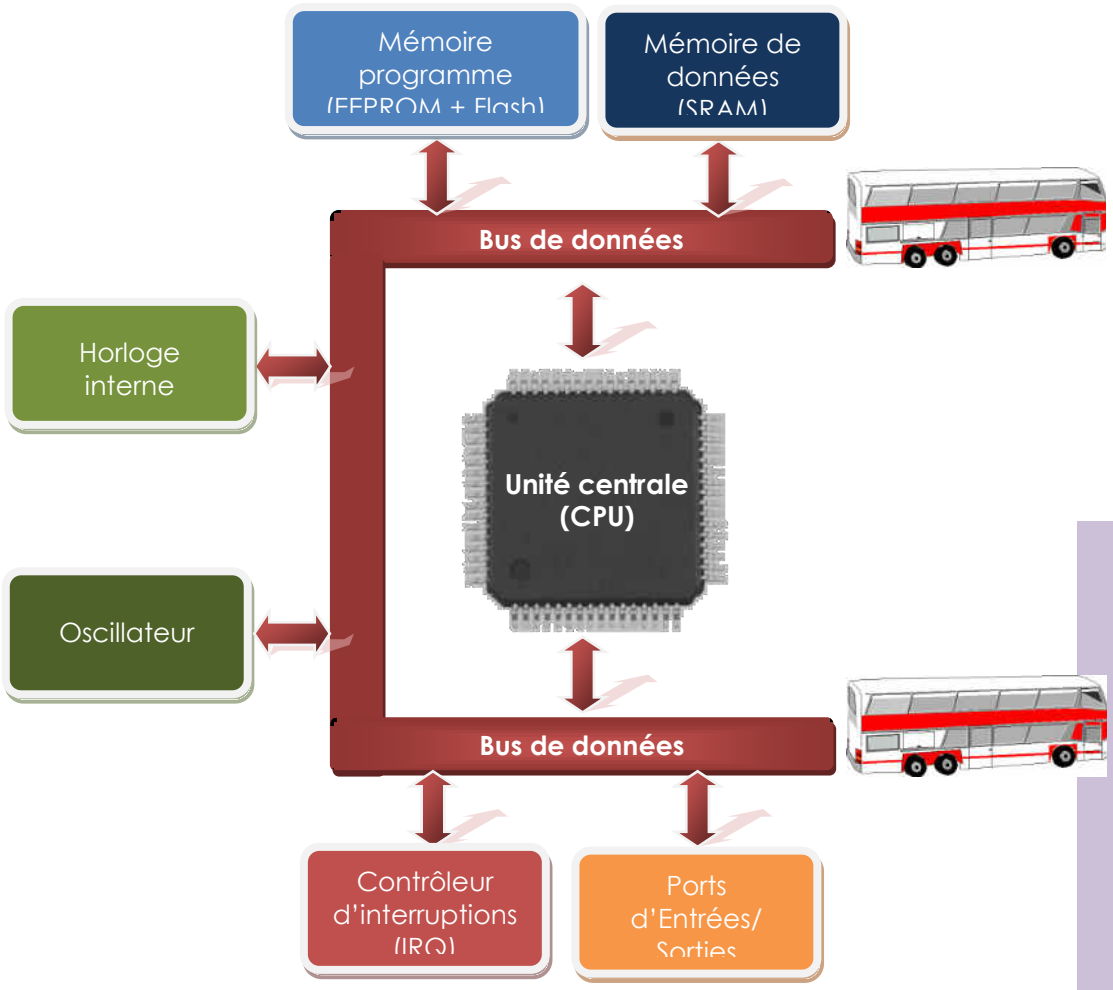
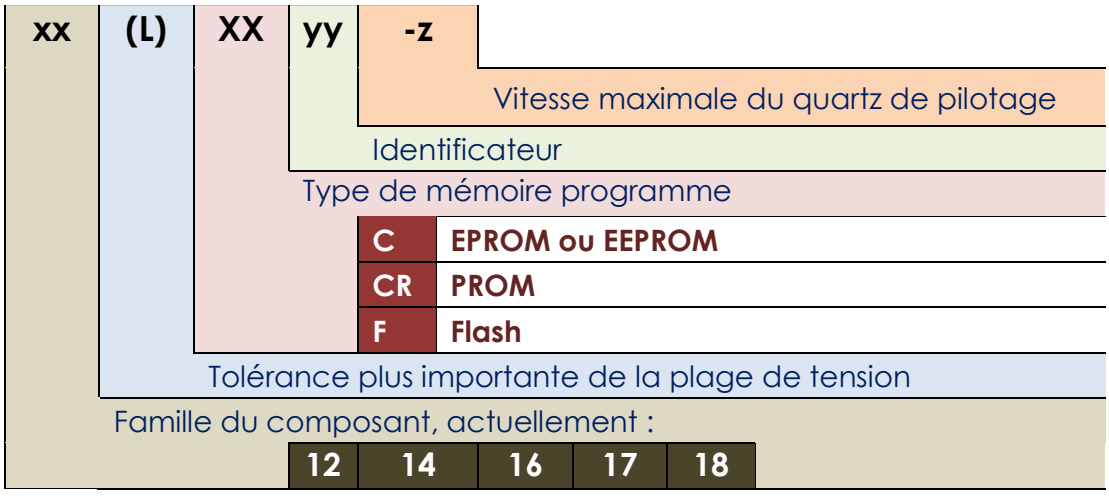


Figure 2.5 : *Caractéristiques principales des microcontrôleurs PIC.*

2.1.2. Identification des PICs de Microchip

Un PIC est généralement identifié par une référence de la forme suivante :



Exemple

16	F	84	-10

-10 : Quartz à 10MHz au maximum

84 : Identificateur No. 84

F : Mémoire programme du type **Flash**

16 : Famille Mid-Line :

Le tableau ci-dessous illustre certaines caractéristiques de quelques PICs.

Tableau 2.3 : *Caractéristiques de quelques PICs.*

Type	Taille de mémoire programme	Taille de mémoire de données	EEPROM en octets	Fréquence Max	Lignes d'E/S	Nombre de broches
12C508	**512x12**	**25**	**-**	**4**	**6**	**8 broches**
16C72A	**2048x14**	**128**	**-**	**20**	**22**	**28 broches**
16F84	**1024x14**	**68**	**64**	**20**	**13**	**18 broches**
16F628	**2028x14**	**224**	**128**	**20**	**16**	**18 broches**
16F876	**8192x14**	**368**	**256**	**20**	**22**	**28 broches**
16F877	**8192x14**	**368**	**256**	**20**	**33**	**40 broches**

2.2. Microcontrôleur PIC 16F84A

Le PIC 16F84 est un microcontrôleur 8 bits (dispose d'un bus de données de 8 bits) d'architecture de type RISC. Ses principales caractéristiques et constituants sont données par le tablrau 2.4.

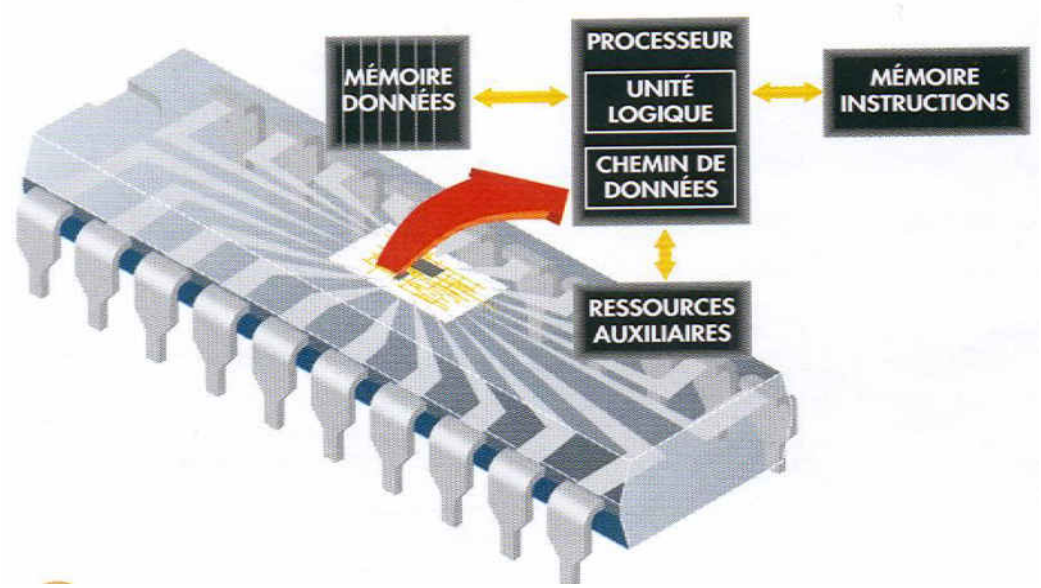


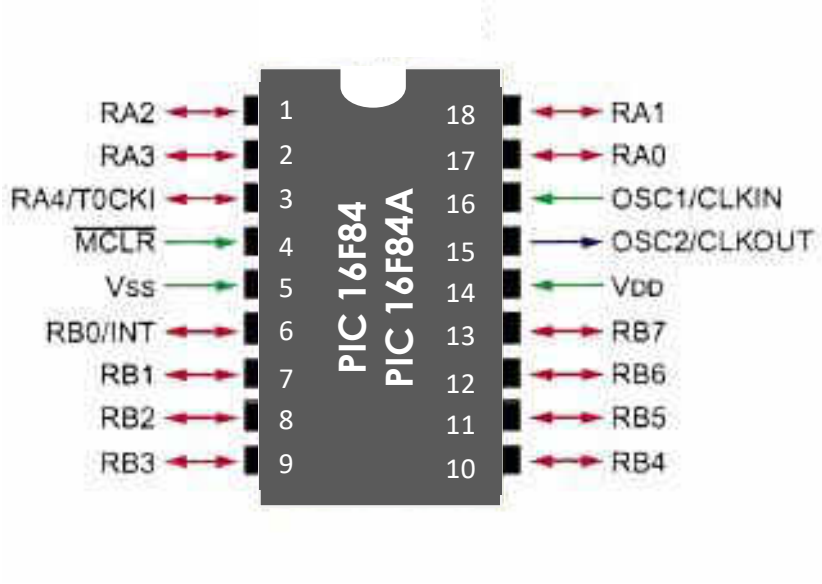
Figure 2.6 : *Principales constituants du PIC 16F84.*

Tableau 2.4 : *Principales caractéristiques et constituants du PIC 16F84.*

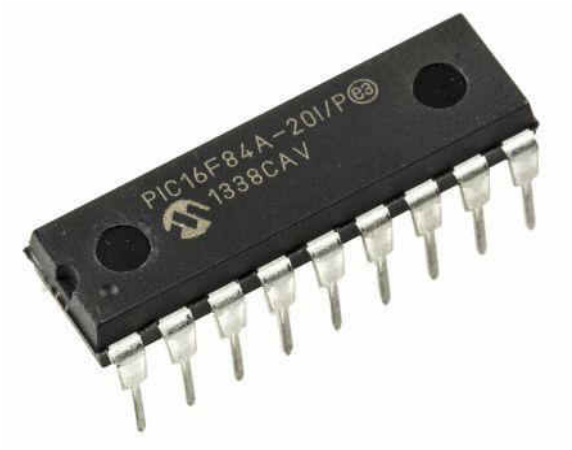
Principaux caractéristiques	✓ 35 Instructions (RISC) codées sur 14 bits (famille Mide-Line). ✓ Données sur 8 bits. ✓ Vitesse maximum 10 MHz soit une instruction en 400 ns (1 cycle machine = 4 cycles d'horloge). ✓ Deux ports d'entrées / sorties. ✓ 4 sources d'interruption. ✓ 1000 cycles d'effacement/écriture pour la mémoire flash. ✓ 10.000.000 pour la mémoire de donnée EEPROM.
Principaux constituants	✓ Mémoire de type FlashROM pour le programme; ✓ Mémoire de type RAM pour les registres; ✓ Mémoire de type EEPROM pour les données à sauvegarder; ✓ Registres particuliers: W, FSR et d'état; ✓ Unité Arithmétique et Logique (UAL) ; ✓ Ports d'entrées / sorties; ✓ Pile à 8 niveaux.

2.2.1. Brochage du PIC 16F84A

Ce microcontrôleur est commercialisé dans un boîtier DIP (Plastic Dual In-line Package) 18 broches comme indique la figure suivante :



(a)



(b)

Figure 2.7 : *Brochage du PIC 16F84.*

Tableau 2.5 : *Fonctions des broches du microcontrôleur PIC 16F84.*

Broches	Description	
VDD	Alimentation (2,0 V à 5,5 V DC)	
VSS	Masse (0 V)	
MCLR	Reset (Master Clear : Actif au niveau bas)	
OSC1/CLKIN, OSC2/CLKOUT	Circuit d'horloge	
RA0, RA1, RA2, RA3, RA4/T0CKI	Port A (5 bits) : Port d'Entrées / Sorties	
RA4/T0CKI	Broche multifonction	
	RA4	Ligne d'Entrée / Sortie du port A
	T0CKI	Entrée d'horloge externe du timer
RB0/INT, RB1, RB2, RB3, RB4, RB5, RB6, RB7	Port B (8 bits): Port d'Entrées / Sorties	
RB0/INT	Broche multifonction	
	RB0	Ligne d'Entrée / Sortie du port B
	INT	Entrée d'interruption externe

- Chaque broche des ports A et B ne peut débiter plus de 20 mA ou absorber plus de 25 mA.
- Le total des intensités débitées par le port A ne peut dépasser 50 mA.
- Le total des intensités débitées par le port B ne peut dépasser 100 mA.

- Le total des intensités absorbées par le port A ne peut dépasser 80 mA.
- Le total des intensités absorbées par le port A ne peut dépasser 150 mA.
- Le PIC 16F84 ne fonctionne qu'avec deux niveaux logiques 0 et 1 ; Bas et Haut (Les broches comportent des trigger de Schmitt ou TLL), illustrés par le schéma des niveaux logiques de la figure 2.8.
-

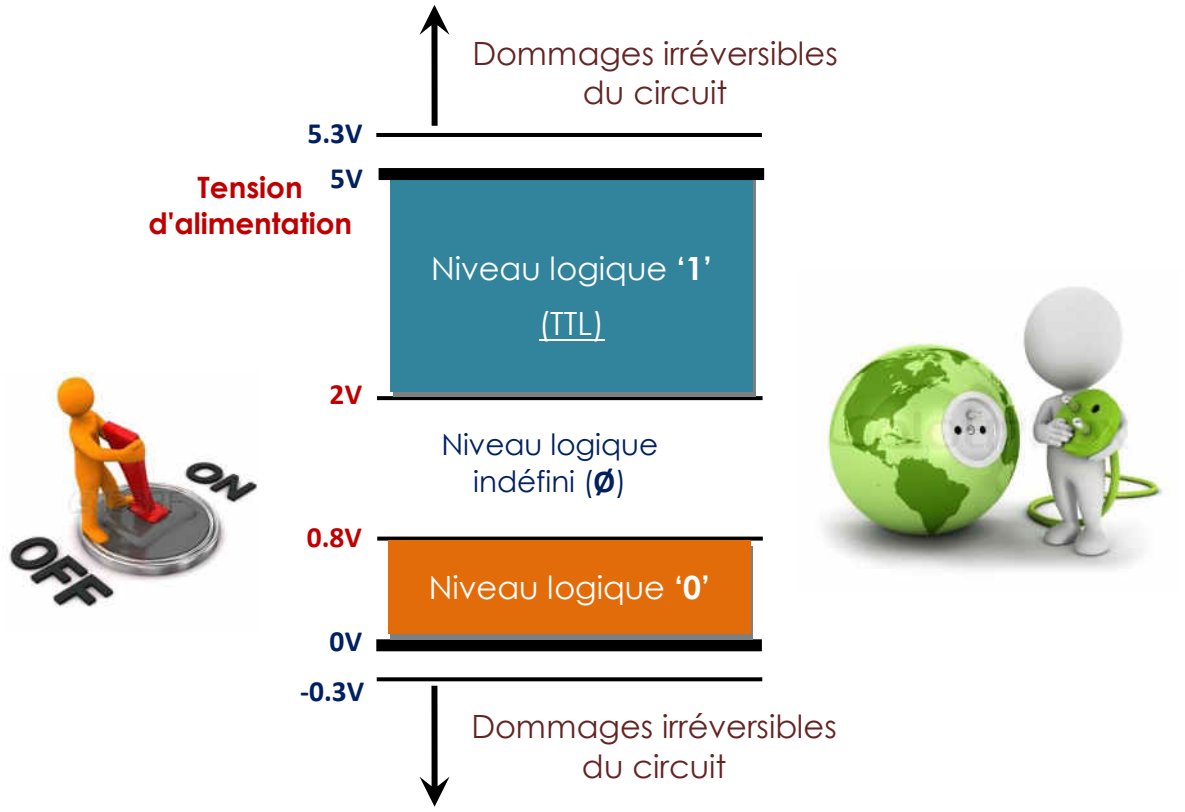


Figure 2.8 : *Niveaux logiques du PIC 16F84.*

2.2.2. Organisation interne du PIC 16F84A

La structure générale du PIC 16F84 comporte 6 blocs :

- Mémoire de programme.
- Mémoire de données.
- Processeur (unité de commande, UAL).
- Bloc de gestion d'alimentation.
- Bloc d'horloge.
- Ressources auxiliaires (Périphériques connectés aux ports d'entrées / sorties).

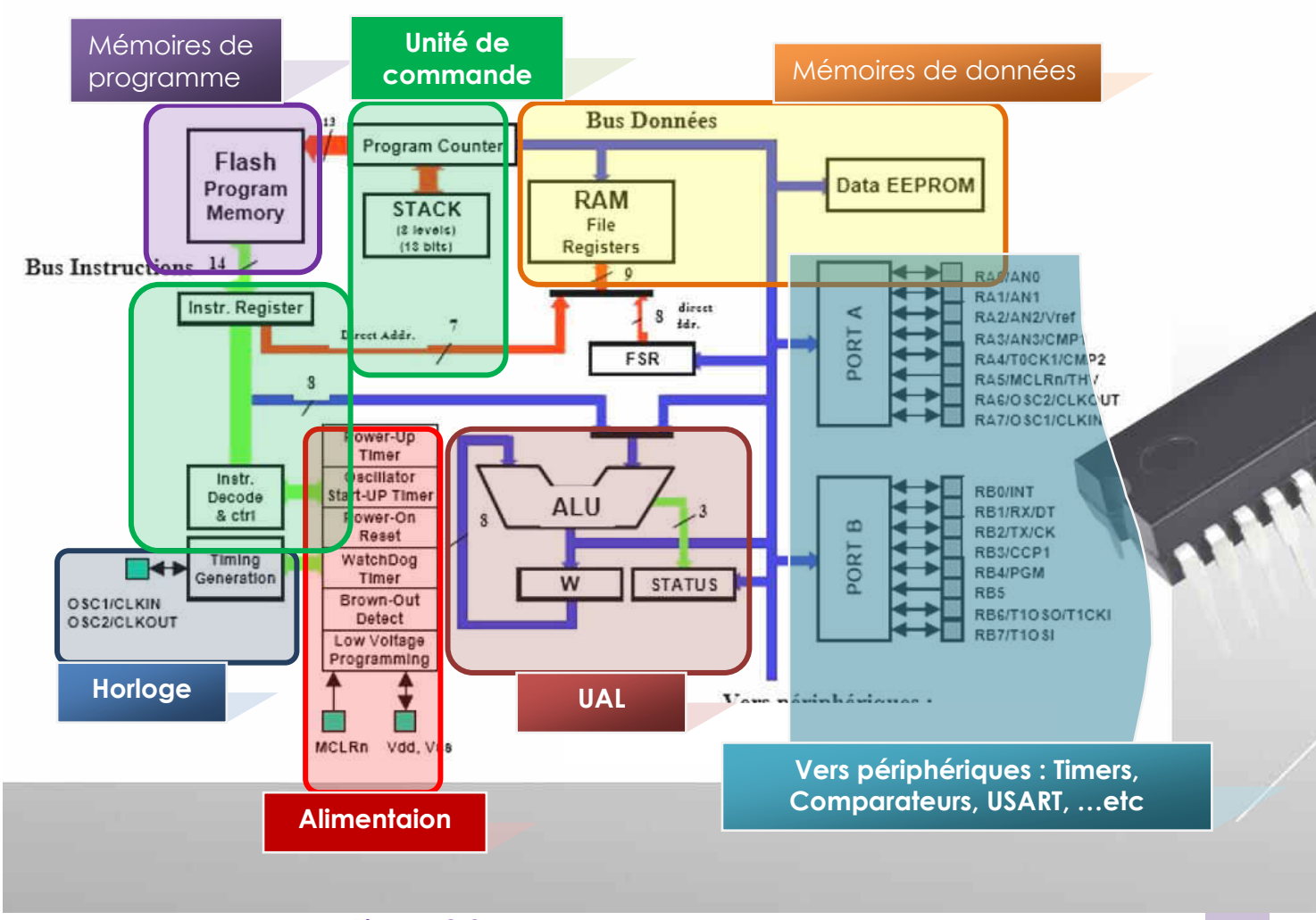


Figure 2.8 : *Structure générale du PIC 16F84.*

La structure interne détaillée du PIC16F84 constituée des éléments donnés dans le schéma bloc ci-dessous. Il s'agit de :

- ✓ Une unité arithmétique et logique (ALU).
- ✓ Une mémoire flash de programme de 1k "mots" de 14 bits.
- ✓ Une mémoire RAM constituée de:
  - ✓ Des registres de control **SFR** (Special Function Registers).
  - ✓ 68 octets de RAM utilisateur appelés aussi **GPR** (General Purpose Resisters).
- ✓ Une mémoire EEPROM de données de 64 octets.
- ✓ Un compteur de programme PC (Program Counter).
- ✓ Une pile (Stack) de 8 niveaux (octets).
- ✓ Un registre contenant le code de l'instruction à exécuter.
- ✓ Un bus spécifique pour le programme (program bus).
- ✓ Un bus spécifique pour les données (data bus).
- ✓ Deux ports d'entrée sortie, un de 8 bits (Port A) et un de 5 bits (Port B).
- ✓ Un timer/Compteur cadencé par une horloge interne ou externe.

- ✓ Un chien de garde / compteur qui est un timer particulier.
- ✓ Un système d'initialisation à la mise sous tension (Power-up timer, etc.).
- ✓ Un système de génération d'horloge à partir du quartz externe (timing génération).

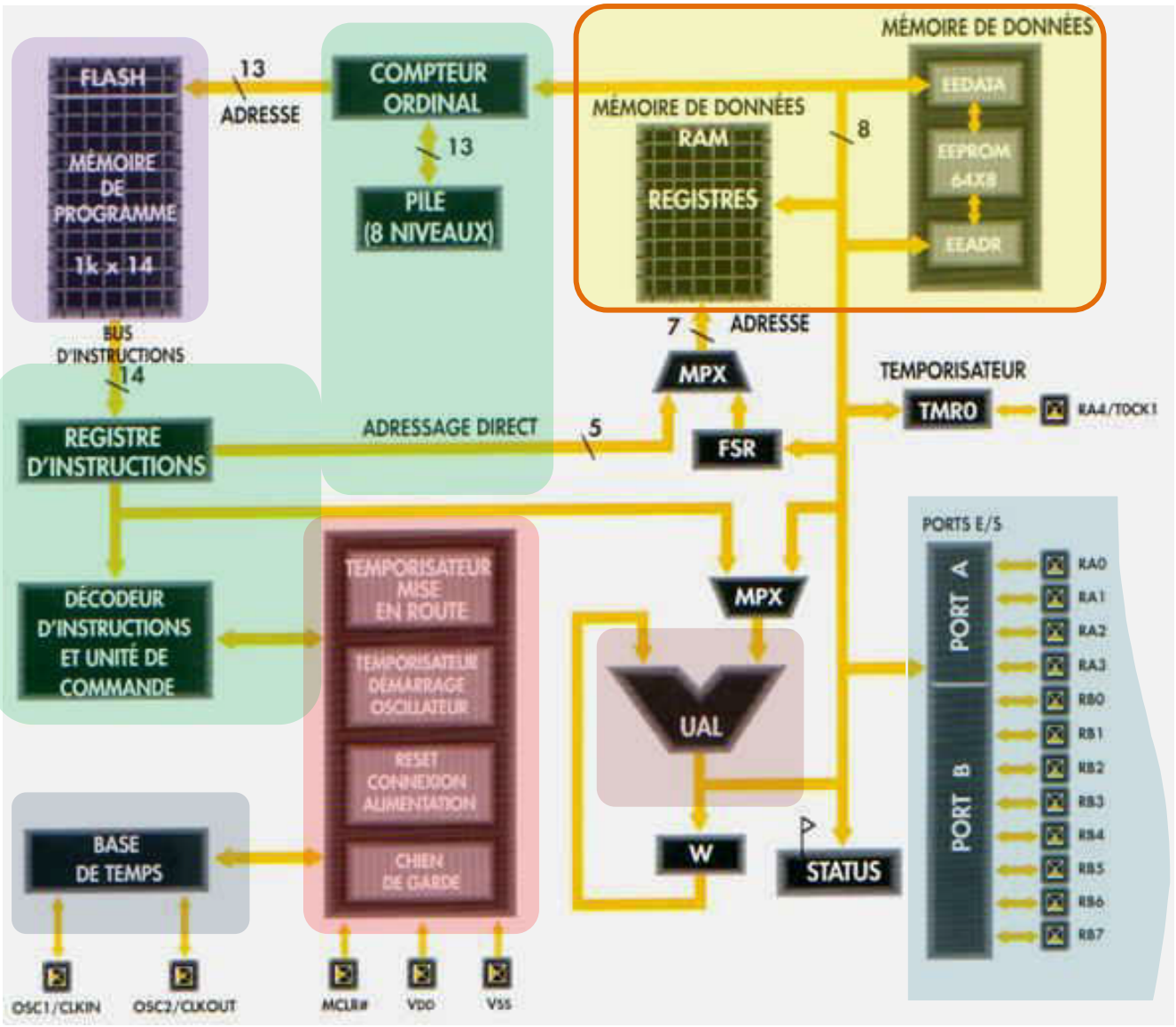
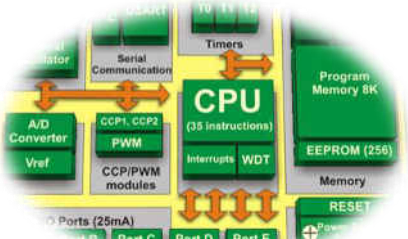


Figure 2.9 : *Architecture interne du PIC 16F84.*

Info

- Le Compteur Programme (PC) contient l'adresse de l'instruction à exécuter.
- Seulement 35 instructions codées sur 14 bits.
- Toutes les instructions ont un seul 1 cycle machine par instruction, sauf pour les sauts (2 cycles machine).
- Vitesse maximum 20 MHz soit une instruction en 400 ns (1 cycle machine = 4 cycles d'horloge).
- Mémoire programme de 1024 mots.



- Mémoire RAM de données de 68 octets.
- Mémoire EEPROM de 64 octets.
- Données de 8 bits.
- 15 Registres pour des fonctions spéciales.
- Pile de 8 niveaux de profondeur (La pile désigne les emplacements mémoire où sont empilées les adresses de retour lors des appels à des sous-programmes).
- La pile ne comportant que 8 niveaux (octets), il ne faut pas imbriquer plus de 8 sous-programmes.
- Adressage direct, indirect and relatif.
- 4 sources d'interruption :
  - ✓ Broche externe RB0/INT ;
  - ✓ Débordement du timerTMR0 ;
  - ✓ Interruption sur transition sur les broches PORTB<7:4> ;
  - ✓ Ecriture complète des données sur la mémoire EEPROM.
- 13 broches d'entrées / Sorties avec contrôle individuel de la direction ;
- Source de courant important pour la commande d'une LED :
  - ✓ 25 mA max. dissipé par broche ;
  - ✓ 25 mA source de courant max. par broche ;
  - ✓ TMR0: Timer8-bit programmable.
- Mémoire programme FLASH à 10 000 cycles Effacement / Ecriture;
- Mémoire données EEPROM à 10 000 000 cycles Effacement / Ecriture, avec un maintien de données > 40 ans.
- Programmable par liaison série sur 2 broches (ICSP™).
- Reset à la mise sous tension [Power-on Reset (POR)], Power-up Timer(PWRT).
- Un Timer chien de garde (WDT) avec son propre oscillateur RC.
- Protection de code.
- Mode SLEEP pour la sauvegarde d'énergie.
- Oscillateur sélectionné au choix.
- Large plage de tension de fonctionnement :
  - ✓ Commercial: 2.0V to 5.5V.
  - ✓ Industriel : 2.0V to 5.5V.

2.3. Fonctionnement du PIC 16F84

Pour mieux comprendre le bon fonctionnement du PIC 16F84, les blocs fondamentals constituants l'architecture interne seront étudiés en détails, à savoir :

- ✓ L'Horloge.
- ✓ Le RESET.
- ✓ Le processeur ou l'Unité Centrale.
- ✓ Les mémoires (Programme / Données).

- Les ressources auxiliaires :
  - Ports d'entrées et de sorties.
  - Temporisateur (TMR).
  - Chien de garde (WATCHDOG).
  - Mode sommeil (consommation réduite).
  - Interruptions.

Mise en marche

L'utilisation et la mise en œuvre très simple des PICs les a rendus extrêmement populaire. Pour faire fonctionner le PIC, il suffit :

- D'alimenter le circuit par ses deux broches **VDD** et **VSS**.
- De fixer sa vitesse de fonctionnement à l'aide d'un **QUARTZ**.
- D'élaborer un petit système pour permettre de réinitialiser le microcontrôleur sans avoir à couper l'alimentation (**RESET** (Master Clear)).
- D'écrire le programme en langage assembleur ou en C sur un ordinateur grâce au logiciel **MPLAB** de MICROCHIP (**logiciel gratuit**) puis de le compiler pour le transformer en langage machine et le transférer dans le PIC grâce à un **programmateur**.
- De mettre en service l'application développée.

2.3.1. L'Horloge

Les broches OSC1/CLOCKIN et OSC2/CLOCKOUT sont réservées à l'horloge. L'horloge du PIC16F84A peut être soit interne soit externe.

- L'Horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC.
- Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 4, 10 ou 20 MHz selon le type de microcontrôleur.
- Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser le PIC sur un processus particulier (Master – Slave).
- Selon la configuration des bits FOSC1 et FOSC0 du registre de **CONFIGURATION**, quatre modes de fonctionnement peuvent être définies :

Tableau 2.5 : *Types d'horloges utilisés du PIC 16F84.*

FOSC1	FOSC0	Type de l'Oscillateur
0	0	RC Resistance/Capacité : Réseau de R et C (Fmax = 4MHz).
0	1	XT Crystal/Resonator : Oscillateur à **QUARTZ** ou externe (Fmax = 4MHz).
1	0	HS High Speed Crystal/Resonator : Oscillateur à **QUARTZ** grand vitesse (Fmax= 10MHz).

1	1	LP Low Power Crystal : Oscillateur à **QUARTZ** basse consommation (Fmax= 200KHz).

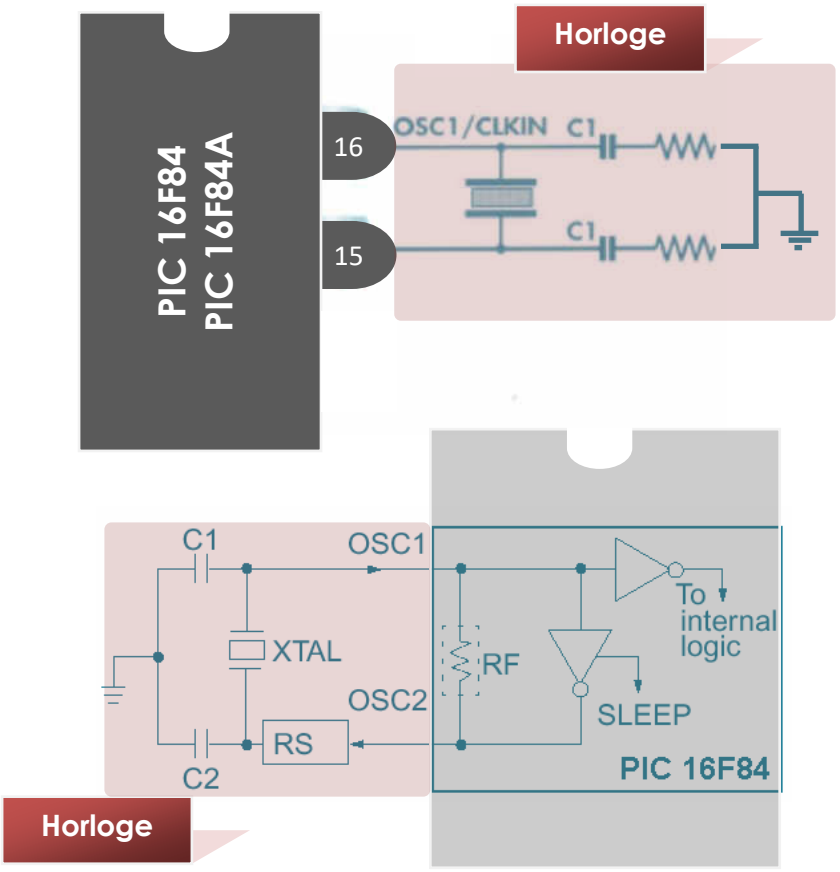


Figure 2.10 : *Horloge interne LP, XT ou HS.*

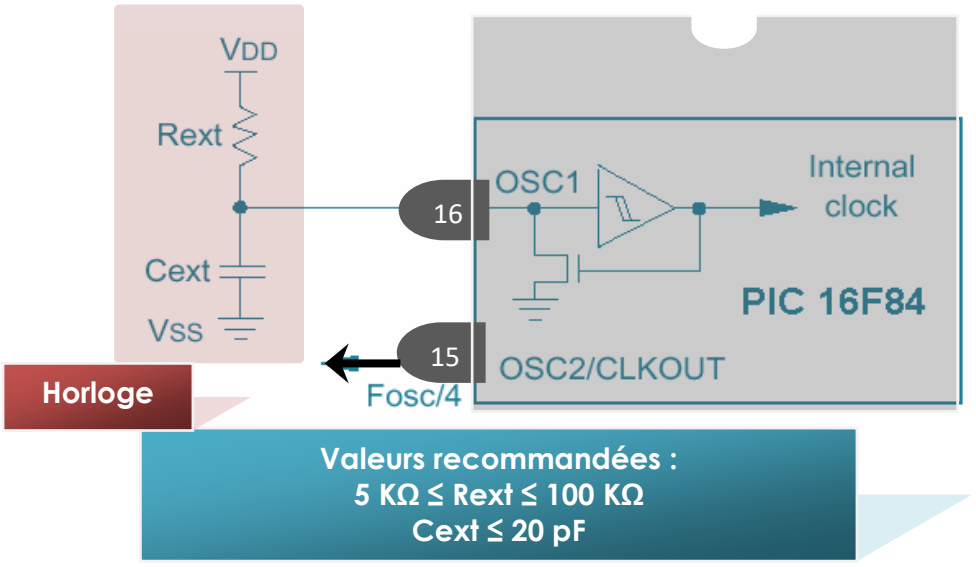


Figure 2.11 : *Horloge interne à oscillateur RC.*

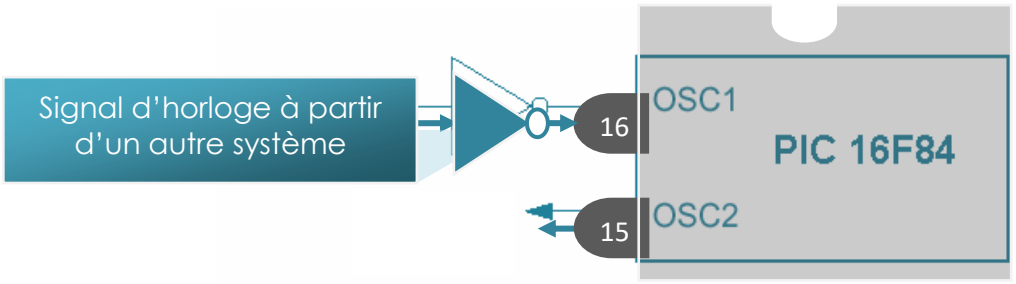


Figure 2.12 : *Horloge externe LP, XT ou HS.*

Le tableau ci-dessous illustre les valeurs typiques des composants électroniques adaptés pour différents types et fréquences des oscillateurs pour le cas de PIC 16F84.

Tableau 2.6 : *Sélection des capacités adéquates pour un Quartz du PIC 16F84.*

Mode	Fréquences	OSC/C1	OSC/C2
LP	32 KHz	68 – 100 pF	68 – 100 pF
	200 KHz	15 – 33 pF	15 – 33 pF
XT	100 kHz	100 – 150 pF	100 – 150 pF
	2 MHz	15 – 33 pF	15 – 33 pF
	4 MHz	15 – 33 pF	15 – 33 pF
XS	2 MHz	15 – 33 pF	15 – 33 pF
	20 MHz	15 – 33 pF	15 – 33 pF

- L'oscillateur à quartz présente une meilleure précision que l'oscillateur RC.
- Avec un oscillateur RC, la fréquence de l'oscillation est fixée par Vdd, Rext et Cext. Elle peut varier légèrement d'un circuit à l'autre.

- ✓ En mode XT, LP et HS un quartz ou un résonateur (filtre céramique) est branché entre les connexions OSC1 et OSC2. On peut avoir des fréquences allant jusqu'à 4, 10 ou 20 MHz selon le type de µC.
- ✓ Le filtre passe bas RS, C2 limite les harmoniques dus à l'écrêtage et réduit l'amplitude de l'oscillation.

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

2.3.2. RESET

Le RESET ou l'opération de réinitialisation du microcontrôleur se fait généralement via l'entrée $\overline{\text{MCLR}}$ (Broche 4). Cette broche est utilisée aussi comme entrée de la tension de programmation.

Le RESET (Etat '0' sur la broche $\overline{MCLR}$.) s'effectue en mettant un '0' logique sur la broche **4** du microcontrôleur. Il existe plusieurs matérialisations de système de RESET: RESET manuel, RESET automatique et RESET mixte.

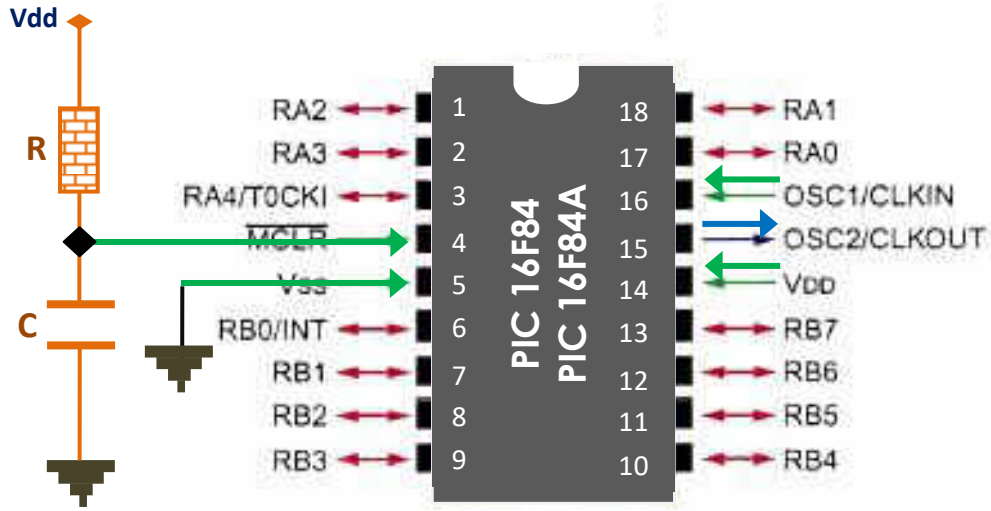


Figure 2.13 : *RESET automatique (C = 1µF, R = 1 KΩ).*

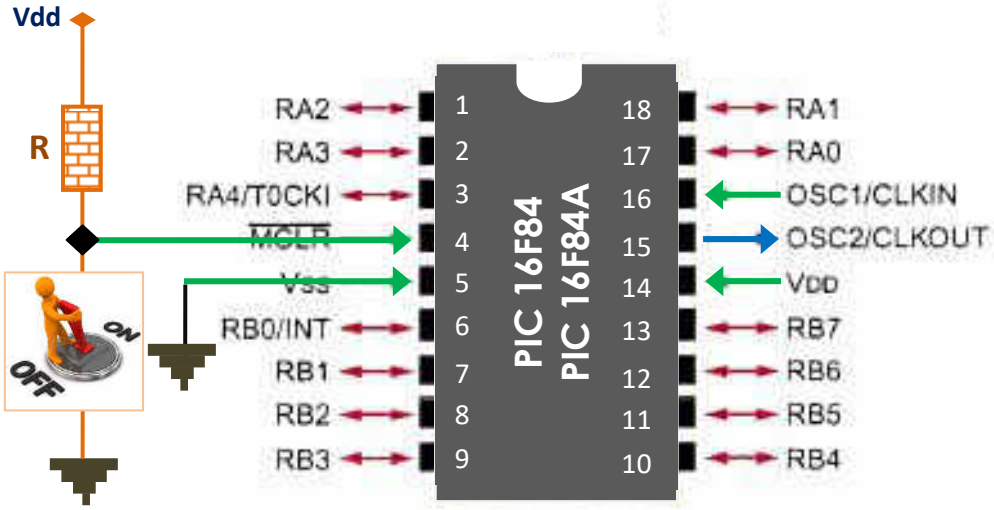


Figure 2.13 : *RESET Manuel (R = 1 KΩ).*

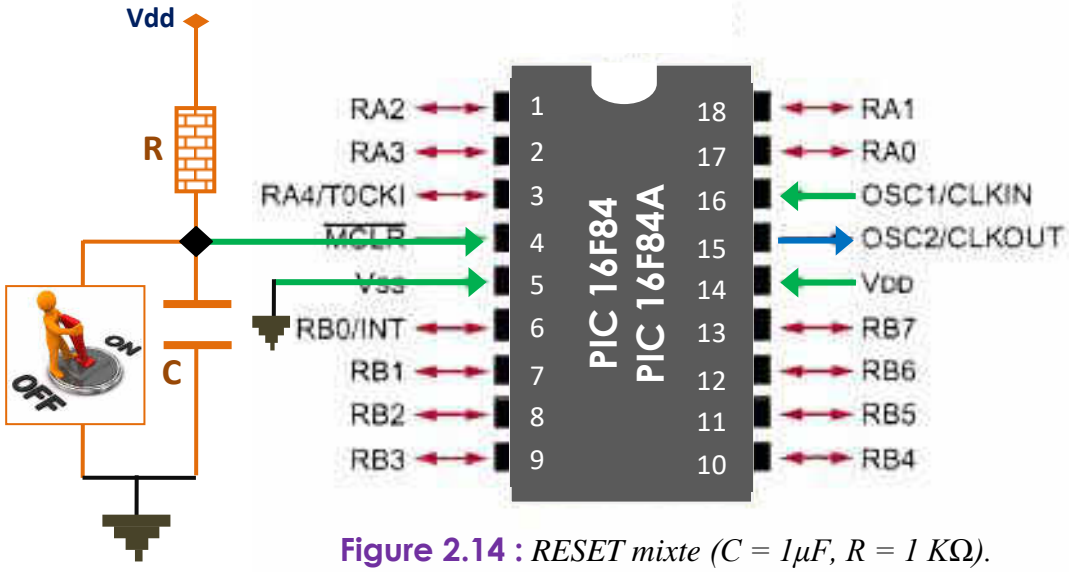


Figure 2.14 : *RESET mixte (C = 1µF, R = 1 KΩ).*

- Le RESET peut avoir plusieurs causes :
  - ✓ Mise sous tension
  - ✓ Etat 0 sur broche $\overline{\text{MCLR}}$.
  - ✓ Débordement du timer du chien de garde.
- Lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur ($\overline{\text{MCLR}} = 0$), le microcontrôleur s'arrête, place tout ses registres dans un état connu et se redirige vers le début de la mémoire de programme pour recommencer le programme au début (adresse dans la mémoire de programme : 0000 Hex).
- Lorsque le RESET intervient, le µC peut être :
  - ✓ En fonctionnement normal
  - ✓ En mode SLEEP (mise en veille).
- Le modèle de RESET mixte, permettant un RESET automatique lors de la mise sous tension et également un interrupteur de RESET manuel.

- Afin de maintenir les broches d'entrées dans un niveau logique donné 'o' ou '1' (niveau par défaut), Il est nécessaire d'ajouter la résistance Pull-up/ Pull-down (Rpu/ Rpd) ayant une grande valeur d'ordre 10K comme indique le schéma de la figure ci-dessous.

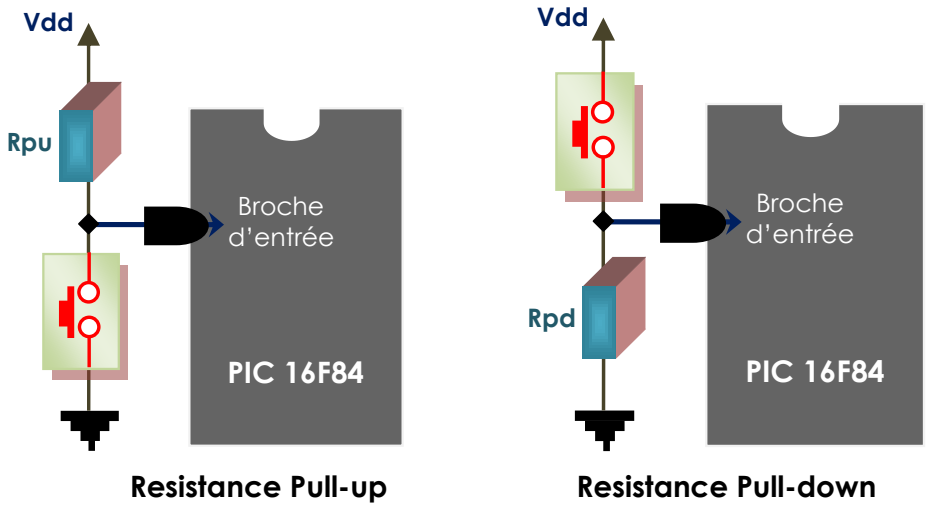


Figure 2.14 : *Système de maintient des niveaux logiques des broches d'entrées des Ports du PIC 16F84.*

- L'entrée du PIC 16F84 est maintenue au niveau logique '1' pour le cas du Resistance Pull-up et '0' pour le cas de Resistance Pull- down, lorsqu'on

ferme l'interrupteur (L'entrée est forcée à la masse (resp. Vdd) qui correspond à '0' (resp. '1').

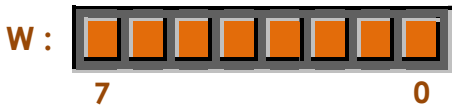
2.3.3. Processeur

L'unité centrale (processeur ou CPU) se charge de l'exécution des programmes qu'on lui a demandés et de la coordination entre les différents organes du système. Elle est composée de deux unités fondamentales :

- ✓ Unité de traitement ou Unité Arithmétique et Logique (**UAL**).
- ✓ Unité de commande.
- ✓ Des registres à usage général.

L'UAL est chargé d'exécuter les opérations arithmétiques (+, -, /, etc.) et logiques (NOT, AND, OR, etc.). Elle contient

- ✓ Tous les circuits électroniques qui réalisent effectivement les opérations désirées.
- ✓ Des registres et des indicateurs sont associés à l'UAL :
  - Le registre de travail (**W : Working register**) sert à contenir les opérandes et les résultats intermédiaires.



  - Les indicateurs (**Registre d'état : STATUS**) pour indiquer l'état du résultat : Résultat nul, >0, <0, débordement, …etc.

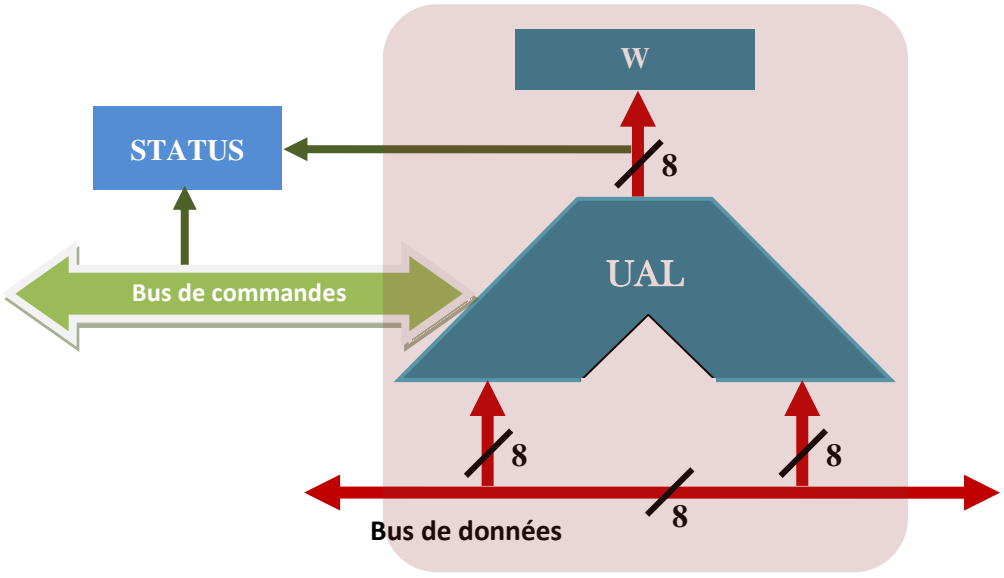


Figure 2.15 : *UAL du PIC 16F84.*

Info

- Le registre de travail (W) n'est pas adressable comme les autres registres.
- Dans le cas des opérations à deux opérandes, un opérande est dans le registre de travail W, l'autre est une constante, ou est contenu dans un autre registre ou une case mémoire (selon le mode d'adressage utilisé).

- Dans le cas des opérations à un seul opérande, l'opérande est soit dans W, soit dans un autre registre ou une case mémoire (selon le mode d'adressage utilisé).
- Il est à noter que Les valeurs des flags du registre d'état C, DC et Z sont affectées selon les les résultats des opérations effectuées par l'UAL.

L'unité de commande extrait de la mémoire centrale la nouvelle instruction à exécuter. Elle analyse cette instruction et établi les connexions électriques correspondantes fans l'unité de traitement. Autrement, son rôle est de coordonner et de synchroniser toutes les actions (micro opérations) durant l'exécution des tâches. Elle est composée au minimum de :

- D'un registre instruction (**RI (14 bits)**),
- D'un compteur de programme (PC),
- D'un séquenceur ou générateur de séquences (**GS :** décodeur de fonctions).

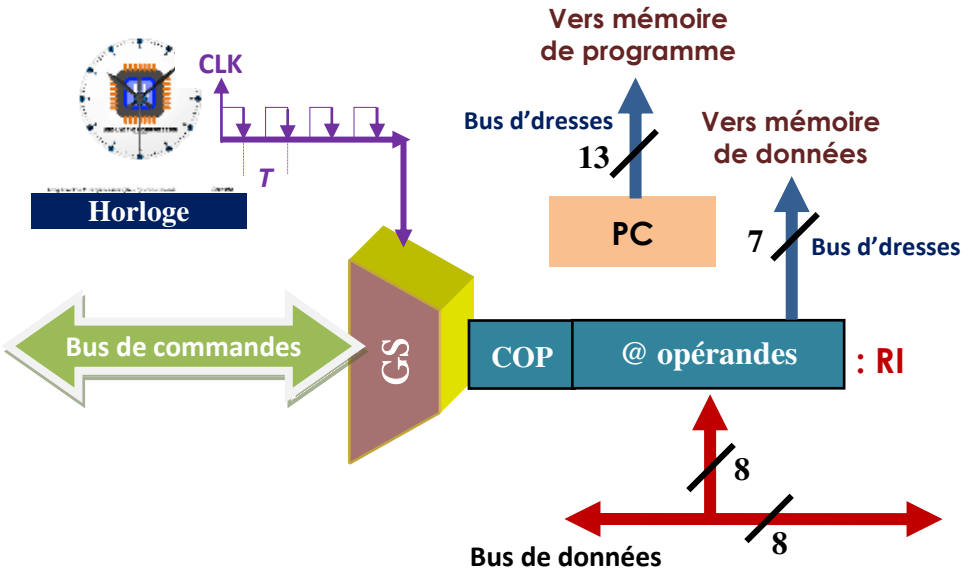


Figure 2.16 : *Unité de commande du PIC 16F84.*

2.3.4. Organisation de la mémoire

Le PIC 16F84 est conçu selon l'architecture de Harvard. Donc, il possède 2 mémoires:

- ✓ Mémoire de programme.
- ✓ Mémoire des données.

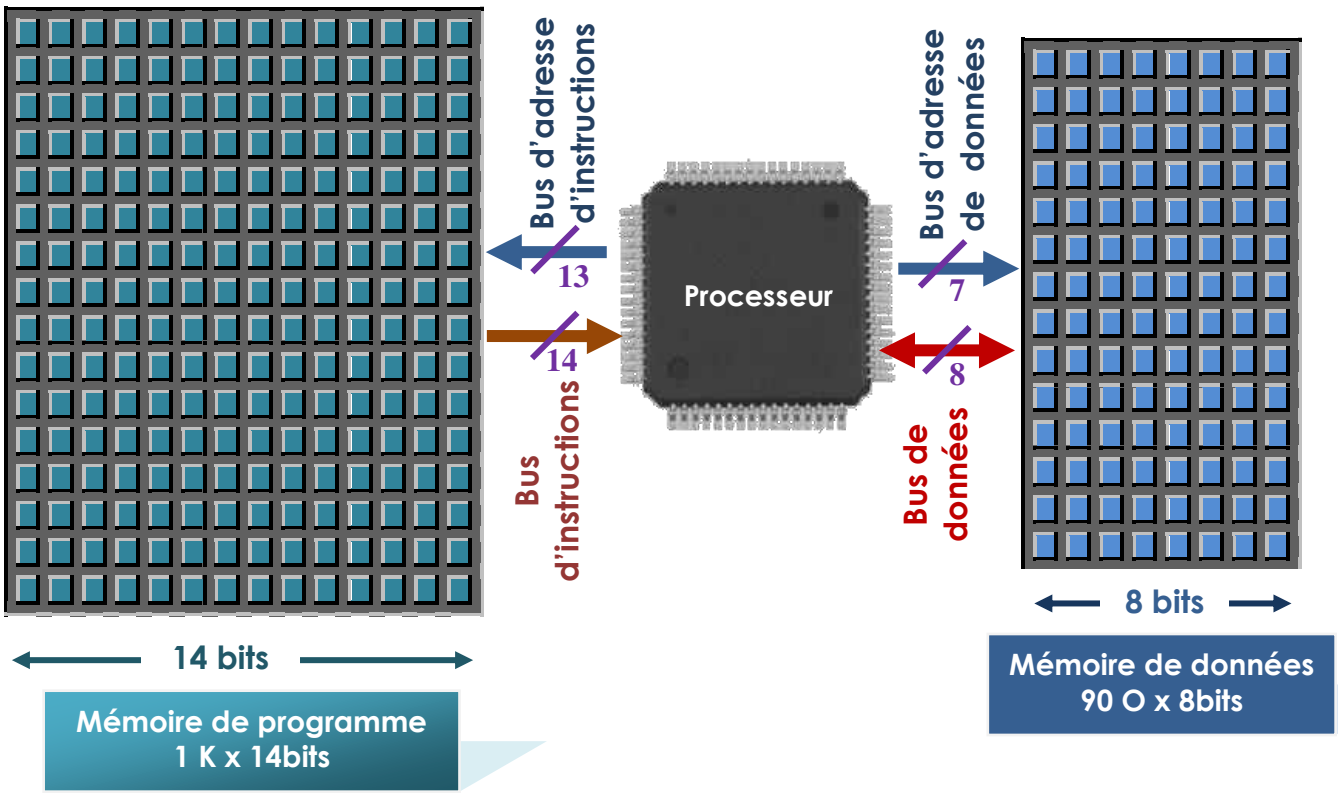


Figure 2.17 : *Blocs mémoires du PIC 16F84.*

2.3.4.1. Mémoire de programme

La mémoire programme est de type **Flash** (non volatile). Elle contient le programme à exécuter (programme de gestion du matériel).

- Le PIC 16F84 possède un compteur ordinal (PC) qui permet d'adresser 2^{13} **= 8 K x 14bits**.
  - ✓ Dans le cas du PIC16F84, seulement 1 K x 14 bits (1024 emplacements) est implémenté physiquement (0000h-03FFh).
    - Les instructions sont codées sur 14 bits.
    - L'adresse 0000h contient le vecteur du RESET.
    - L'adresse 0004h l'unique vecteur d'interruption du PIC.

Info

- La pile et le compteur de programme, n'ont pas d'adresse dans la plage de mémoire, ce sont des zones réservées par le système.
- La pile est une zone mémoire particulière. Pour le PIC 16F84, elle est constituée de 8 emplacements (**8 niveaux**) de 13 bits du type LIFO.
- La pile intervient dans le mécanisme interne des instructions :
  - ✓ **CALL.**
  - ✓ **RETURN.**
  - ✓ **RETLW (Saut dans un tableau).**
  - ✓ **RETFIE (Fin de la routine d'interruption).**

La technologie utilisée pour les mémoires de programme (Flash) permet plus de 1000 cycles d'effacement et de programmation.

- ✓ Mise sous tension
- ✓ Etat 0 sur broche MCLR.
- ✓ Débordement du timer du chien de garde.

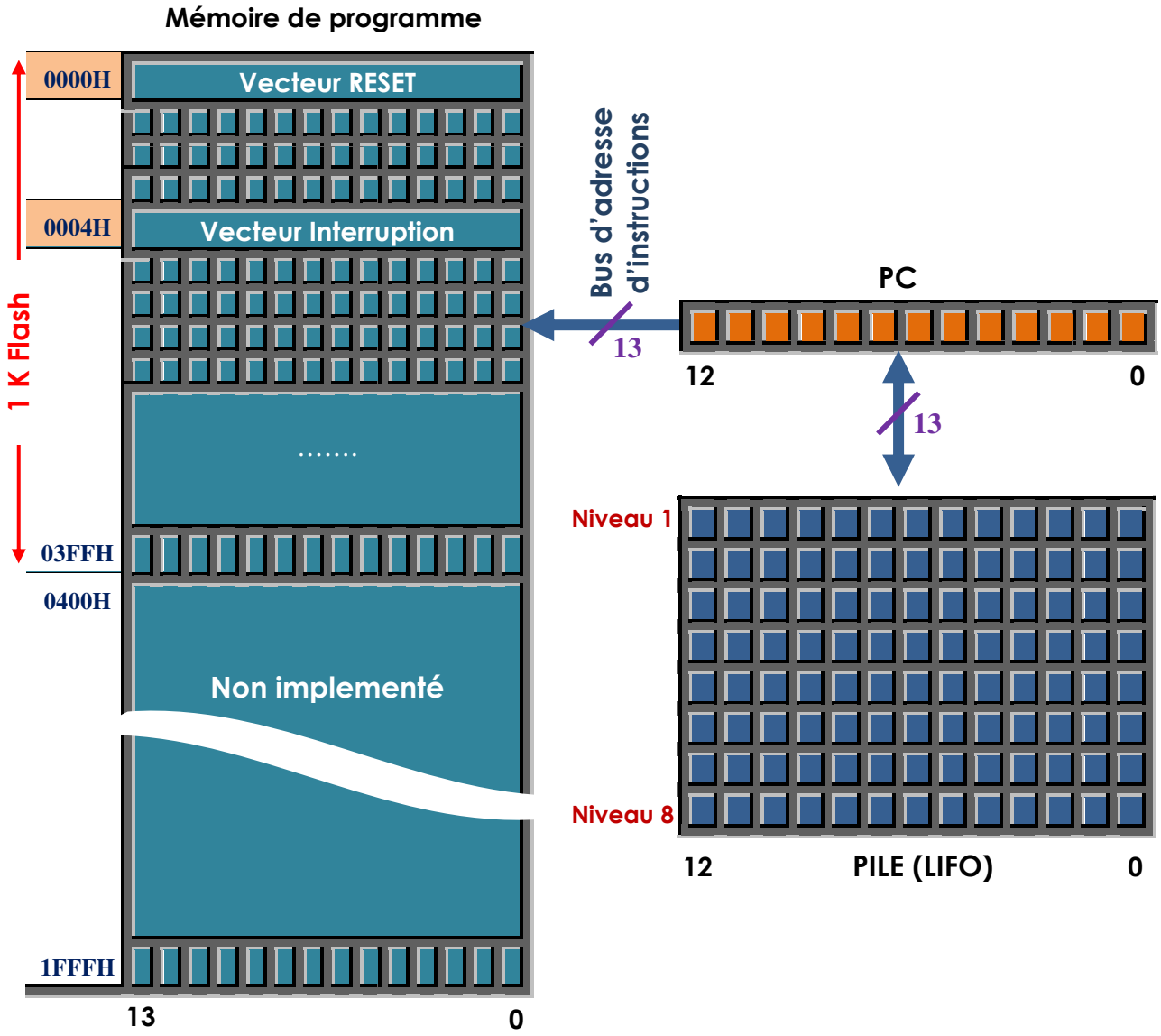


Figure 2.18 : *Mémoire de programme du PIC 16F84.*

2.3.4.2. Mémoire de données

La mémoire de données est partitionnée en deux espaces:

1. Mémoire RAM de **68 octets**.
   - ✓ Mémoire volatile.
   - ✓ Mot mémoire: 8 bits RAM = 1 Octet.
   - ✓ Elle contient deux parties :
     - Les **SFR** (Special Function Registers) : Ces registres permettent de contrôler les opérations sur le circuit.
     - Les registres généraux (**GPR**: General Purpose Registers) : Ces registres sont libres pour l'utilisateur (usage général).
2. Mémoire EEPROM de **64 octets**.
   - ✓ Accessible pour lecture et écriture.
   - ✓ Non volatile.
   - ✓ Plus lente que la RAM.
   - ✓ Accessible à l'aide des registres **EEADR** (registre d'adresse) et **EEDATA (registre de données)** sous le contrôle de deux registres de contrôle **EECON1** et **EECON2**.

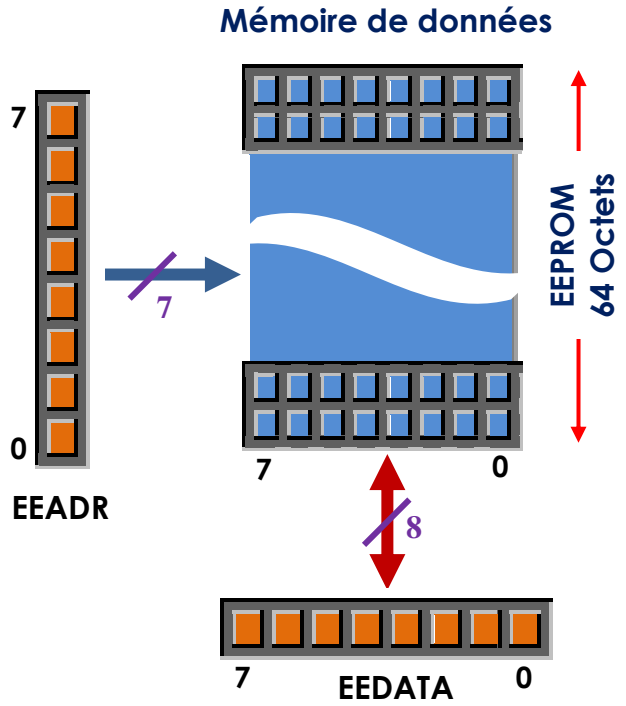


Figure 2.19 : *Mémoire de données EEPROM du PIC 16F84.*

Info

- La mémoire EEPROM de données est constituée de 64 octets accessibles en **lecture** comme en **écriture** depuis un programme.
- Ces octets sont **conservés** après une coupure de courant et sont très utiles pour conserver des paramètres temporaires (semi permanents).
- Les registres **EEADR** (Registre d'adresse) et **EEDATA** (Registre de données) sont associés à la mémoire EEPROM de données.

Les registres de contrôle (**EECON1** et **EECON2**) sont associés à la mémoire EEPROM permettant la configuration du mode de fonctionnement de cette mémoire.

a. Mémoire de données RAM

La mémoire données RAM est divisée en deux bancs (banques). Cette division est assurée par deux bits de contrôle qui se trouvent dans le registre STATUS

- ✓ Le banc 0 est sélectionné en mettant le bit RP0 du registre STATUS à 0.
- ✓ Le banc 1 est sélectionné en mettant le bit RP0 du registre STATUS à 1.

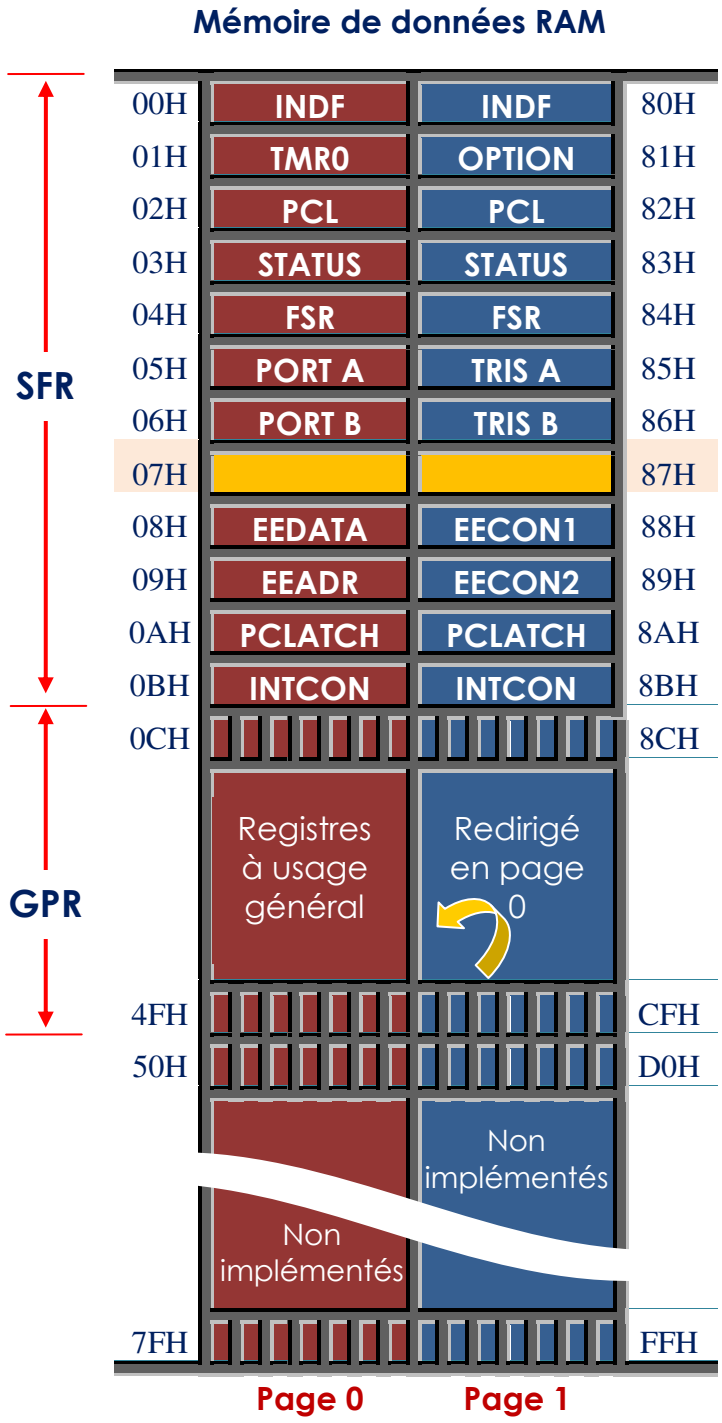


Figure 2.20 : *Mémoire de données RAM du PIC 16F84.*

- Chaque banc est composé de 128 octets.
- Les 12 premières lignes de chaque banc sont réservées pour les **SFR** (*Special Function Registers*).
- La RAM **GPR** (RAM utilisateur d'usage général ou encore *General Purpose Registers*) de 68 octets. Cette zone commence à 0CH ou 8CH (selon le bit RP0 du registre STATUS) et se termine à 4FH ou CFH, soient 68 octets disponibles pour les variables du programme.
- Les GPR sont accessibles soit directement soit indirectement à travers les registres FSR et INDF, selon le mode d'adressage utilisé.
- Les adresses GPR dans le banc 0 et le banc 1 sont mappées.

Exemple: l'adresse 0Ch et 8Ch sont accédés par le même FSR.

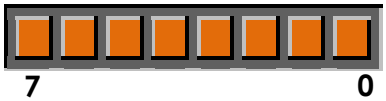
a.1. Registres spéciaux - SFR

La zone mémoire de données RAM commencée à l'adresse 00H ou 80H selon le bit RP0 et se termine à l'adresse 0BH ou 8BH contient des registres de contrôle et d'état des périphériques et des registres du noyau de l'unité centrale (STATUT, INDF, SFR, etc.).

- ✓ SFR permettent la gestion du circuit.
- ✓ Certains registres ont une fonction générale, d'autres ont une fonction spécifique attachée à un périphérique donné.
- ✓ L'ensemble de registres SFR est souvent appelé **fichier des registres**.

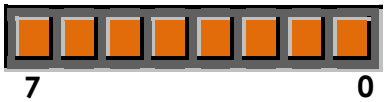
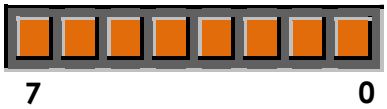
Le tableau ci-dessous illustre clairement l'ensemble des registres SFR.

Tableau 2.5 : *Registres spéciaux du PIC 16F84.*

Registre	Adresse	Description
INDF	00H - 80H	Le registre INDF est un accumulateur des registres pointé par FSR et il est utilisé pour l'accès indirect à la mémoire (adressage indirect). INDF :  7 0
TMR0	01H	Registre lié au compteur. TM0 :  7 0
PCL	02H – 82H	PCL contient les poids faibles du compteur de programmes (**PC (13 bits**). Le registre PCLATH (0AH-8AH) contient les poids forts. PCL :  7 0 PCLATH PCL PC :  12 8 7 0
PCLATH	0AH – 8AH	
STATUS	03H – 83H	Contient l'état de l'unité arithmétique et logique (**Registre d'état**) ainsi que les bits de sélection des banques.

STATUS :

Bit 0	**C**	Carry/Borrow
	1	Débordement bit du résultat de l'opération précédente
	0	Pas de débordement
Bit 1	**DC**	Digital carry / borrow\
	1	Débordement du 4eme bit du résultat de l'opération précédente
	0	Pas de débordement
Bit 2	**Z**	bit Zero
	1	Résultat de l'opération précédente nul
	0	Résultat de l'opération non nul
Bit 3	**PD**	Power down bit
	1	Après un démarrage ou après l'instruction CLRWDT
	0	Après l'exécution de l'instruction SLEEP
Bit 4	**TO**	Time Out du watch dog
	1	Après un démarrage, CLRWDT ou SLEEP
	0	Après un time-out du watch dog
Bit 5	**RP0**	Sélection de la bank de registres active
Bit 6	**RP1**	
	00	Bank0
	01	Bank1
	10	Non utilisé sur le 16F84
	11	Non utilisé sur le 16F84
Bit 7	**IRP**	Non utilisé dans le 16F84
	0	/
	1	/

FSR	**04H – 84H**	Utilisé pour l'adressage indirect. **FSR :** 
PORTA	**05H**	Donne accès en lecture ou écriture au Port A (5 bits). Les sorties sont à drain ouvert. Le bit 4 peut être utilisé en entrée de comptage. **PORT A :** 
PORTB	**06H**	Donne accès en lecture ou écriture au Port B. Les sorties sont à drain ouvert. Le bit 0 peut être utilisé en entrée d'interruption. **PORT B :** 
EEDATA	**08H**	Permet l'accès aux données dans la mémoire EEPROM.

		EEDATA : 
EEADR	**09H**	Permet l'accès aux adresses de la mémoire EEPROM. **EEADR :**  L'EEPROM ne contenant que 64 octets d'adresse 00 à 3F, les 2 bits de poids fort de EEADR sont donc toujours à 0.
INTCON	**0BH - 8BH**	Bits d'autorisation et Flags d'interruptions (Masque d'interruptions).

INTCON :

Bit 0	**RBIF**	Flag d'interruption sur le Port B
	1	Au moins une broche RB7:RB4 à changé d'état (doit être effacé par programme)
	0	Pas de changement d'état sur RB7:RB4
Bit 1	**INTF**	Flag d'interruption sur RB0/INT
	1	Il y a eu une demande d'interruption sur RB0/INT
	0	Pas d'interruption sur RB0/INT
Bit 2	**TOIF**	Flag de débordement de TMR0
	1	Le TMR0 a débordé (à effacer par programme)
	0	Le TMR0 n'a pas débordé
Bit 3	**RBIE**	Autorisation de l'interruption lors d'un changement d'état sur le PortB
	1	Autorise l'interruption sur RB7 : RB4
	0	Interdit l'interruption sur RB7 : RB4
Bit 4	**INTE**	Autorisation de l'interruption sur RB0/INT
	1	Autorise l'interruption sur RB0/INT
	0	Interdit l'interruption sur RB0/INT
Bit 5	**TOIE**	Autorisation de l'interruption de débordement de TMR0
	1	Autorise l'interruption de TMR0
	0	Interdit l'interruption de TMR0
Bit 6	**EEIE**	Autorisation de l'interruption de fin d'écriture en EEPROM
	1	Autorise l'interruption de fin d'écriture en EEPROM
	0	Interdit l'interruption de fin d'écriture en EEPROM
Bit 7	**GIE**	Autorisation Globale des Interruptions
	1	Autorise toutes les interruptions
	0	Interdit toutes les interruptions

OPTION	**81H**	Contient des bits de configuration pour divers périphériques. Le registre **OPTION** est utilisé pour configurer le **TIMER TMR0** et le **Watchdog WDT** dans le PIC 16F84.

OPTION :

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
7							0

Bit 0	PS0	PS2:PS1:PS0 Valeur du Prescaler du TMR0 ou du WDT
Bit 1	PS1	
Bit 2	PS2	

PS2:PS1:PS0	Prescaler TMR0	Prescaler WDT
000	1/2	1/1
001	1/4	1/2
010	1/8	1/4
011	1/16	1/8
100	1/32	1/16
101	1/64	1/32
110	1/128	1/64
111	1/256	1/128

Bit 3	PSA	Assignation du Prescaler
	1	Prescaler assigné au chien de garde WDT
	0	Assigné au Timer TMR0
Bit 4	TOSE	Sélection du front actif pour le comptage sur RA4/TOCKI
	1	Comptage sur front descendant
	0	Comptage sur front montant
Bit 5	TOCS	Source de l'horloge du Timer0
	1	Comptage sur la broche RA4/TOCKI
	0	Comptage sur l'horloge interne CLKOUT
Bit 6	INTEDG	Sélection du front actif de l'INT externe
	1	Interruption sur le front montant de RB0/INT
	0	Interruption sur le front descendant de RB0/INT
Bit 7	RBPU	Résistances de rappel des entrées du PortB
	1	Les résistances sont désactivées
	0	Les résistances du PortB sont activées

TRISA	85H

Indique la direction (en entrée ou en sortie) du port A.

TRISA :

-	-	-					
			4	.....		0	
			TRISA4			TRISA0	

- Le sens de fonctionnement du port A est défini par le contenu du registre TRISA:
  - ✓ Un bit à **0** La broche correspondante est programmée en **sortie.**
  - ✓ Un bit à **1**, La broche est programmée en **entrée**.

TRISB	86H

Indique la direction (en entrée ou en sortie) du port B.

		TRISB :  Le sens de fonctionnement du port B est défini par le contenu du registre TRISB: ✓ Un bit à **0** La broche correspondante est programmée en **sortie.** ✓ Un bit à **1**, La broche est programmée en **entrée**.
EECON1	**88H**	Permet le contrôle d'accès à la mémoire EEPROM (Définit le mode de fonctionnement de l'EEPROM).

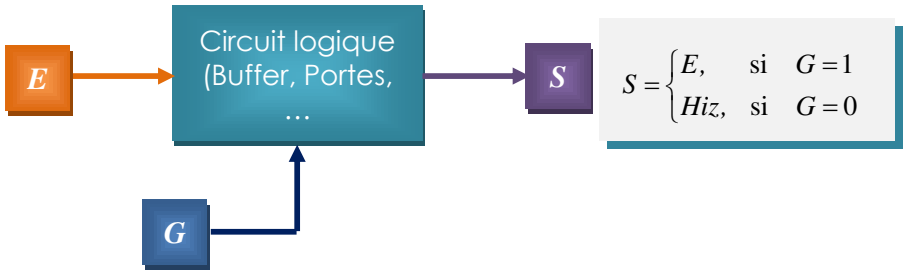
EECON1 :

Bit 0	**RD**	Read Data (Lecture d'une donnée)
	1	Lire une donnée auprès de l'EEPROM
	0	Interdit la lecture de la donnée auprès de l'EEPROM
Bit 1	**WR**	WRite data (Ecriture des données)
	1	Autoriser l'écriture des données dans l'EEPROM
	0	Interdit l'écriture des données dans l'EEPROM
Bit 2	**WREN**	WRite Enable (Activer le mode écriture dans l'EEPROM)
	1	Autoriser l'écriture des données dans l'EEPROM
	0	Interdit l'écriture des données dans l'EEPROM
Bit 3	**WRERR**	Write ERRor (Signaler en cas d'erreur d'écriture dans l'EEPROM)
	1	Il y a eu une erreur d'écriture dans l'EEPROM
	0	Pas d'erreur d'écriture dans l'EEPROM
Bit 4	**EEIF**	EEPROM Interrupt Flag (Ce bit doit être remis à 0 par le programme)
	1	L'écriture d'une donnée dans l'EEPROM est terminée
	0	L'écriture d'une donnée dans l'EEPROM n'est terminée
Bit 5	/	Non utilisé dans le PIC 16F84
Bit 6	/	Non utilisé dans le PIC 16F84
Bit 7	/	Non utilisé dans le PIC 16F84

EECON2	**89H**	Le registre EECON2 d'adresse 89H n'est pas physiquement implanté et ne sert qu'à la sécurisation lors de la phase d'écriture dans l'EEPROM.

TRISA ou **TRISB**, que signifient ???

✓ Le terme **TRIS** vient de fonctionnement de circuit à 3 états (**Tri-States**) : **ON**, **OFF** et **Haute Impédance (Hiz).**



Comment peut-on **LIRE une donnée** de la mémoire **EEPROM** de données ?

1. Placer l'adresse relative dans **EEADR**.
2. Mettre le bit **RD** de **EECON1** à **'1'**.
3. Lire le contenu du registre **EEDATA**.

Comment peut-on **ECRIRE une donnée** sur la mémoire **EEPROM** de données ?

1. L'écriture dans L'EEPROM doit être autorisée : **WREN = '1'**.
2. Placer l'adresse relative dans **EEADR**.
3. Placer la donnée à écrire dans **EEDATA**.
4. Placer **55H** dans **EECON2**.
5. Placer **AAH** dans **EECON2**.
6. Démarrer l'écriture en positionnant le bit **WR**.
7. Attendre la fin de l'écriture, (10 ms) (**EEIF = '1'** ou **WR = '0'**).
8. Recommencer au point 2, si on a d'autres données à écrire.

À la lecture de la signification des bits du registre **INTCON** (gestion des interruptions), on notera que le PIC 16F84 possède 4 sources potentielles d'interruptions:

- ✓ Un changement d'état sur les broches RB4 à RB7.
- ✓ Une source externe via la broche RB0/INT.
- ✓ Débordement du **Timer 0** ;
- ✓ La fin de l'écriture dans l'EEPROM de données.

- Chaque indicateur de changement d'état **RBIF, INTF, RTIF** doit être remis à 0 par le logiciel dans le programme de traitement de l'interruption.
- Lors d'un RESET, tous les bits du registre **INTCON** sauf **RBIF** sont mis à 0 → RBIF garde son état précédent.
- Un seul vecteur d'interruption étant disponible à l'adresse 0004, le programme d'interruption doit déterminer éventuellement quelle est la source d'interruption.

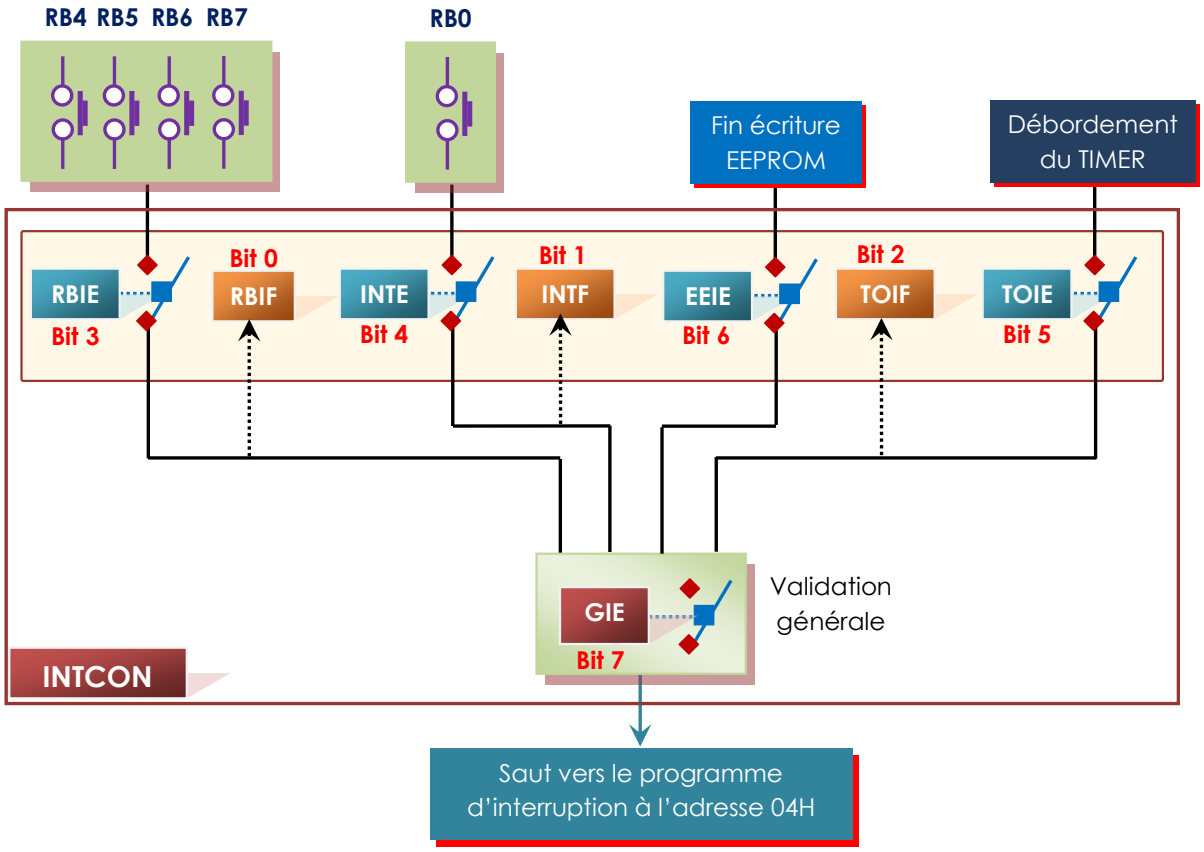


Figure 2.21 : *Synoptique du mode de contrôle des interruptions.*

a.2. Registre de configuration

Une partie importante de fonctionnement du PIC 16F84 est contrôlée par les bits du registre (mot) de configuration qui se trouve à l'adresse 2007H.

Bit 0	FC0	Type de l'horloge
Bit 1	FC1	

FC1:FC0	Description
00	Oscillateur quartz basse consommation (centaines de KHz)
01	Oscillateur quartz standard (centaines de KHz à 4 MHz)
10	Oscillateur quartz haute vitesse (4 à 20 MHz)
11	Oscillateur RC

Bit 2	WDTE	WatchDog Timer Enable : Active le chien de garde

1	Activer le chien de garde
0	Désactiver le chien de garde

Bit 3	PWRTE	*Power-up Timer* Enable : Activer / Désactiver le Power-up timer et génère un délai de 72 ms après un RESET, permettant éventuellement à la tension V_{DD} d'atteindre un niveau acceptable.

1	Activer le *Power-up timer*

		0 Désactiver le *Power-up timer*
Bit 4 … Bit 13	CP	Code Protection : Protège le PIC en lecture
		1 Désactiver le mode de protection du mémoire programme 0 Tous les mémoires programme sont protégées en lecture

a.3. Les PORTS d'Entrées / Sorties

Pour dialoguer avec l'extérieur (application) le PIC 16F84 vous met à disposition 13 Entrées / Sorties programmables individuellement soit en entrée soit en sortie. Ces 13 E/S sont issues de 2 ports nommés PORT A pour les 5 E/S RA0 à RA3 et PORT B pour les 8 E/S RB0 à RB7.

- Le port A est un port d'E/S de 5 bits (RA*i* (**i** = 0, 1, 2, 3, 4)).
  - ✓ La configuration de direction pour chaque bit du port est déterminée avec le registre TRISA.
    - Bit *i* de TRISA = 0 → RA*i* de PORTA configuré en **sortie**.
    - Bit *i* de TRISA = 1 → RA*i* de PORTA configuré en **entrée**.
  - ✓ La broche RA4 est multiplexée avec l'entrée horloge du Timer TMR0. Elle peut être utilisée soit:
    - Comme E/S normale du port A.
    - Comme entrée horloge pour le Timer TMR0.
    - Le choix se fait à l'aide du bit **T0CS** du registre **OPTION**.

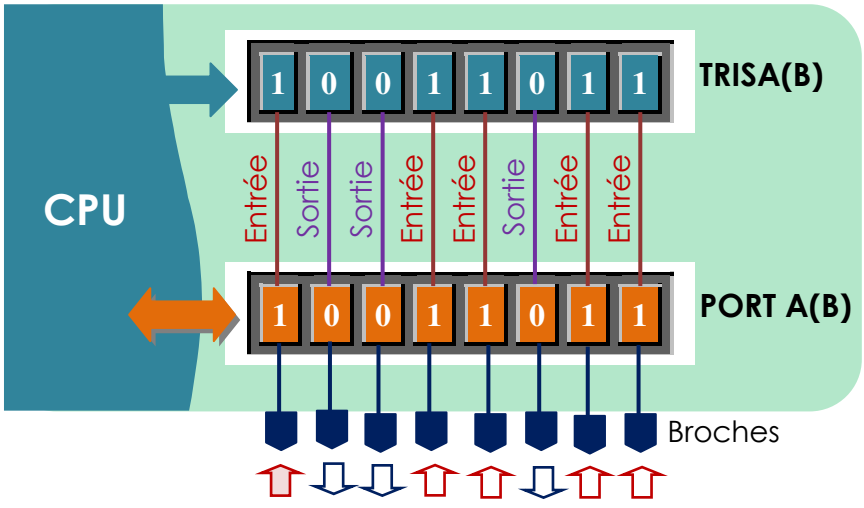


Figure 2.21 : *Exemple de configuration des ports d'E/S.*

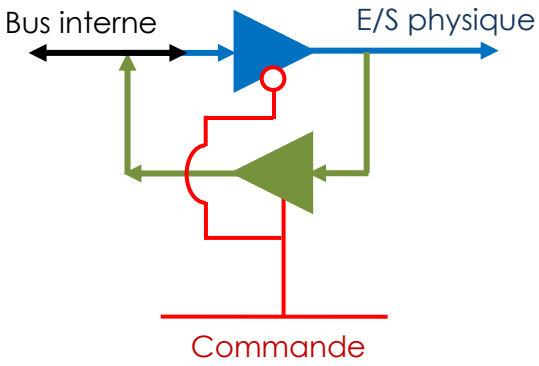


Figure 2.22 : *Exemple d'une ligne bidirectionnelle.*

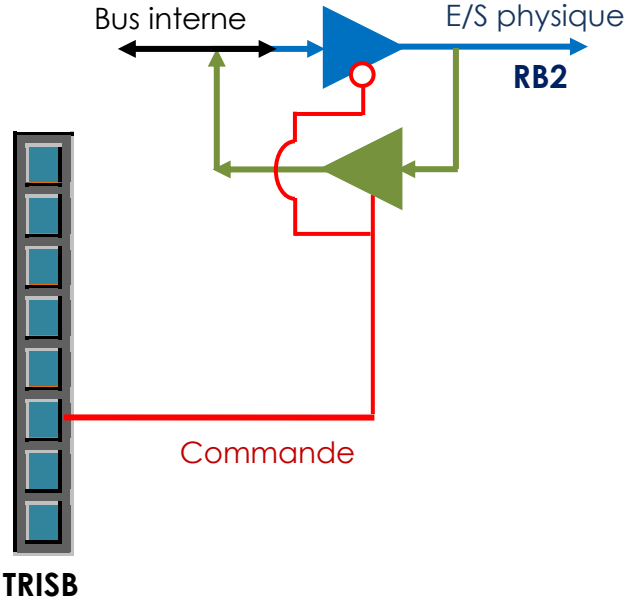


Figure 2.23 : *Activation des lignes d'E/S.*

✓ RA4 est une E/S à drain ouvert, si on veut l'utiliser comme sortie (pour allumer une LED par exemple), il ne faut pas oublier de mettre une résistance externe vers Vdd.

- Si RA4 est positionnée à 0, l'interrupteur est fermé, la sortie est reliée à la masse.
- Si RA4 est placée à 1, l'interrupteur est ouvert, la sortie est déconnectée d'où la nécessite de la résistance externe.

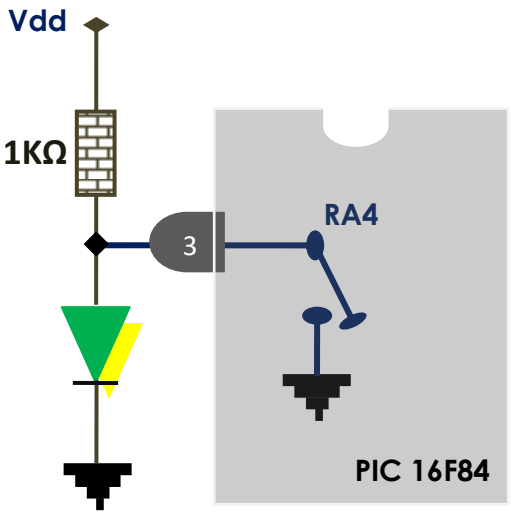


Figure 2.24 : *E/S à drain ouvert.*

- Le port B désigné par PORTB est un port bidirectionnel de 8 bits ((RB*i* (**i** = 0, 1, …,7)).
  - ✓ La configuration de direction se fait à l'aide du registre TRISB.
    - Même configuration que le PORTA.
  - ✓ La broche RB0 peut aussi servir d'entrée d'interruption externe INT.
  - ✓ Les quatre bits de poids fort (RB7-RB4) peuvent être utilisés pour déclencher une interruption RBI sur changement d'état.
  - ✓ Toutes les broches sont compatibles TTL.

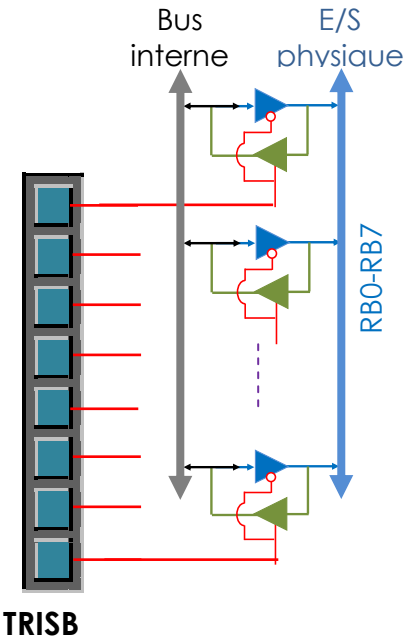


Figure 2.25 : *Configuration en E/S du PORT B.*

- Lors d'un *Reset,* les bits des registres TRISA et TRISE étant mis à 1, toutes les broches des ports A et B sont initialement des entrées.
- En entrée, la broche RA4 peut être utilisée soit comme E/S numérique normale, soit comme entrée horloge pour le TIMER TMR0.
- En sortie, RA4 est une E/S à drain ouvert (open drain), pour l'utiliser comme sortie logique (0/1), il faut ajouter une résistance de pull-up externe (470 ohms–4.7 K ohms), dans ce cas le courant provient de la résistance pull-up.
- La broche RA4/T0CKI est une entrée Trigger de Schmitt à sortie à drain ouvert. Toutes les autres broches RA*i* sont des entrées TTL est des sorties à driver CMOS.

a.4. Le TIMER

Le PIC 16F84 dispose de deux TIMERS :

- ✓ Un à usage général (le **TMR0**) et

- ✓ Un autre utilisé pour le chien de garde (watch dog **WDG**).

a.4.1. Le TIMER TMR0

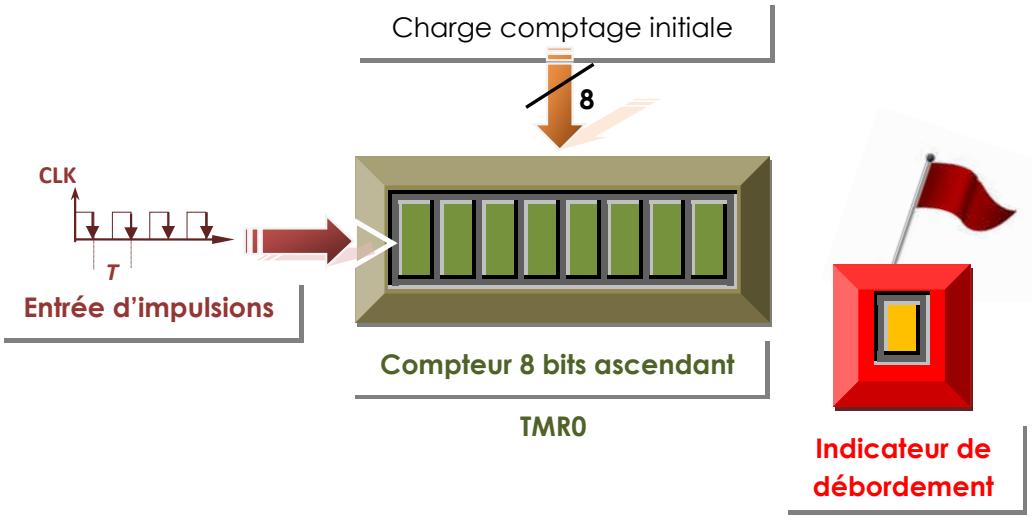
Le TIMER TMR0 permet au PIC 16F84 le contrôle (la gestion) du temps pour assurer le bon fonctionnement.

- ✓ Le TMR0 est un compteur 8 bits qui peut être chargé avec une valeur initiale quelconque.
- ✓ Il est ensuite incrémenté à chaque Top **d'horloge** (front montant (ou descendant)) du signal qui lui est appliqué jusqu'à ce que le débordement ait lieu (passage de FF à 00).
- ✓ Le choix de l'horloge se fait à l'aide du bit **T0CS** du registre **OPTION**.
  - **TOCS = 0** → Horloge interne Fosc/4 (**Mode TIMER ou temporisateur ou contrôle du temps**). Il est possible d'utiliser un prédiviseur de fréquence.
  - **TOCS = 1** → Horloge externe, appliquée à l'entére **RA4/TOCK1** du port A (**Mode compteur d'événements**).

Donc, le TIMER **TMR0** peut remplir deux fonctions:

- ✓ Temporisateur ou contrôle du temps. Son entrée d'incrémentation est alors l'horloge qui correspond au cycle instruction (Fosc/4).
- ✓ Compteur d'événements. Dans ce cas les d'impulsions d'entrées du TIMER sont fournies par la broche RA4/TOCK1.

Le principe de fonctionnement est représenté sur la figure ci-dessous.



(a)

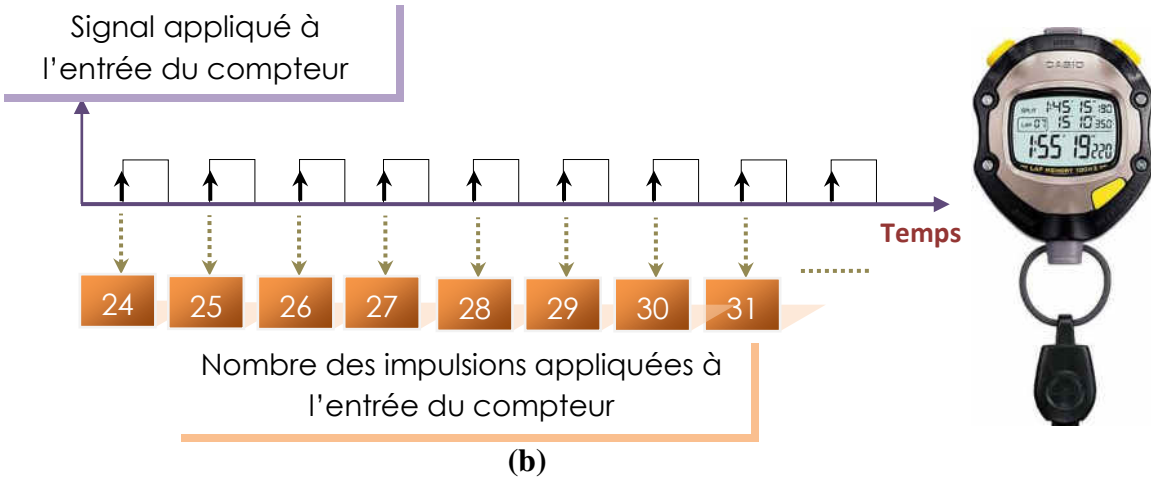


Figure 2.25 : *Principe de fonctionnement du TMR0 du PIC 16F84.*

Info

- Par définition, un **TIMER** doit compter un temps défini par le programme (par exemple 1ms, 10ms, 50ms, etc). Pour cette raison, 2 paramètres peuvent être configurés, à savoir :
  - ✓ La fréquence du signal appliqué à l'entrée du compteur (prescaler).
  - ✓ Le nombre des impulsions à compter.
- Le contenu du **TIMER TMR0** dans le cas du PIC 16F84 est accessible par le registre qui porte le même nom (adresse 01H dans SFR). Il est accessible en lecture comme en écriture à n'importe quel moment. Après une écriture, l'incrémentation est inhibée pendant deux cycles instruction.
- Le PIC 16F84 dispose d'un seul diviseur de fréquence (prédiviseur) qui peut être assigné soit au chien de garde **WDG**, soit au **TMR0** (uniquement un à la fois). L'assignation du prédiviseur se fait grâce au bit **PSA** du registre **OPTION**.
- Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport est fixés par les bits **PS0**, **PS1** et **PS2** du registre **OPTION**.
- La structure interne du TMR0 est donc la suivante :

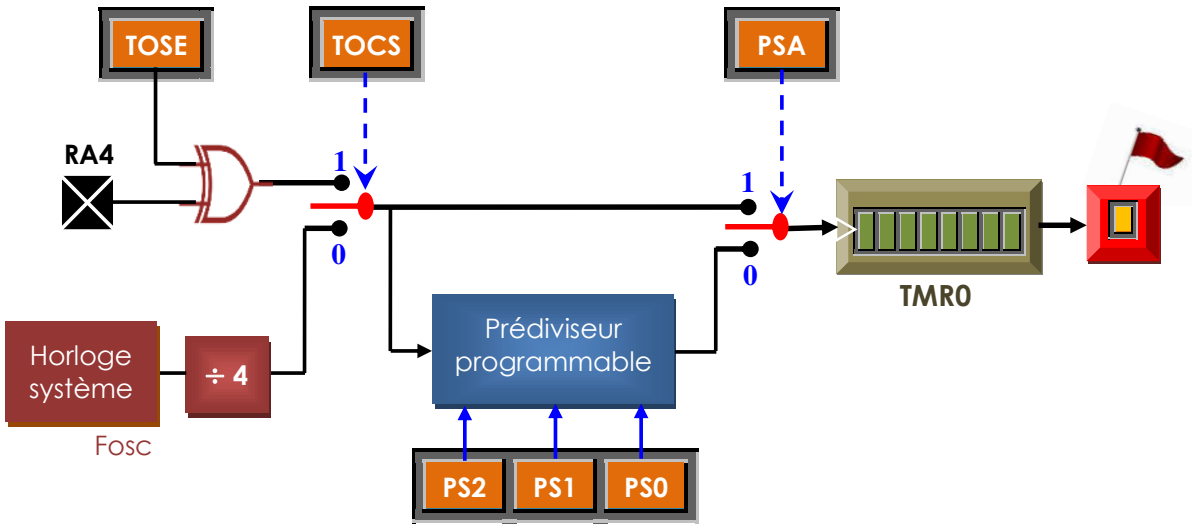


Figure 2.25 : *Structure interne du TMR0 du PIC 16F84.*

OPTION :

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
7	6	5	4	3	2	1	0

PS0 **PS1** **PS2**	**Valeur du Prescaler du TMR0 ou du WDT**		
	PS2:PS1:PS0	Prescaler TMR0	Prescaler WDT
	000	1/2	1/1
	001	1/4	1/2
	010	1/8	1/4
	011	1/16	1/8
	100	1/32	1/16
	101	1/64	1/32
	110	1/128	1/64
	111	1/256	1/128
PSA	Assignation du Prescaler		
	1	Prescaler assigné au chien de garde **WDT**	
	0	Assigné au Timer **TMR0**	
TOSE	Sélection du front actif pour le comptage sur **RA4/TOCKI**		
	1	Comptage sur front descendant	
	0	Comptage sur front montant	
TOCS	Source de l'horloge du Timer0		
	1	Comptage sur la broche **RA4/TOCKI**	
	0	Comptage sur l'horloge interne **CLKOUT**	

Au débordement de TIMER TMR0, le flag **T0IF** du registre **INTCON** est placé à **1**. Ceci peut déclencher l'interruption **T0I** si celle-ci est validée.

- A la mise sous tension, le registre TMR0 contient une valeur indéterminée. Donc, pour commencer à partir d'une valeur précise (0 ou autre), il faut placer explicitement cette valeur dans le programme (phase de configuration du PIC).
- Le prédiviseur programmable compte jusque sa valeur programmée par les PS0, PS1 et PS2.
  - ✓ Une fois cette valeur préprogrammée est atteinte, TMR0 sera incrémenté.
  - ✓ Le prédiviseur recommence son comptage à l'impulsion suivante.
- La durée nécessaire pour que le TIMER génére un **débordement** (overflow) peut être calculée par la formule suivante:

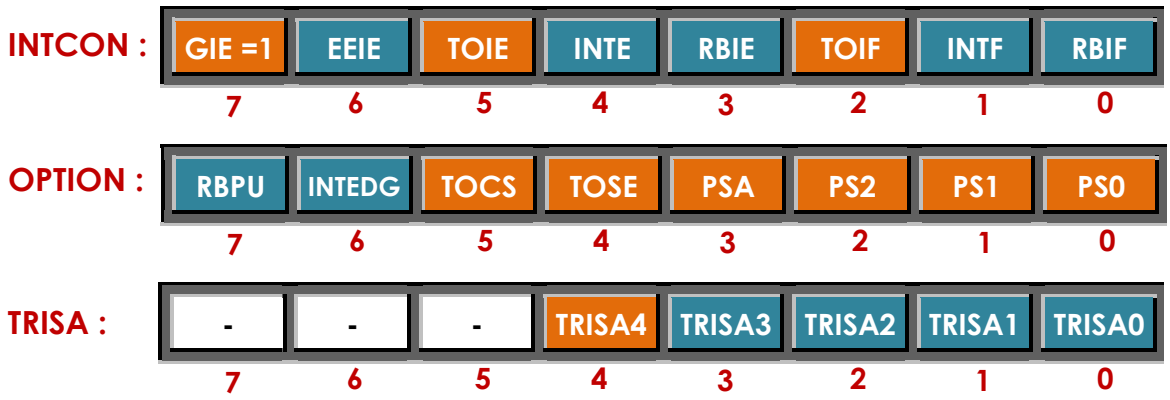
Overflow time = 4 * TOSC* Prescaler* (256 – TMR0)

- Overflow time en µs ;
- **TOSC** est la période d'oscillation = 1/**Fosc** ;
- Prescaler est le pré-diviseur choisi par le registre **OPTION** ;
- **TMR0** est la valeur chargée dans le registre **TMR0** (*valeur initiale*).

- En mode compteur et pour ne pas rater une impulsion, il faut

Tsignal > Tcyc = 4*Tosc

- La gestion du TMR0 est effectuée par les trois SFR suivants :



a.4.2. Le chien de garde (Watchdog TIMER)

Le **WatchDog Timer (WDT)** ou chien de garde est un mécanisme pour protéger le programme dans le PIC 16F84. Son rôle est de surveiller si le programme s'exécute toujours dans l'espace et dans le temps attribués, c'est-à-dire

- Le programme ne s'est pas «égaré» dans une zone non valide du programme (parasite sur l'alimentation), ou
- S'il n'est pas bloqué dans une boucle infinie (bug du programme).

Autrement dit,

Le chien de garde (watchdog) est un dispositif matériel et logiciel qui permet de se prémunir contre les plantages accidentels. L'idée est de provoquer un RESET du CPU afin de relancer l'application. (Les données sont bien sur perdues). Le plantage est défini lorsque le programme n'est pas venu à temps faire signe au watchdog.

Figure 2.25 : *Principe de WatchDog Timer du PIC 16F84.*

- ✓ C'est un compteur 8 bits incrémenté en permanence (même si le PIC 16F84 est en mode **SLEEP**) par une horloge RC) intégrée indépendante de l'horloge système.
- ✓ **WatchDog Timer** sert également à réveiller un PIC placé en mode SLEEP.
- ✓ Lorsqu'il déborde (**WDT TimeOut**), deux situations sont possibles :
  - ✓ Si le PIC 16F84 est en fonctionnement normal, le **WDT TimeOut** provoque un **RESET**. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur par un processus indésirable non contrôlé.
  - ✓ Si le PIC 16F84 est en mode **SLEEP**, le **WDT TimeOut** provoque un **WAKE-UP**, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode **SLEEP**. Cette situation est souvent exploitée pour réaliser des temporisations.

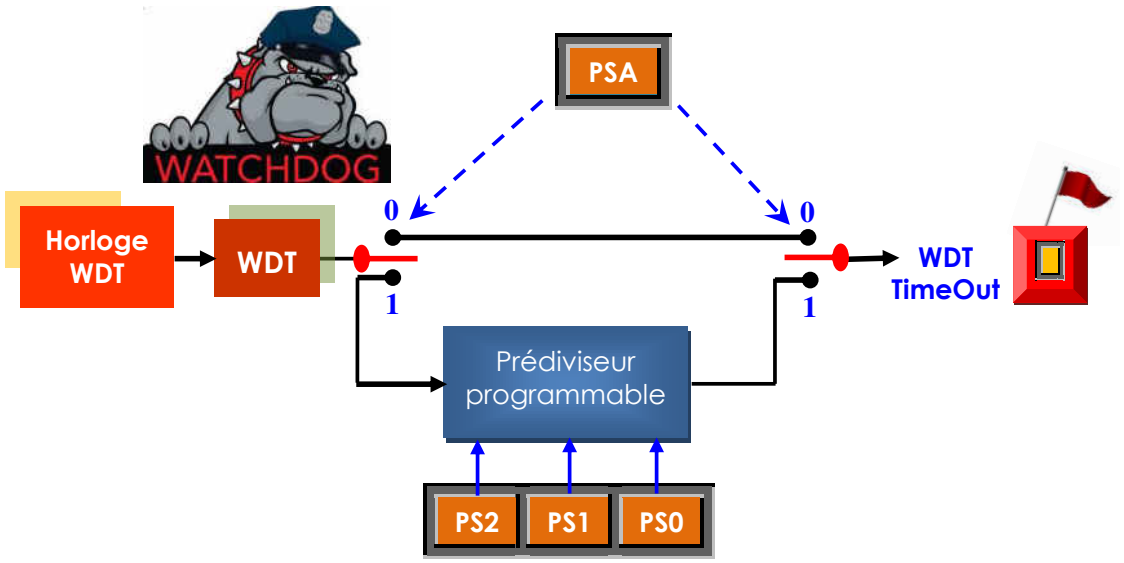


Figure 2.26 : *Fonctionnement de Watchdog TIMER du PIC 16F84.*

Précautions d'emploi

L'utilisation du **WDT** doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un **WDT** timeOut lors de l'exécution d'un programme, on a deux possibilités :

- ✓ Inhiber le **WDT** d'une façon permanente en mettant à '**0**' le bit **WDTE** dans le registre de configurarion **CONFIG_**.
- ✓ Remettre le **WDT** à '**0**' périodiquement dans le programme à l'aide de l'instruction **CLRWDT** pour éviter qu'il ne déborde au cours d'exécution.
- ✓ En cas de plantage, le timer déborde ce qui génère un **RESET**, et le programme redémarre.

Le fonctionnement du **WDT** est lié à un TIMER interne spécifique, les durées de débordements autorisées (valeurs qui varient avec les paramètres (tension, température, etc)) sont données dans le tableau ci-dessous. Ces durées dépendent des bits PSA, PS2, PS1 et PS0 du registre **OPTION**.

Tableau 2.5 : *Durées indicatives de débordement de WDT du PIC 16F84.*

PSA	PS2, PS1, PS0	Taux de prédivision du Watchdog	Durée indicative
0	XXX	1	18 ms
1	000	1	18 ms
1	001	2	36 ms
1	010	4	72 ms
1	011	8	144 ms
1	100	16	288 ms

1	101	32	576 ms
1	110	64	1,15 s
1	111	128	2,3 s

* x = '0' ou '1'

	Durée spécifique de débordement [**ms**] **(WatchDog Timer TimeOut Period (No Presscaler))**			
Paramètres pratiques	Min	**Typique**	Max	Condition
WDT_TimeOut	**7 ms**	**18 ms**	**33 ms**	**Vdd = 5.0V**

La mise en service (resp. son arrêt) du **WDT** se fait au moment de la programmation par la directive:

_CONFIG _WDT_ON (_OFF) ;

a.4.3. Mode SLEEP

Le PIC peut être placé en mode de consommation réduite à l'aide de l'instruction **SLEEP**.

- Dans ce mode, l'horloge système est arrêtée ce qui arrête l'exécution du programme.
- Pour sortir du mode **SLEEP**, il faut provoquer un **WAKE-UP**, soit par programme (Soft) soit par un signal externe (Hard):

1. **RESET externe** (Hard) dû à l'initialisation du PIC en mettant l'entrée MCLR à 0 Niveau bas sur le pin MCLR ce qui provoque un reset à l'adresse 0000. Le PIC reprend l'exécution du programme à partir du début.
2. **TimeOut du WDT** (Soft) si celui-ci est validé. Le PIC reprend le programme à partir de l'instruction qui suit l'instruction SLEEP.
3. **Interruption INT** (sur RB0 ou **RBI** (sur RB4-RB7)) (HARD) ou **EEI** (fin d'écriture dans l'EEPROM de données) (Soft). Le bit de validation de l'interruption en question doit être validé, par contre, le WAKE-UP a lieu quelque soit la position de bit de validation globale **GIE**. On a alors deux situations :
   - **GIE = 0**, Le PIC reprend l'exécution du programme à partir de l'instruction qui suit l'instruction **SLEEP**, l'interruption n'est pas prise en compte.
   - **GIE = 1**, Le PIC exécute l'instruction qui se trouve juste après l'instruction **SLEEP** puis se branche à la fonction **void interrupt ()** pour exécuter la routine d'interruption (soubroutine).

Utilisations Typiques

Cas typiques d'utilisation du mode SLEEP :

Ce mode de fonctionnement est principalement utilisé dans les applications dans lesquelles la consommation en énergie doit être économisée. On

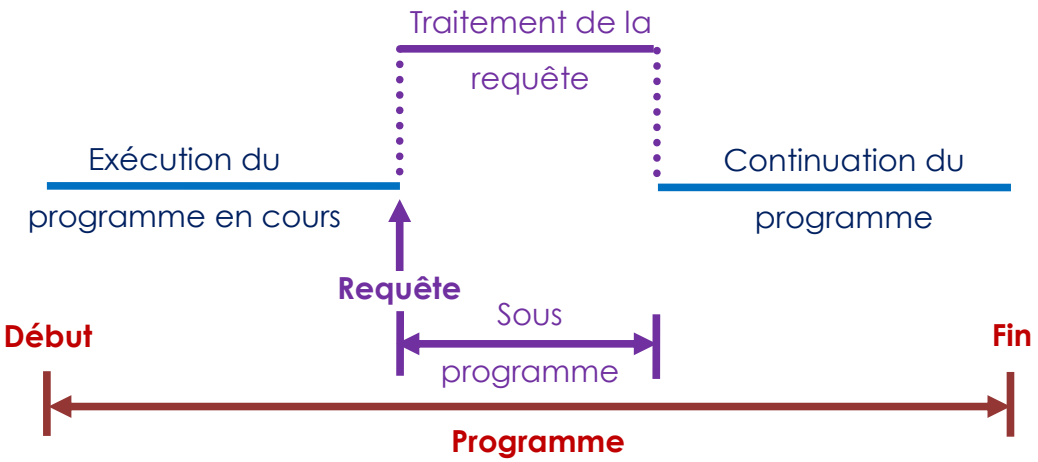
placera donc dans ce cas le PIC en mode **POWER DOWN** ou **SLEEP** aussi souvent que possible.

- Une autre application typique est un programme dans lequel le PIC n'a rien à faire dans l'attente d'un événement extérieur particulier. Dans ce cas, une instruction **SLEEP** fera efficacement l'affaire au lieu de mettre des boucles sans fin.
- **Attention**: Avant d'exécuter l'instruction SLEEP il faut s'assurer que les **flags** sont **effacés**.

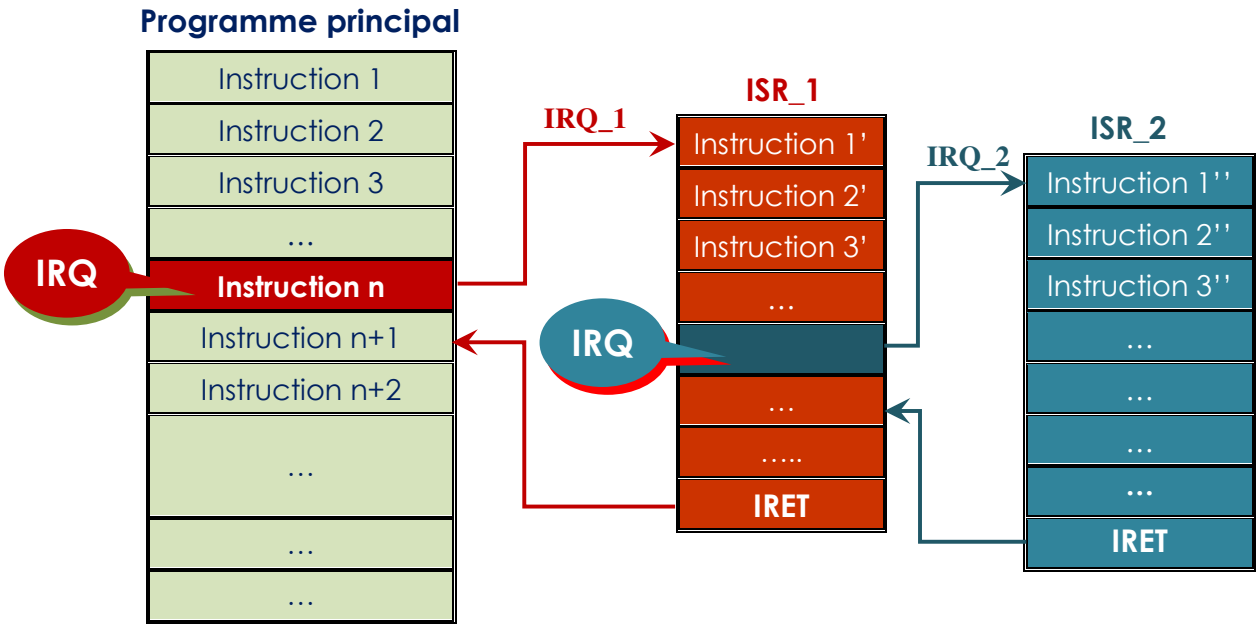
2.4. Gestion des interruptions

L'interruption est un mécanisme fondamental dans le fonctionnement des microcontrôleurs. Elle permet de prendre en compte des événements extérieurs et de leurs associer un traitement particulier. Autrement dit, une interruption est un **signal déclenché** par un **événement interne** ou **externe** (envoyé au microcontrôleur par un périphérique ou un programme) pour l'avertir d'un évènement à traiter (**demande d'interruption ou requête d'interruption**).

- ✓ Une interruption (interne (Soft) ou externe (Hard)) provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption (void interrupt()). A la fin de cette procédure, le microcontrôleur reprend le programme à l'endroit où il s'était arrêté. La pile (stack) sert donc à sauvegarder les adresses de branchement vers les routines des interruptions.
- ✓ La séquence classique de fonctionnement d'une interruption est illustrée par le schéma de la figure ci-dessous.



(a)



* **IRQ :** Interrupt Request (Demande d'interruption → Evénement déclencheur).
* **ISR :** Interrupt Service Routine (Exécution de programme de l'interruption).
* Toute interruption démarre à partir de l'adresse 04H.

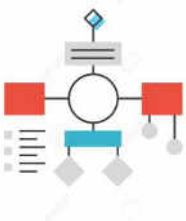
(b)

Figure 2.27 : *Principe de l'exécution d'une interruption.*

En résumé

L'algorithme suivant décrire les étapes de déroulement de l'exécution d'une interruption.

Algorithme 2.1 : *Etapes de déroulement d'une interruption.*



1. Terminer l'instruction en cours
2. Sauvegarder l'état du processeur.
3. Interdire les interruptions de niveau moins élevé.
4. Accéder au ISR.
5. Exécuter le ISR.
6. Restituer l'état du processeur.
7. Reprendre l'exécution du programme abandonné.

✓ A chaque interruption sont associés deux bits:
- Un bit de validation (**E** : **Enabale**) permet d'autoriser ou non l'interruption et
- Un drapeau (**F** : **Flag**) permet au programmeur de savoir de quelle interruption il s'agit.

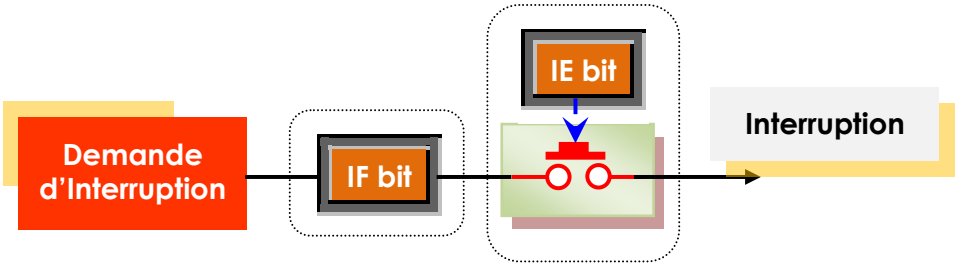


Figure 2.28 : *Mécanisme matériel d'une interruption.*

✓ Le registre **INTCON** en lecture écriture permet de configurer les différentes sources d'interruption dans le cas du PIC 16F84. Le PIC16F84 possède **4** sources d'interruption, illustrées dans le tableau ci-dessous. A chaque interruption sont associés deux bits schémartisés par la figure 2.28.

Tableau 2.5 : *Sources d'interruption pour le PIC 16F84.*

Source	Description	
TMR0	Débordement du TIMER 0 (TMR0)	Une fois que le contenu du **TMR0** passe de *FFH à 00H*, une interruption peut être générée
EEPROM	Ecriture dans l'EEPROM de données	Cette interruption peut être générée lorsque l'écriture dans une case EEPROM interne est terminée.
RB0/INT	Interruption sur la broche RB0 du PORT B	Une interruption peut être générée lorsque, la broche **RB0**, encore appelée **INTerruptpin**, étant configurée en entrée, le niveau qui est appliqué est modifié.
PORTB	Interruption sur le Port B (**RB4 - RB7**)	Une interruption peut être générée lors du changement d'un niveau sur une des broches **RB4 - RB7**. Il n'est pas possible de limiter l'interruption à une seule de ces broches. L'interruption sera effective pour les 4 broches ou pour aucune

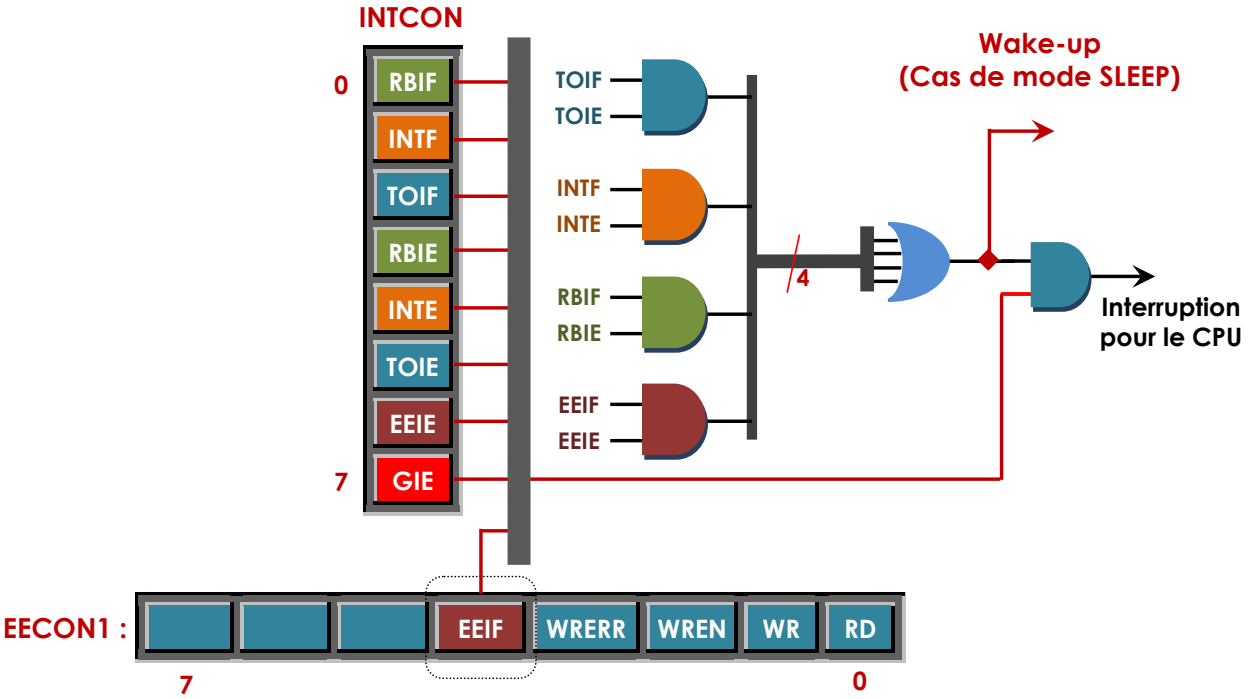
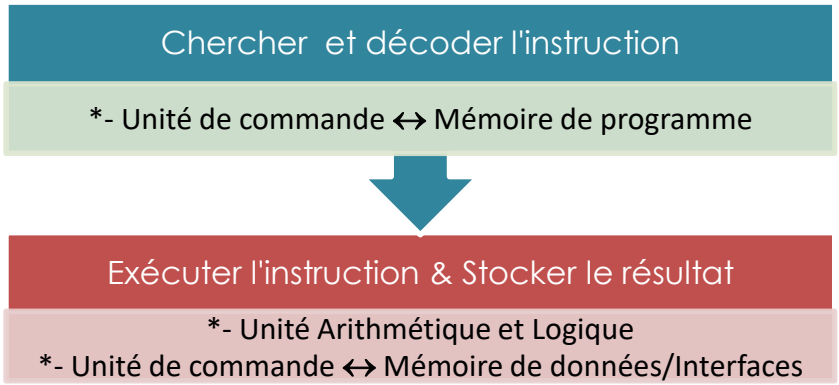


Figure 2.28 : *Logique des interruptions dans le PIC 16F84.*

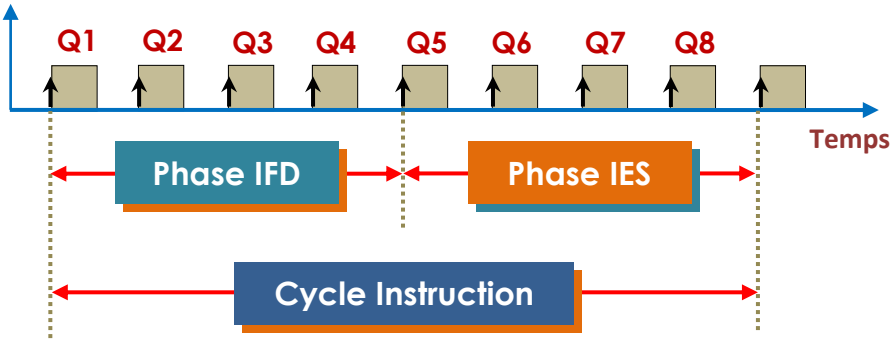
2.5. Principe de fonctionnement du PIC

La tâche d'un un microcontrôleur est d'exécuter des programmes (Ensemble des instructions) répondant au cahier de charge de l'utilisateur. Dans le cas de l'architecture RISC, Une instruction est exécutée en deux phases, où chaque phase dure 4 cycles d'horloge:

1. **Phase IFD** = Phase de recherche et de décodage de l'instruction (Fetch and Decode (IFD));
2. **Phase IES** = Phase d'exécution effective de l'instruction et stockage le résultat (Execute and Save (IES));



(a)



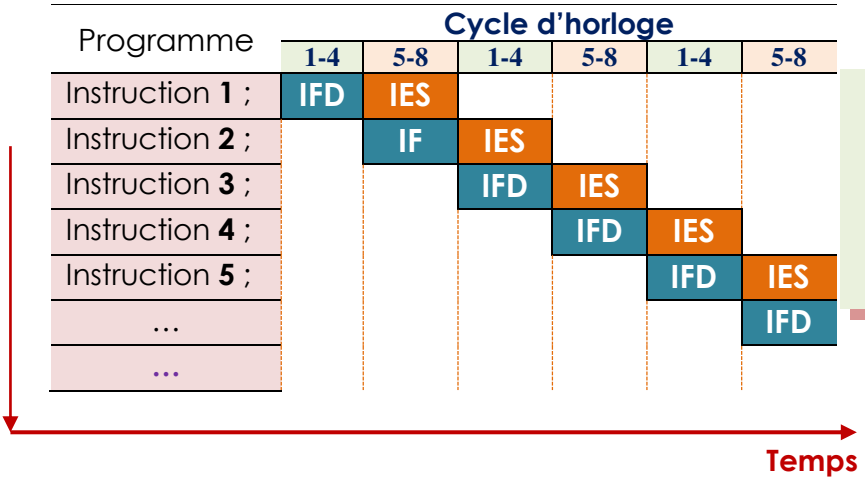
(b)

Figure 2.8 : *Etapes d'exécution d'une instruction dans le cas de PIC 16F84.*

Info

- Instruction = 02 phases (**IFD**, **IES**);
- Phase = ensemble des opérations élémentaires (micro opérations (µop)) ;
- 1 µop = 1 période d'Horloge ;
- 1 Cycle machine = Phase **IFD** + Phase **IES** ;
  = Temps d'exécution d'une instruction ;
- L'enchaînement des phases est cadencé par les TOPs (coups) d'horloge.
- Chaque instruction possède son propre temps d'exécution,(Généralement 8 cycles d'horloge) ;

L'architecture particulière des PICs (Architecture de Harvard ou les bus sont différents pour les données et les programmes) lui permet de réduire ce temps par deux tout en utilisant la notion de pipeline à 2 étages comme montre le schéma de la figure ci-dessous.



Ce type de pipeline est notamment utilisé sur certains **microcontrôleurs Atmel AVR et PIC**.

2.5.1. Déroulement d'un programme

Le déroulement d'un programme s'effectue d'une manière séquentielle. A la mise sous tension, le processeur va

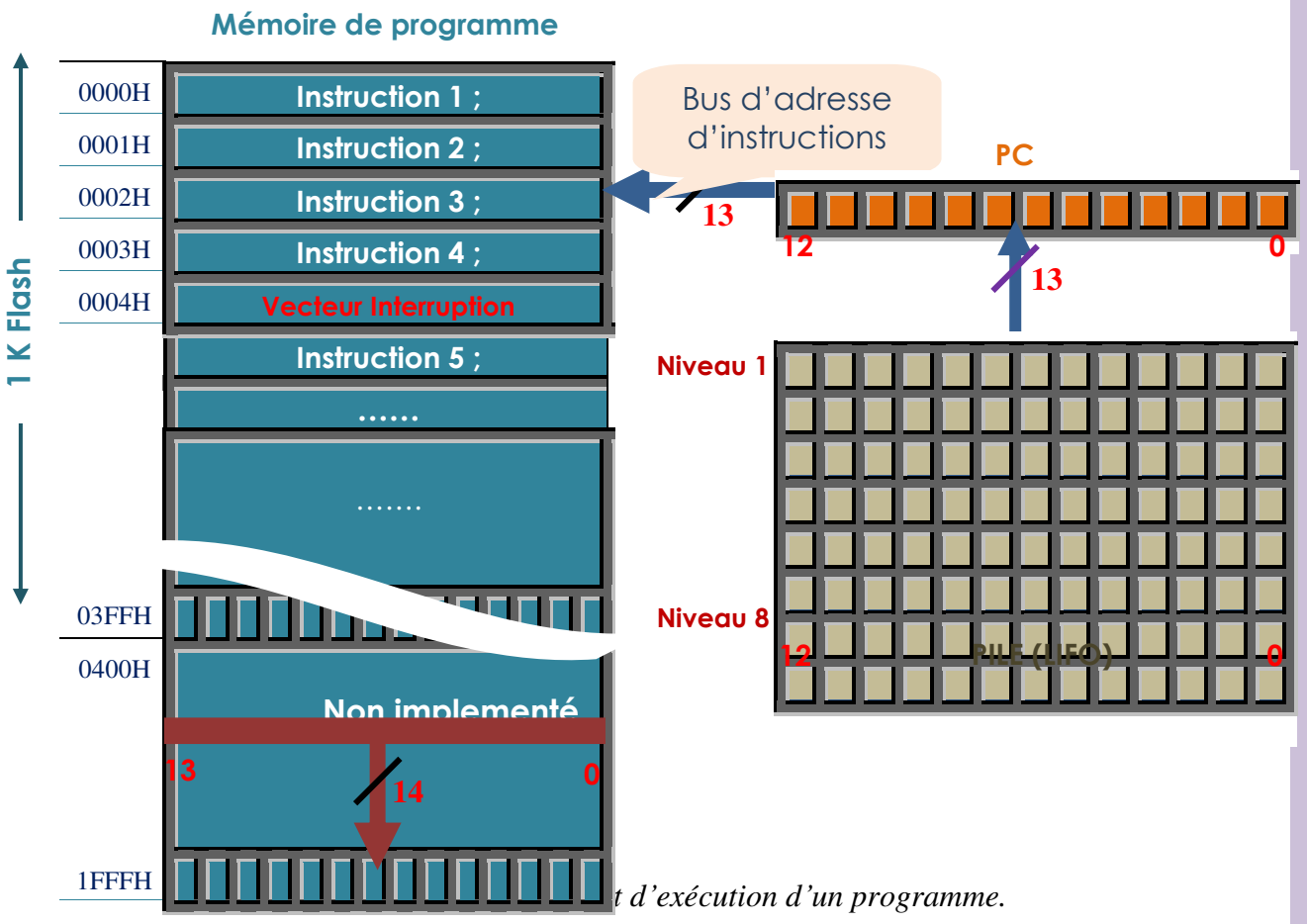
1. Chercher la première instruction qui se trouve à l'adresse 0000H de la mémoire de programme,
2. Exécuter cette première instruction.

3. Chercher la deuxième instruction à l'adresse 0001H et ainsi de suite.

Cas Particulier

En cas de branchement vers des sous programmes ou le cas des interruptions, le processeur termine l'exécution de l'instruction en cours, puis mettre l'adresse la prochaine instruction dans la PILE, l'adresse de branchement dans le PC et aller exécuter le sous programme demandé. Une fois termine, il va récupérer l'adresse de retour à partir de la PILE et continu l'exécution du reste du programme.

La figure ci-dessous illustre le déroulement d'exécution des programmes.



t d'exécution d'un programme.

Attention

Il est à noter que la PILE possède uniquement huit niveaux (cases mémoires), cela signifie qu'il n'est pas possible d'imbriquer plus de 8 sous programmes. Car au-delà de 8, le processeur ne sera plus capable de retourner à l'adresse de base du programme principal.

- L'adresse **0000H** est réservée au vecteur de démarrage **RESET**, cela signifie que c'est à cette position que l'on accède chaque fois qu'il se produit une réinitialisation (0 volts sur la patte **MCLR**). C'est pour cette raison que le programme de fonctionnement du microcontrôleur doit toujours démarrer à partir de cette adresse.
- L'adresse **0004H** est assignée au vecteur **d'interruption** et fonctionne de manière similaire à celle du vecteur de **RESET**. Quand une interruption est produite et validée, le compteur ordinal **PC** se charge avec **0004H** et l'instruction stockée à cet emplacement sera exécutée.

2.6. Mise en œuvre

L'utilisation et la mise en œuvre très simple des PICs les a rendus extrêmement populaire. Il est à noter que la société **MICROCHIP**, fabricant des PIC est en passe de devenir le leader mondial dans le domaine des microcontrôleurs devant d'autres concurrent à savoir **MOTOROLA**, **INTEL**, etc.

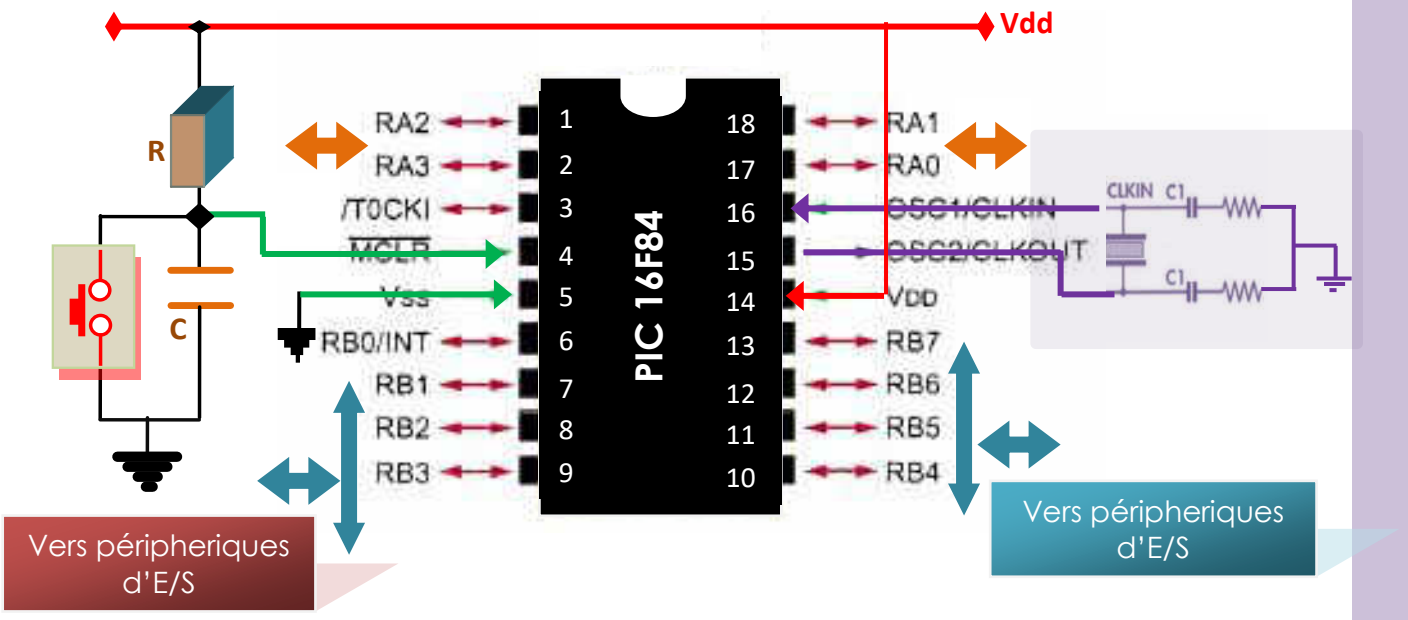


Figure 2.8 : *Mise en œuvre d'un PIC 16F84.*

Pour faire fonctionner le PIC, il suffit de:

- D'alimenter le circuit par ses deux broches **Vdd** et **Vss**.
- De fixer la vitesse de fonctionnement volue à l'aide d'un **QUARTZ**.
- De réaliser un système de réinitialisation (**RESET**) du microcontrôleur sans avoir à couper l'alimentation.
- D'écrire le programme en langage assembleur ou en **C** (**Mikro C**) sur un ordinateur grâce au logiciel **MPLAB** de **MICROCHIP** (logiciel gratuit).
- De compiler pour le transformer en langage machine.
- De transférer le fichier résultant (programme de gestion de l'application développée) dans le PIC grâce à un programmateur.

De rendre le système opérationnel.

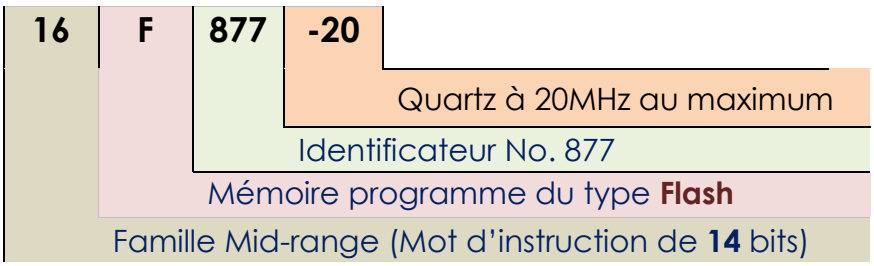
Les caractéristiques du PIC 16F84 fournissent par Microchip :

- ✓ Mémoire de programme : 1KO, type Flash.
- ✓ Mémoire de données RAM : 68 octets.
- ✓ Mémoire de données E²PROM : 64 octets.
- ✓ Niveau de la pile : 8.
- ✓ Jeux d'instruction RISC : 35 de 14 bits.
- ✓ Temps d'exécution des instructions normales : 4*Tosc.
- ✓ Temps d'exécution des instructions de saut : 8*Tosc.
- ✓ Cause d'interruption : 4.
- ✓ Fréquence max de travail : 10 MHz.
- ✓ Lignes E/S numérique : 13.
- ✓ Temporisateur : un pour l'utilisateur, un pour le Watchdog.
- ✓ Tension d'alimentation : 2 à 6 V continu.
- ✓ Tension de programmation : 12 à 14 V continu.
- ✓ Boîtier : DIL 18 broches.
- ✓ Selon la version de PIC utilisée, le nombre de registres internes au circuit est différent.
- ✓ Ainsi, les registres présentés ci-après sont les plus couramment utilisés:
  - ✓ Registres d'E/S: PORT.
  - ✓ Registre d'état: STATUS.
  - ✓ Registres de direction: TRIS.
  - ✓ Registre Compteur Programme: PC.
  - ✓ Registre de travail: W.

2.7. Le Microcontrôleur PIC 16F877

Le microcontrôleur PIC 16F877 est une version améliorée du PIC 16F84. Il est disponible en 40/44 broches, selon le type de boitier :

- ✓ 40-PIN DIP.
- ✓ 44-PIN TQFN (PLCC).
- ✓ 44-PIN QFN (QFP).



Son brochage est donné par le schéma ci-dessous.

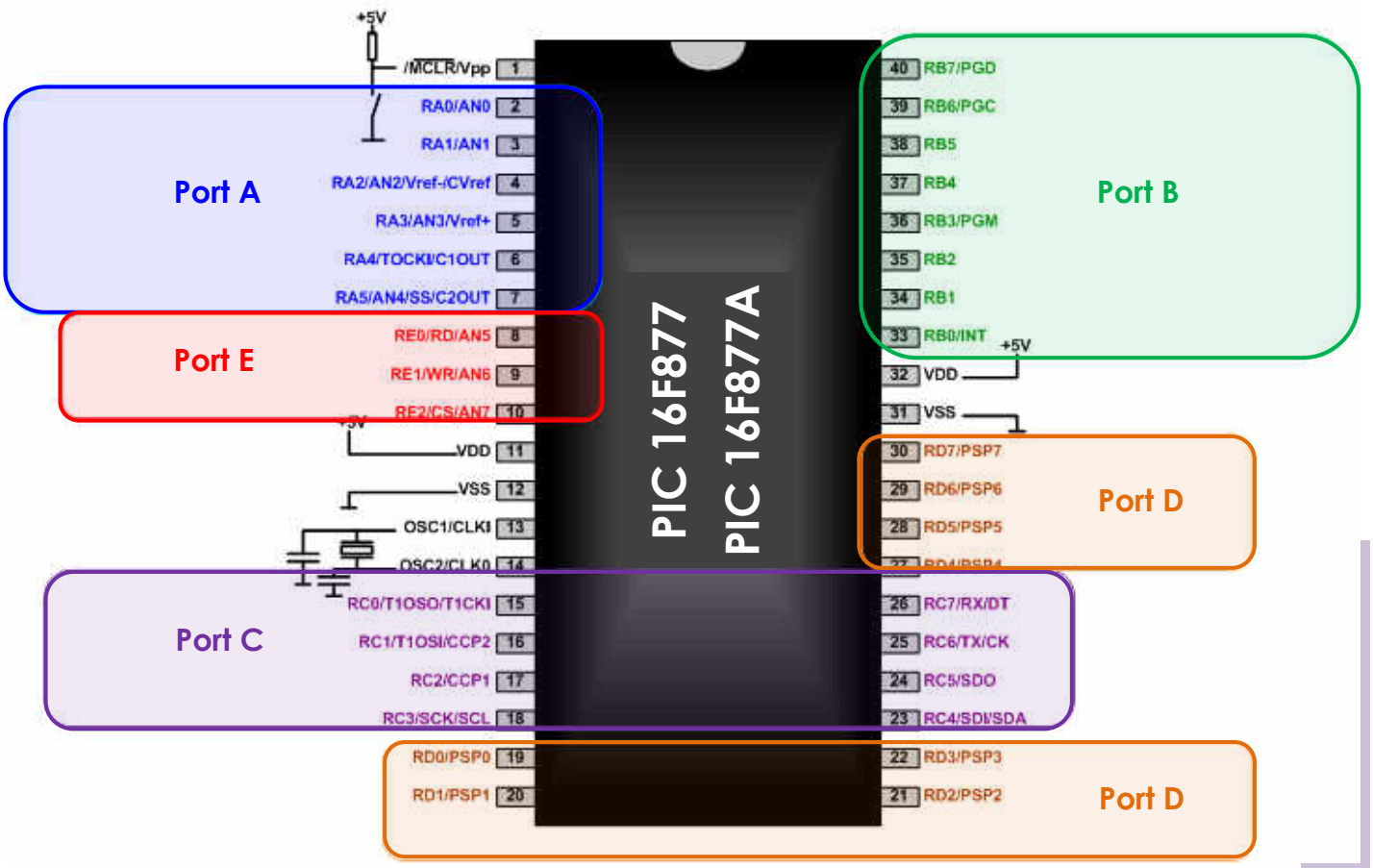
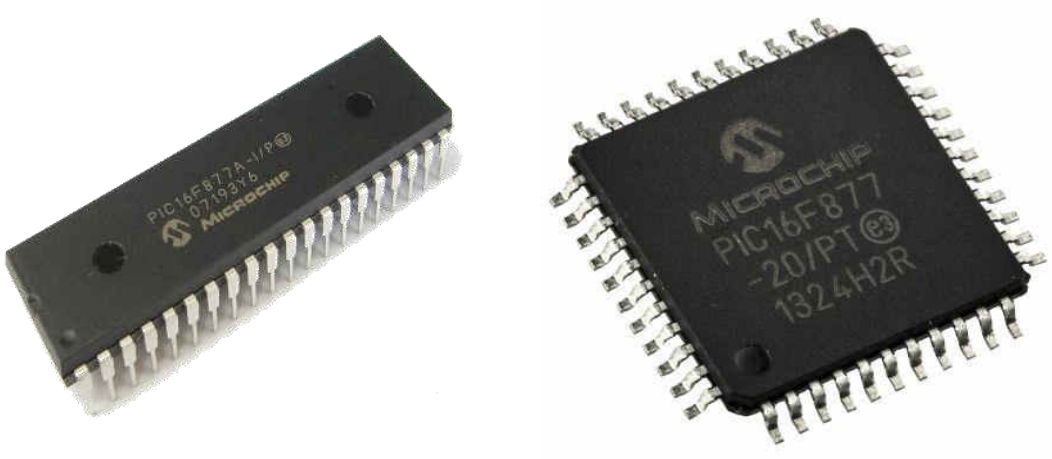


Figure 2.8 : *Brochage du PIC 16F877.*



40-PIN DIP. 44-PIN TQFN (PLCC).

Figure 2.8 : *Différents boitiers du PIC 16F877.*

Les éléments essentiels du PIC 16F877, sont :

- ✓ Une mémoire programme de type EEPROM flash de 8K mots de 14 bits.
- ✓ Une RAM donnée de 368 octets. Elle est répartie de la manière suivante :

- 80 octets en Banque 0, adresses 20H à 6FH.
- 80 octets en Banque 1, adresses A0H à 0*EFH.
- 96 octets en Banque 2, adresses 110H à 16FH.
- 96 octets en Banque 3, adresses 190H à 1EFH.

- ✓ Une mémoire EEPROM de données de 256 octets.
- ✓ Trois TIMERS avec leurs Prescalers : TMR0, TMR1 et TMR2.
- ✓ Un chien de garde (WDT).
- ✓ 13 sources d'interruption.
- ✓ Générateur d'horloge, à quartz (jusqu' à 20 MHz) ou à Oscillateur RC.
- ✓ Protection de code.
- ✓ Fonctionnement en mode SLEEP (mode de consommation réduite).
- ✓ Possibilité aux applications utilisateur d'accéder à la mémoire programme.
- ✓ Tension de fonctionnement de 2 à 5V.
- ✓ Jeux de 35 instructions.
- ✓ Programmation par mode ICSP (In Circuit Serial Programming) 12V ou 5V.
- ✓ Deux modules de comparaison et Capture CCP1 et CCP2 (Capture, Compare et PWM).
  - Module capture 16 bits avec une résolution max. 12,5 ns.
  - Module Compare 16 bits avec une résolution max. 200 ns.
  - Module PWM avec une résolution max. 10 bits.
- ✓ Cinq ports d'entrée sortie, A (6 bits), B (8 bits), C (8 bits), D (8 bits) et E (3 bits).
- ✓ Convertisseur Analogiques Numériques (CAN) 10 bits à 5 canaux.
- ✓ USART, Port série universel, mode asynchrone (RS232) et mode synchrone.
- ✓ SSP, Port série synchrone supportant I2C.

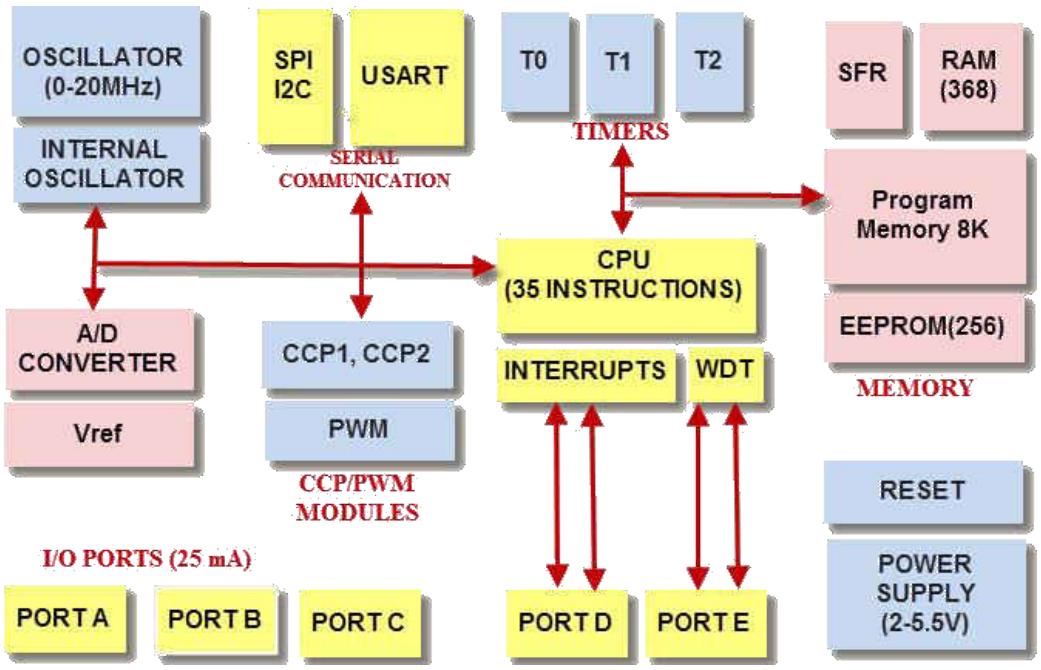
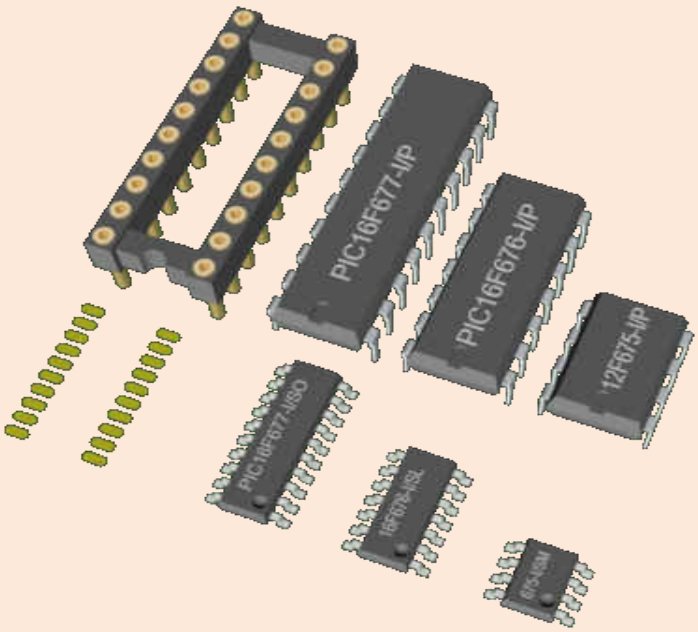


Figure 2.8 : *Architecture interne du PIC 16F877.*

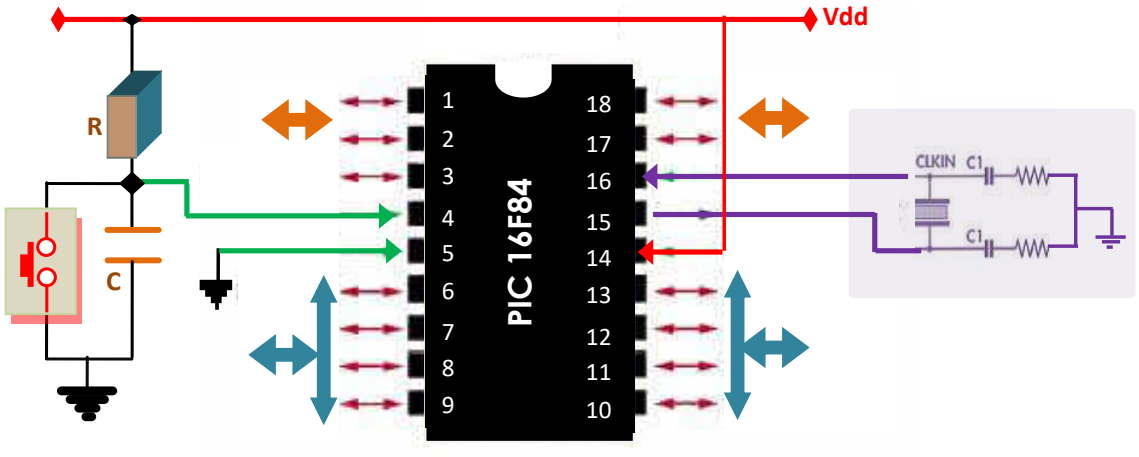
Info

On trouve dans le marché des PICs une grande série avec différents de boitiers et de brochage. La figure ci-dessous illustre quelques types de PIC avec différents nombres de broches.



EXERCICE 3.1

1. Expliquer brièvement la différence entre l'architecture VAN NEUMANN et l'architecture HARVARD.
2. Quelles sont les caractéristiques d'une architecture RISC ?
3. Quelles sont les caractéristiques d'une architecture CISC ?
4. Expliquer brièvement la différence entre un microcontrôleur et un microprocesseur.
5. C'est quoi un PIC ?
6. Citer les familles dont les PIC sont subdivisés ?
7. Dans quels types d'applications l'usage d'un microcontrôleur est plus efficace que l'usage d'un microprocesseur ? Expliquer et donner des exemples.
8. Identifier les désignations suivantes :
   - ✓ 16 F 84 -04-
   - ✓ 12 C 508
   - ✓ 16 C 72 A
   - ✓ 16 F 628
   - ✓ 18 FL 442
9. Pour chacun des pics cités précédemment donner la taille en bit de chaque instruction ainsi que sa famille.
10. Un microcontrôleur PIC est identifié par 16F876-10. Sur combien de bit ses instructions sont-elle codées? Quel est le type de sa mémoire de code ? Quelle est la fréquence maximale de son oscillateur ?
11. Donner les noms des broches utilisées dans le circuit de la figure suivant :



- ✓ Comment réaliser une horloge pour le PIC16F84 ? Discuter…
- ✓ A quoi sert le circuit RC ?
- ✓ Donner les différents types de réalisations du circuit RESET. Discuter…
- ✓ Combien de broches d'entrée/sortie possède-t-il ?
- ✓ Quelle est la capacité mémoire du composant PIC16F84 (donner la nature et la taille des différentes zones de mémoire)
- ✓ Sur quelle plage de tension peut-on alimenter ce composant ?
- ✓ Quelle est la fréquence maximale de l'oscillateur ?
- ✓ Combien de fois peut-on écrire dans la mémoire FLASH et dans l'EEPROM ?

EXERCICE 3.2

Expliquer le rôle de chaque élément de la liste ci-dessous pour le cas d'un PIC 16F84 ? Discuter…

- ✓ Mémoire programme ;
- ✓ Registre compteur de programme ;
- ✓ Port A et Port B d'entrées – sorties ;
- ✓ Unité Arithmétique et logique ;
- ✓ RAM ;
- ✓ E²PROM ;
- ✓ Horloge système ;
- ✓ Registre de décodage des instructions ;
- ✓ Registre d'état ;
- ✓ Registre de travail ;
- ✓ Registre d'instruction ;
- ✓ Timer ;
- ✓ Pointeur de pile ;
- ✓ Bus internes ;

- ✓ Reset ;
- ✓ Watch dog ;
- ✓ Alimentation

EXERCICE 3.3

Citer les mémoires dont un PIC 16F84 est constitué.

- ✓ A quoi sert une mémoire programme pour un PIC 16F84 ?
- ✓ Quelle est la taille de la mémoire programme ?
- ✓ Quelle le type de la mémoire de programme du 16F84 ?
- ✓ Sur combien de bit une instruction peut être codée ?
- ✓ Schématiser le vecteur mémoire d'adresse 3FA.
- ✓ Schématiser la mémoire programme du 16F84.
- ✓ Donner l'adresse du premier vecteur de la mémoire programme.
- ✓ A quoi correspond le vecteur d'adresse 000 au niveau de la mémoire programme ?
- ✓ A quoi correspond le vecteur d'adresse 004 au niveau de la mémoire programme ?
- ✓ Combien d'instruction qu'on peut écrire dans une mémoire programme si on commence par le vecteur d'adresse 005.

EXERCICE 3.4

- ✓ Quelles sont les mémoires qu'on trouve dans une mémoire de données ?
- ✓ Quelle est la taille de la mémoire EEPROM des donnés ?
- ✓ A quoi sert la mémoire de données EEPROM ?
- ✓ Sur combien de bits on peut coder une adresse mémoire de donnée EEPROM ?
- ✓ Dessiner une adresse mémoire de données ?
- ✓ Comment peut-on lire une donnée de la mémoire EEPROM de données ?
- ✓ Comment peut-on écrire une donnée sur la mémoire EEPROM de données ?

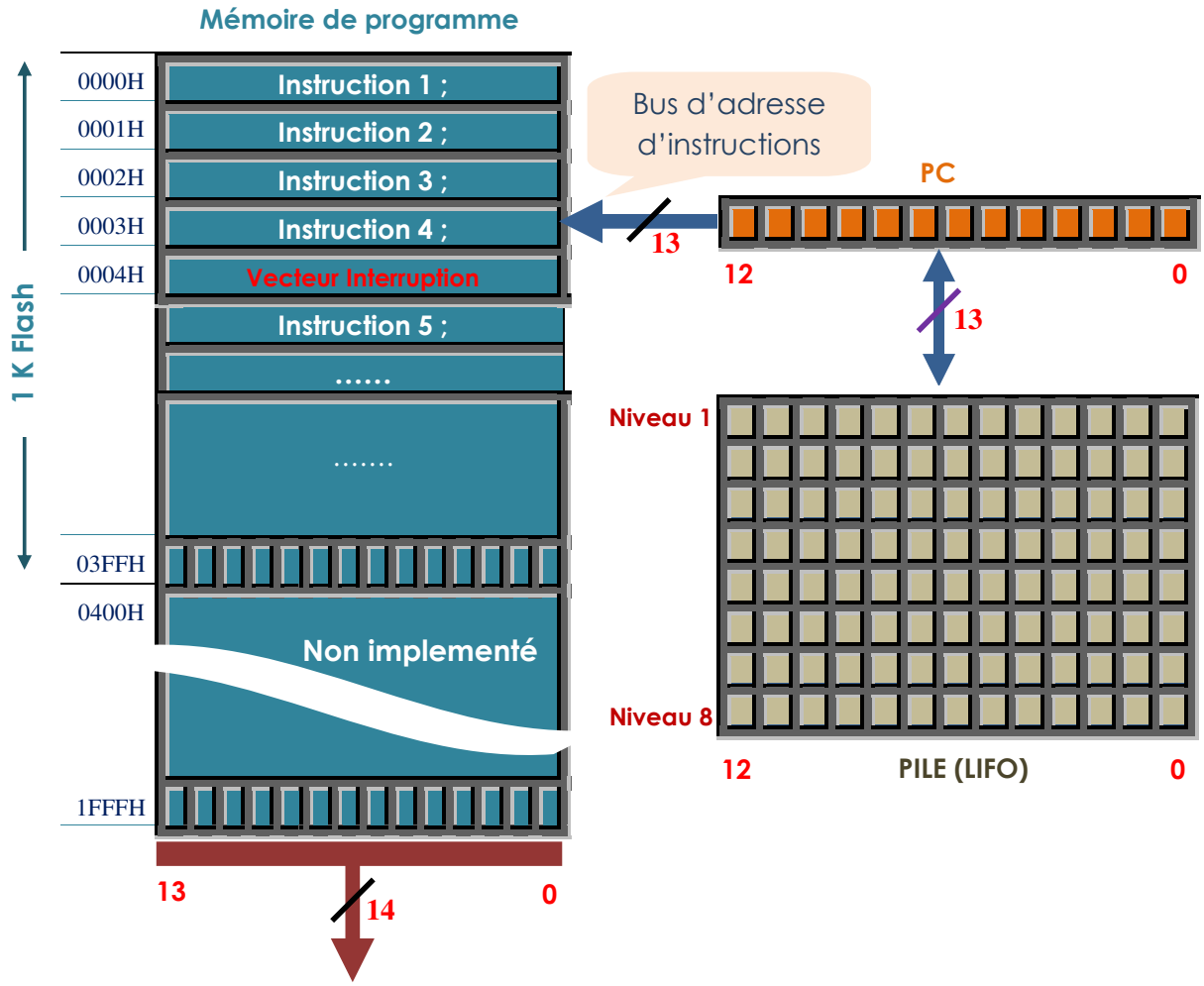
EXERCICE 3.5

- ✓ Donner la taille de la mémoire de données RAM
- ✓ Sur combien de banques est subdivisée la mémoire de données RAM ?
- ✓ Comment peut-on accéder à chaque banque ?
- ✓ Donner le nombre de bits de chaque registre de cette mémoire de données RAM.
- ✓ Donner le nom de deux portions constituant chaque banque
- ✓ Dans la suite on s'intéresse aux registres SFR :
  - ⌖ Quel est l'utilité du registre FSR et du registre INDF ?
  - ⌖ Dessiner le registre OPTION.

- Quel est le rôle du bit RBPU/ ?
- Quel est le rôle du bit INTEDG ?
- Quel est le rôle du bit RTS(T0CS) ?
- Quel est le rôle du bit RTE(T0SE) ?
- Quel est le rôle du bit PSA ?
- Quel est le rôle du bit PS2 ?
- Quel est le rôle du bit PS1 ?
- Quel est le rôle du bit PS0 ?
- Quelle est l'utilité du registre TMR0 ?
- Indiquer la banque de la mémoire RAM où peut trouver les registres PCL et PCLATH.
- A quoi servent ces deux registres (PCL et PCLATH) ?
- A quoi servent les registres PORTA et PORTB ?
- Quelles sont les valeurs à donner TRISA et TRISB pour rendre PORTA et PORTB comme entrées ?
- Quelles sont les valeurs à donner TRISA et TRISB pour rendre PORTA et PORTB comme sorties ?
- Quelle est l'utilité du registre EEADR ?
- Quelle est l'utilité du registre EEDATA ?
- Quelle est l'utilité du registre EECON1 ?
- Dessiner le registre EECON1. Expliquer le rôle de chaque bit…
- Dessiner le registre INTCON. Expliquer le rôle de chaque bit…
- Quelle est l'utilité du registre STATUS? Expliquer le rôle de chaque bit…
- A quoi sert le registre de travail W ?

EXERCICE 3.6

Le schéma suivant illustre le principe d'exécution d'un programme écrit dans la mémoire programme d'un PIC 16F84 :



- ✓ Quel est le rôle du PC ?
- ✓ Donner le nombre maximal que peut compter le PC.
- ✓ Donner le nombre maximal d'instruction contenu dans la mémoire programme.
- ✓ Déduire le rôle de la pile.

EXERCICE 3.7

Donner la réponse correcte parmi les différentes réponses proposées. Justifier votre choix.

1. Le débordement du WDT provoque:

- ☐ L'initialisation de l'EEPROM du PIC
- ☐ Reset du PIC
- ☐ Activation d'une interruption

2. La EEPROM du PIC 16F84 est de

- ☐ 64 octets
- ☐ 68 octets
- ☐ 255 octets

3. Le PIC16F877 possède :

- ☐ 01 Timer et 02 Port d'entrée sortie

4. Si la fréquence de l'oscillateur d'un PIC est de 10 MHz, le cycle instruction sera de :

- ☐ 1µs

- ☐ 02 Timer et 03 Port d'entrée sortie
- ☐ 03 Timer et 04 Port d'entrée sortie

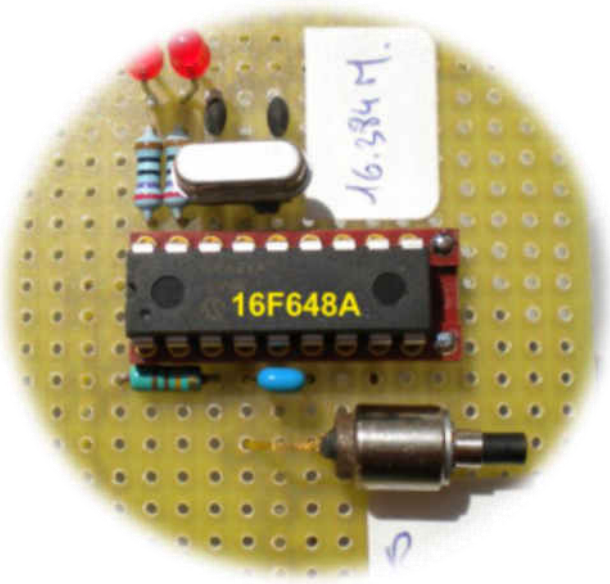
- ☐ 0.4µs
- ☐ 0.25µs

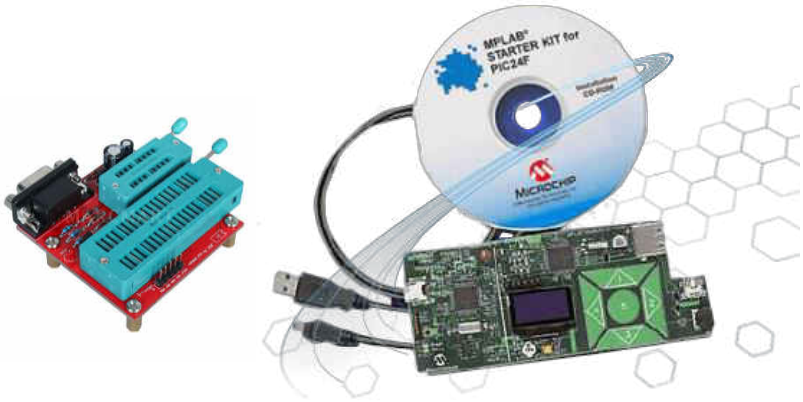
5. La valeur maximale en hex que peut contenir un registre est :

- ☐ 128
- ☐ FF
- ☐ 256h
- ☐ 255h

6. La mémoire flash du PIC 16F84 est de

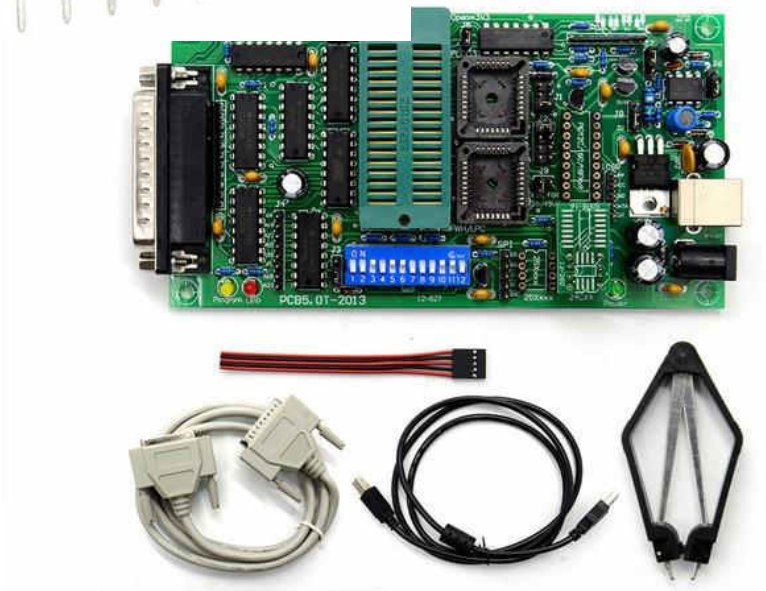
- ☐ 64 Koctets
- ☐ 1 Moctets
- ☐ 512 Koctets

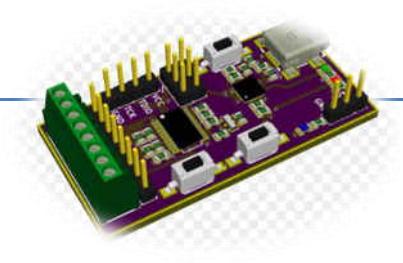




CHAPITRE **3**

Architecture logicielle du PIC 16F84





CHAPITRE 3

Architecture logicielle du PIC 16F84

Objectifs

L'objectif est de

- ✓ Présenter d'une manière profonde Les processus de développement d'un programme à exécuter sur un PIC.
- ✓ Décrire les fonctionnalités logicielles et les concepts associés au PIC 16F84.
- ✓ Etudier la programmation du PIC 16F84.

Avant Propos

Les processus de développement d'un programme à exécuter sur un PIC sont :

1. Ecrire un programme en langage assembleur du PIC dans un fichier texte et le sauvegarder avec l'extension **.asm**.
2. Compiler ce programme avec l'assembleur **MPASM** fourni par **Microchip**. Le résultat est un fichier exécutable avec l'extension **.hex** contenant une suite d'instruction compréhensible par le PIC.
3. Transplanter le fichier **.hex** dans la mémoire programme du PIC (mémoire **FLASH**) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de **Microchip** ou tout autre programmateur acheté ou réalisé par soit même.
4. Mettre le PIC dans son montage final puis mettre sous tension.

Il est à noter que **Microchip** propose gratuitement (Téléchargeable à partir du l'URL : http://www.microchip.com/) l'outil de développement MPLAB qui regroupe

- ✓ **L'éditeur** de texte,
- ✓ Le **compilateur** MPASM,
- ✓ Un **outil de simulation** et
- ✓ Le **logiciel de programmation**.

Le programmateur lui-même, n'est malheureusement pas gratuit.

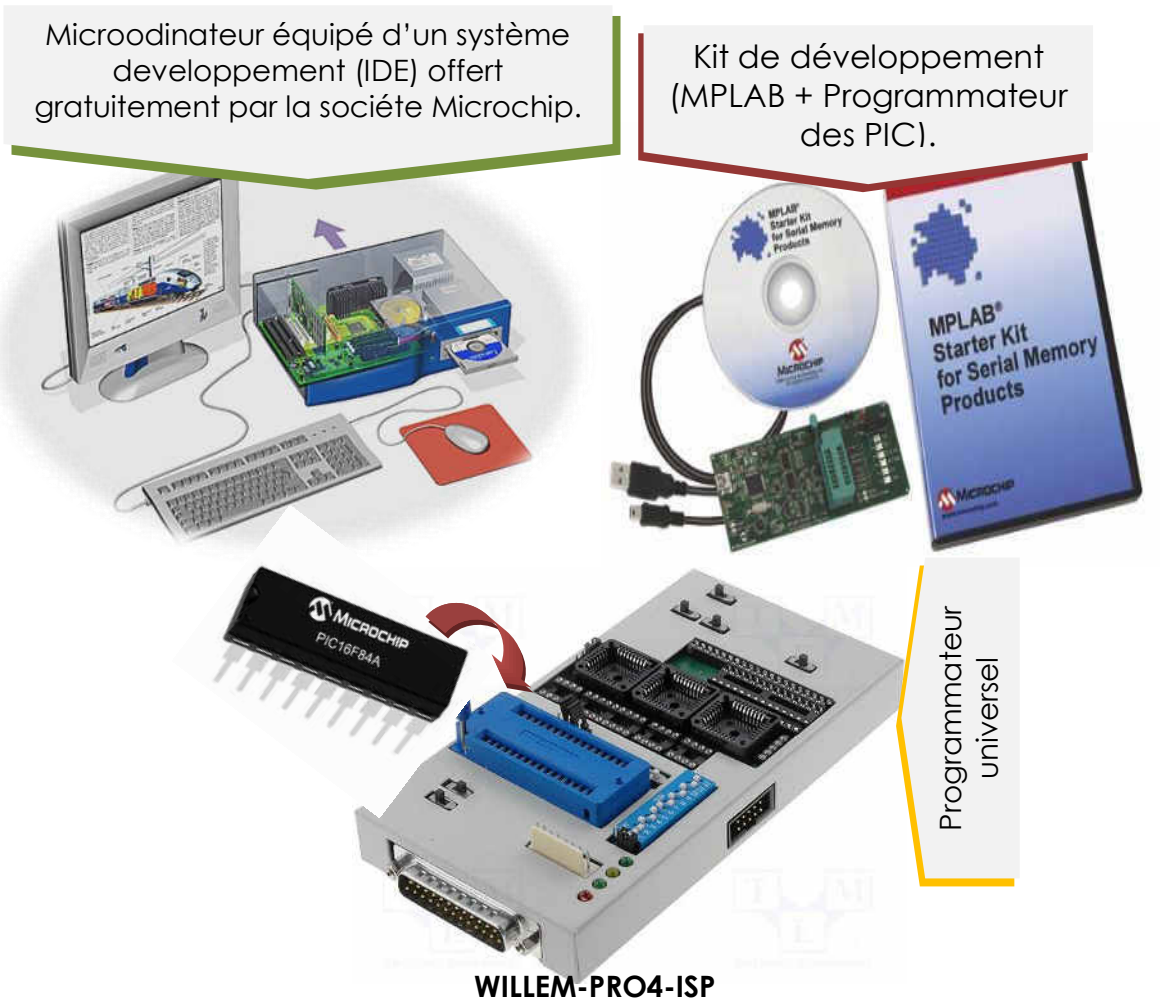


Figure 3.1 : *Système de développement des PICs.*

3.1. Structure d'un programme

Le microcontrôleur exécute le programme chargé dans sa **mémoire FLASH**. Les mots binaires sur 14 bits pour le cas de PIC 16F84a sont considérés par le CPU comme le système (commande) de gestion de la carte développée.

Pour obtenir le fichier de 'commande' binaire (ou même Héxadécimal), il est necessaire (voir obligatoire) d'utiliser des langages de programmation (**compilateurs**) qui utilisent des abréviations (code plus lisible pour le programmeur), à savoir

- ✓ Langage bas niveau comme l'assembleur ou
- ✓ Des langages évoluées comme le langage C.

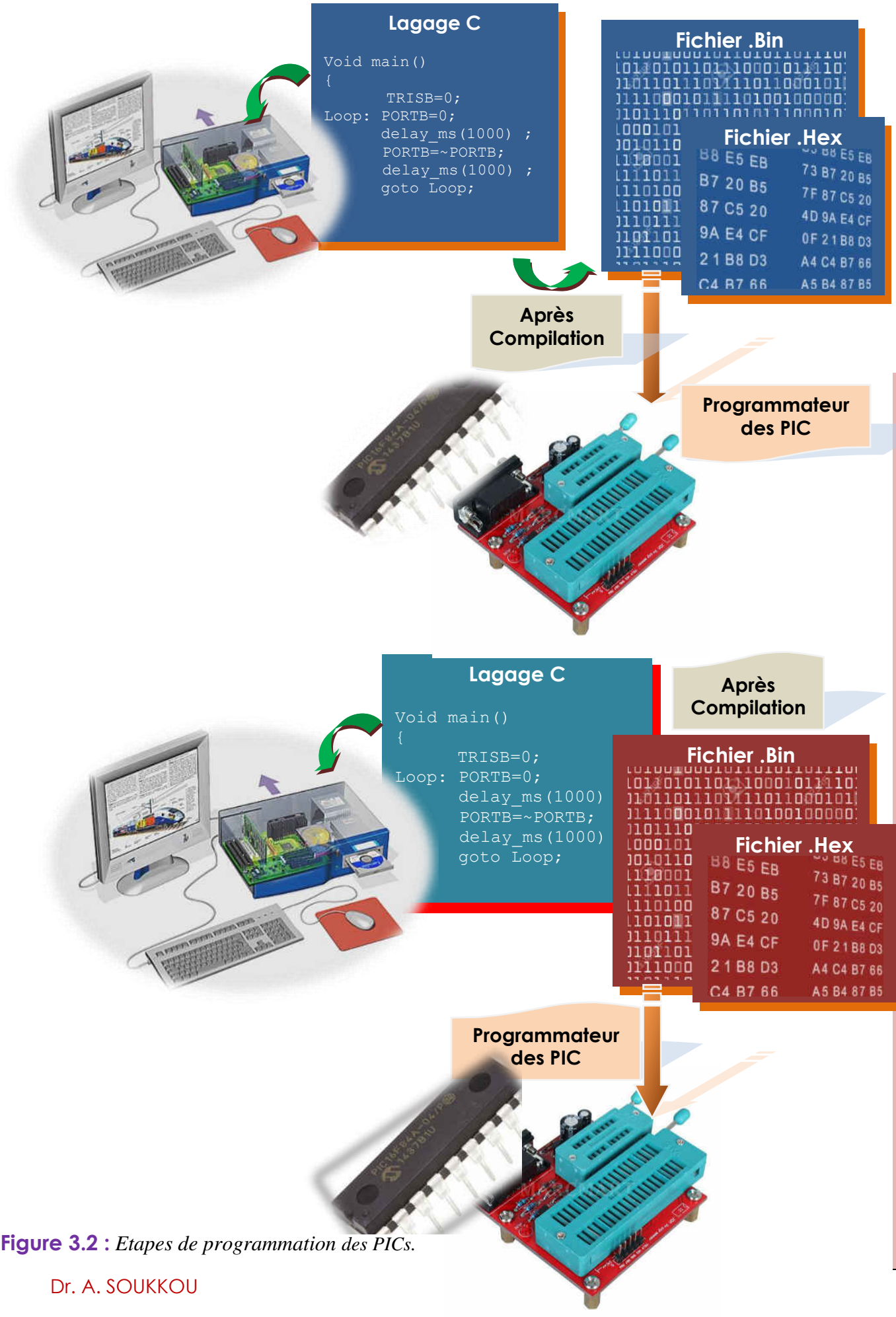
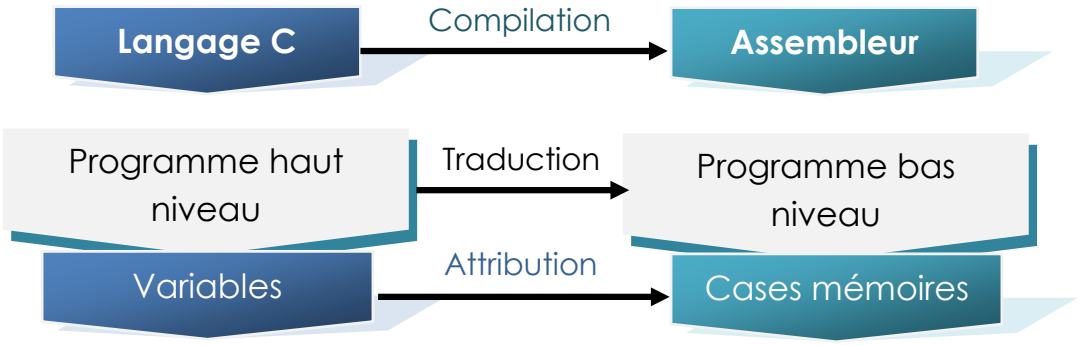


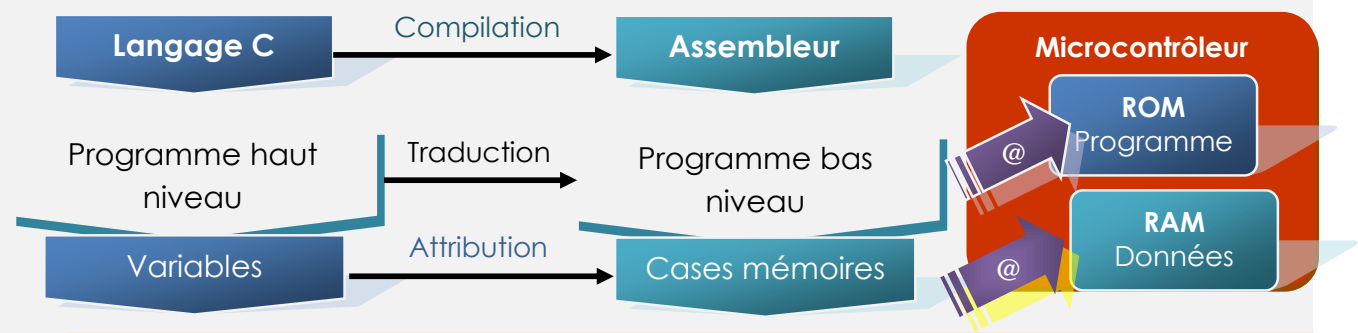
Figure 3.2 : *Etapes de programmation des PICs.*

Info

- Il est à noter que les microcontrôleurs ne travail qu'avec des **langages de bas niveau (Assembleur)**. Cependant, ces langages étant difficiles à comprendre et à appréhender pour les êtres humains, c'est la raison pour laquelle les fabricants des microcontrôleurs prévoient de les programmer à l'aide des **langages de plus haut niveau** (C, C++, Python, etc.)
- Chaque type des microcontrôleurs et microprocesseurs possède son propre assembleur ou son propre jeu d'instruction.
- Une fois le code écrit dans le langage associé au type de microcontroleur, il existe des programmes qui permettent de transformer des **langages de haut niveau** (Python, C, C++, etc.) vers ces **langages machines**. Ce sont les **compilateurs**.
- Dans le commerce, il ya plusieurs variétés de compilateurs des différents fabricants et avec différents langages de haut niveau.
- La compilation peut être réalisée en deux étapes différentes :
  - ✓ **Traduction du code** du langage évolué (C comme exemple) vers un langage de bas niveau qu'est l'assembleur du PIC utilisé.
  - ✓ **Attribution** des lignes de code et des variables **à des espaces mémoires** (ROM pour le programme, RAM pour les données ou variables) en fonction du type de microcontroleur utilisée.



- Dans le cas général, ces compilateurs sont intégrés dans un **environnement de développement** propre au type de microcontrôleurs (**IDE**) et ils sont associés à une série de bibliothèques de fonctions permettant de gérer les traitements internes ou/et de communiquer avec des périphériques d'Entrées/Sorties.
- La compilation permet d'associer, dans un seul (et même) fichier, **l'ensemble du code écrit et des fonctions utilisées** pour l'application développée (provenant des diverses bibliothèques utilisées). On obtient alors en sortie un fichier binaire pouvant par la suite être transférer sur la cible à travers un programmateur des PIC.



Une fois le transfert du programme est effectué par le biais du programmateur du PIC, le microcontroleur doit être placé dans la carte d'application et l'ensemble sera opérationnel sans l'intervention du programmateur ou de l'ordinateur auquel il est connecté.

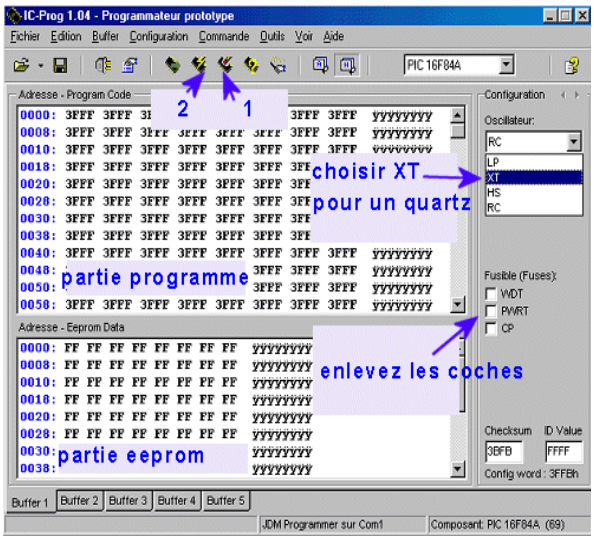
Cas Particulier

Le logiciel MPLAB disponible sur le site **www.microchip.com** permet de saisir via un éditeur de texte la source d'un programme écrit en assembleur du PIC, puis de compiler celui-ci afin de le charger (transférer) dans la mémoire programme du PIC concerné (Flash).

- ✓ Une fois le fichier source compilé il suffit de le transférer dans la mémoire programme du PIC.
- ✓ Pour effectuer le transfert, il faut un programmateur des PIC.
- ✓ De nombreux montages de programmateur de PIC ont été publiés et commercialisés.
- ✓ Simplification du tracé du circuit imprimé.
- ✓ Fiabilité ''assuré'' du système.
- ✓ Consommation réduite de l'énergie.
- ✓ Coûts réduit de main d'œuvre (câblage) : Conception et montage.
- ✓ Intégration dans un seul boîtier.

- ✓ Un logiciel est également nécessaire pour le transfert du fichier compilé vers le PIC, le plus populaire est **ICPROG**, disponible sur le site :

www.ic-prog.com.



3.2. Programmer avec le PIC 16F84

Certains registres sont indispensables au déroulement des programmes écrits en assembleur du PIC 16F84.

1. Le registre de travail **W** (Work), associé directement à l'UAL indépendant des registres SFR où le processeur ne peut déplacer une donnée ou effectuer une opération arithmétique ou logique sur un registre (SFR ou GPR) qu'en utilisant le registre de travail W.

2. Le registre d'état **STATUS** : Son rôle est de donner des indications liées à l'exécution de la dernière instruction par le CPU (Etat du processeur). Il permet aussi de sélectionner les différentes pages mémoires en RAM ("banques").
3. Le compteur programme **PC**: Il pointe sur la prochaine instruction à exécuter. Il fait partie de SFR :

   - ✓ **PCL_8 (PC Low).**
   - ✓ **$PCLATH_5$ (PC LATch counter High).**
   - ✓ **$PC_{13} = PCL_8 + PCLATH_5$.**

     - Le premier octet est lu a partir de du registre PCL.
     - Le registre PCL peut être consulté pour lecture et pour écriture.
     - Les 5 derniers bits du PC (bit12 - bit 8) ne sont pas accessibles ni pour lecture ni pour écriture et ils sont lus a partir du registre.
4. Lors de la mise sous tension, tous les registres spécifiques (**SFR**) sont placés dans un état déterminé comme montre la figure ci-dessous.

Tableau 3.1 : *Etats des SFR lors d'un RESET dans le PIC 16F84.*

Registre	Adresse	Valeurs après RESET
INDF	00H - 80H	---- ----
TMR0	01H	xxxx xxxx
PCL	02H – 82H	0000 0000
PCLATH	0AH – 8AH	---0 0000
STATUS	03H – 83H	0001 1xxx
FSR	04H – 84H	xxxx xxxx
PORTA	05H	---x xxxx
PORTB	06H	xxxx xxxx
EEDATA	08H	xxxx xxxx
EEADR	09H	xxxx xxxx
INTCON	0BH - 8BH	0000 000x
OPTION	81H	1111 1111
TRISA	85H	---1 1111
TRISB	86H	1111 1111
EECON1	88H	---0 x000
EECON2	89H	---- ----

*. **–** Non implémenté.
*. **x** = '0' ou '1'.

3.1.1. Les instructions du PIC 16F84

Tous les PICs Mid-Range ont un jeu de 35 instructions. Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (**COP**) ainsi que l'opérande (**@OP**).

- ✓ **COP** : Permet de spécifier le type de l'opération à effectuer.
- ✓ **@OP** : Permet de spécifier le ou les arguments de l'opération (un ou plusieurs opérande).

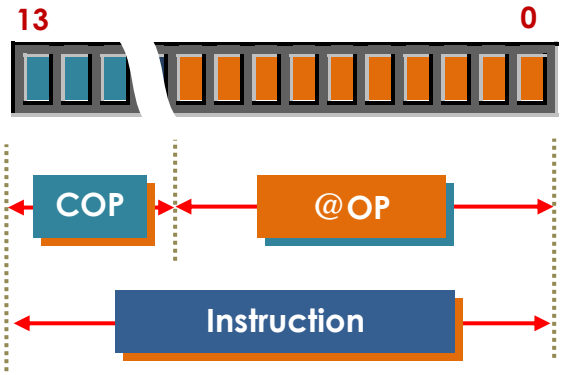


Figure 3.4 : *Format d'instruction du PIC 16F84.*

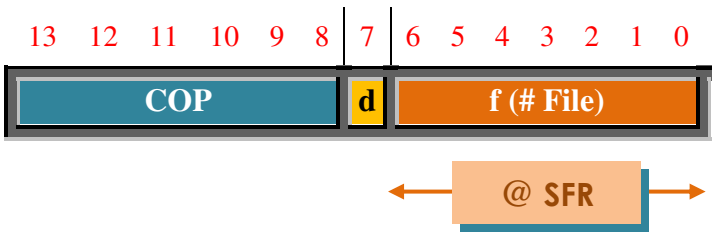
Par la suite, on utilise la nomenclature de la table ci-dessous.

Tableau 2.5 : *Nomenclature utilisée.*

Nomenclature	Description
f	✓ Registre SFR (adresse du File Registre)
d	✓ Spécifier l'emplacement du résultat (Destination de l'opération) : ✓ **W** si **d = 0** ou ✓ **f** si **d = 1**
K	✓ Valeur immédiate (Literal value).
b	✓ Position du bit dans l'octet : Permet de sélectionner le nombre de bits affectés par l'opération.
OD	✓ Opérande déstination.
OS	✓ Opérande source.

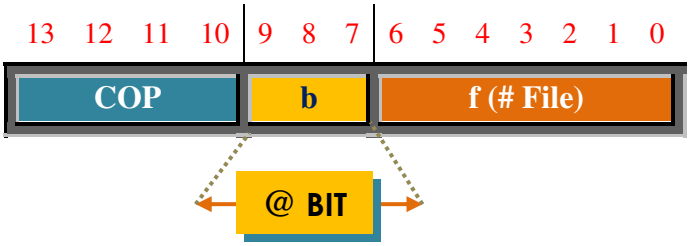
Pour simplifier, les 35 instructions peuvent être classées selon quatre critères qui sont :

1. **Opérations orientés octets**



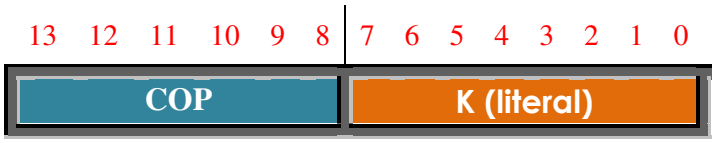
* Si **d = 0**, le résultat est placé dans le registre de travail **W**;
* Si **d = 1**, le résultat est placé dans le registre **SFR** spécifié dans l'instruction ;
* @SR : Adresse du registre concerné ;

2. Opérations orientés bits



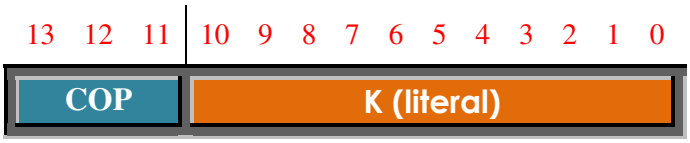
* **b (3 bits)** : représente la position du bit à affecter dans le registre f ;
* **f** : représente le numéro du registre dans lequel le bit est localisé ;
* **@BIT** : Bit concerné ;
* **f (#FILE)** : Adresse du registre concerné ;

3. Opérations littérale:



* **K (8 bits)** : Valeur immédiate (constante);

4. Opérations de contrôle :



* **K (11 bits)** : Valeur immédiate (Adresse de branchement);
* Cette format est utilidée dans le de branchement (saut) (**GO TO**) et d'appels aux sous programmes (sous-routines) (**CALL**).

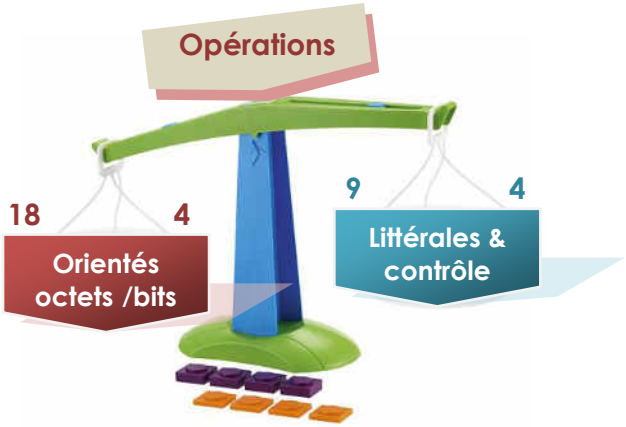


Figure 3.4 : *Classement des 35 instructions selon le nombre des bits (**COP, @OP**).*

Les instructions pourront avoir plusieurs écritures telles que :

Tableau 3.5 : *Format des instructions pour le PIC 16F84.*

Type	Exemple
INST ;	**RETURN** ; - Instruction -
INST Val ;	**MOVLW 05** ; - Instruction + valeur -
INST OD, OS ;	**ADDWF reg1, W** ; - Instruction + source, destination -
INST OS, K ;	**BSF reg1, 3** ; - Instruction + source, numéro bit du registre -

Microchip groupe les instructions des PIC mide-rang sous les rubriques mentionnées ci-dessous.

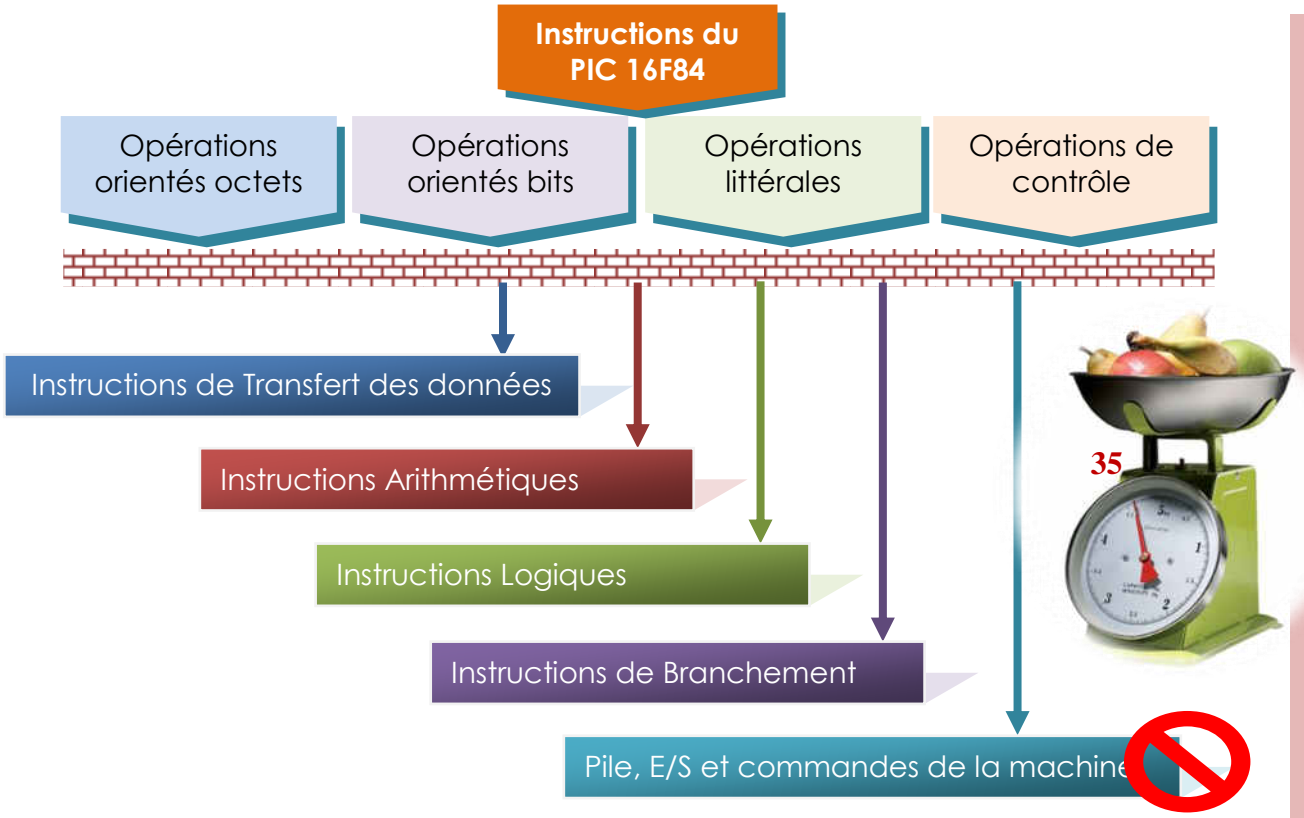


Figure 3.4 : *Groupes d'instructions du PIC 16F84.*

Info

- Dans la mémoire programme, les instructions sont codées en suites binaires de '0' et de '1'. Le compilateur se charge de traduire de code mnémonique en code binaire.
- Chaque ligne du programme assembleur peut contenir jusqu'à 4 types d'informations :
  - ✓ Une étiquette (adresse de l'instruction en mémoire).
  - ✓ L'instruction en code mnémonique.
  - ✓ Le ou les opérandes.

✓ Des commentaires.

Etiquette :	Code Opération (**COP**)	Opérande (s) (@**OP**)	Commentaires ;

- Une ligne contient 255 caractères maximum.
- A part les instructions de branchement (saut), **toutes les instructions** sont exécutées en **un cycle d'horloge**.
- Si on utilise par exemple un quartz de 4MHz (Sachant que l'horloge fournie au PIC est prédivisée par 4), on obtient donc

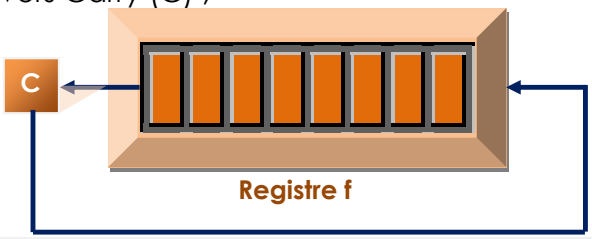
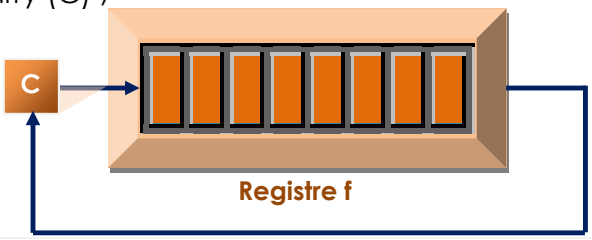
  1 000 000 cycles/seconde = 1MIPS (1 Million d' Instructions Par Seconde).
- Avec une horloge de 20MHz, on obtient une vitesse de traitement plus importante.

Le jeu d'instructions du PIC 16F84 est présenté dans les tableauxci-dessous. Il contient au total :

- ✓ 9 opérations sur des octets.
- ✓ 10 opérations sur des bits.
- ✓ 11opérations de contrôle et opérations sur les littéraux.

Tableau 3.5 : *Jeu d'instruction du PIC 16F84.*

Code mnémonique	Description	Bits Affectés	Cycle Inst
Instructions de Transfert des données			
movf f, d ;	Si d=0, Copie le contenu de f dans W, sinon copie f dans f ;	**Z**	**1**
movlw k ;	Chargement (littéral) d'une valeur k dans W ;	**Z**	**1**
movwf f ;	Copier le contenu de W dans le registre f ;		**1**
clrf f ;	Placer zéro dans f ;	**Z**	**1**
clrw	Placer zéro dans W (passage du bit Z dans STATUT '1' ;	**Z**	**1**
Clrwdt ;	Initialiser (Remise à zéro) le timer du chien de garde ;	**TO PD**	**1**
swapf f,d ;	Permuter (échanger) les deux quartets de f. Résultat dans W si d=0 ou dans f si d=1 ;		**1**
Instructions arithmétiques			
addlw k ;	Ajoute une constante k à W. Résultat dans W ;	**C DC Z**	**1**
addwf f, d ;	Ajouter W à f. Résultat dans W si d=0 ou dans f si d=1 ;	**C DC Z**	**1**
btfsc f, b ;	Tester le bit b de f. Incrémente PC si b=0 (Sauter la prochaine instruction) ;		**1(2)**

btfss f, b ;	Tester le bit b de f. Incrémente PC si b=1 (Sauter la prochaine instruction) ;		**1(2)**
incf f, d ;	Incrémente f ;	**Z**	**1**
incfsz f, d ;	Inrémenter f. Résultat dans W si d=0 ou dans f si d=1 ;		**1(2)**
sublw k ;	Sustrait W d'une constante k (W ← k-W) ;	**C DC Z**	**1**
subwf f, d ;	Soustrait W de f. Résultat dans W si d=0 ou dans f si d=1 ;	**C DC Z**	**1**
decf f, d ;	Décrémenter f. Résultat dans W si d=0 ou dans f si d=1 ;	**Z**	**1**
decfsz f, d ;	Décrémenter f. Si f=0, incrémente PC (Sauter la prochaine instruction). Résultat dans W ou dans f ;		**1(2)**
Instructions logiques			
andlw k ;	Effectuer un ET logique entre une la constante k et W. Résultat dans W ;	**Z**	**1**
andwf f, d ;	Effectuer un ET logique entre W et f. Résultat dans W si d=0 ou dans f si d=1 ;	**Z**	**1**
iorlw k ;	Effectuee un OU logique entre une constante et W ;	**Z**	**1**
iorwf f, d ;	Effectuer un OU logique entre W et f. Résultat dans W si d=0 ou dans f si d=1 ;	**Z**	**1**
comf f, d ;	Complementer f à 1. Résultat dans W si d=0 ou dans f si d=1 ;	**Z**	**1**
rlf f, d ;	Effectuer une rotation de bits à gauche à travers Carry (C) ; 	**C**	**1**
rrf f, d ;	Effectuer une rotation de bits à droite à travers Carry (C) ; 	**C**	**1**
xorlw k ;	Effectue un OU exclusif entre une constante et W. Resultat dans W ;	**Z**	**1**
xorwf f, d ;	OU exclusif entre W et f. Résultat dans W si d=0 ou dans f si d=1 ;	**Z**	**1**

bcf f, b ;	Remise à zéro le bit b de f ;		**1**
bsf f, b ;	Mise à 1 le bit de f à 1 ;		**1**
Instructions de branchement (saut) / mise en veuille			
goto Label ;	Branchement à une adresse Label (Etiquette) PC ←@SAUT = Label ;		**2**
sleep ;	Mise en mode veille le PIC ;	**TO PD**	**1**
nop	Aucune opération n'aura lieu		**1**
call Label ;	Empile PC et affecte PC de l'adresse d'un sous programme (Etiquette): PILE ← PC+1 ; PC ←@SAUT = Label (Etiquette) ;		**2**
retfie ;	Retour d'interruption ;		**2**
retlw k ;	Retour de sous programme avec chargement de valeur k dans W.		**2**
return ;	Retour de sous programme		**2**

3.1.2. Les modes d'adressage du PIC 16F84

Les modes d'dressage permettent de **localiser précisément** les opérandes d'une instruction. En d'autre terme c'est le cheminement des informations (données).

Certaines architectures offrent les instructions disponibles avec différents modes d'adressage, d'autres possèdent des modes d'adressage spécifique à chaque instruction. Le PIC 16F84 utilise 3 modes d'adressage, à savoir :

1. Adressage immédiat (Littéral).
2. Adressage Direct.
3. Adressage indirect.

Tableau 3.5 : *Modes d'adressage du PIC 16F84.*

Mode	Instruction	Exemple	Description
Adressage immédiat (Littéral)	**INST Val ;**	**movlw 0x50 ;** (W ← 0x50)	Les données à traiter sont disponibles dans l'instruction elle-même.
Adressage direct	**INST f, adr ;** **INST adr, f ;**	**movf 0x10, w ;** W← le contenu de l'emplacement mémoire spécifié par l'adresse 10H.	L'adresse des données est spécifiée dans l'instruction elle-même. La donnée est contenue dans un registre. Ce dernier peut être par un nom (par Exemple W) ou une adresse mémoire.

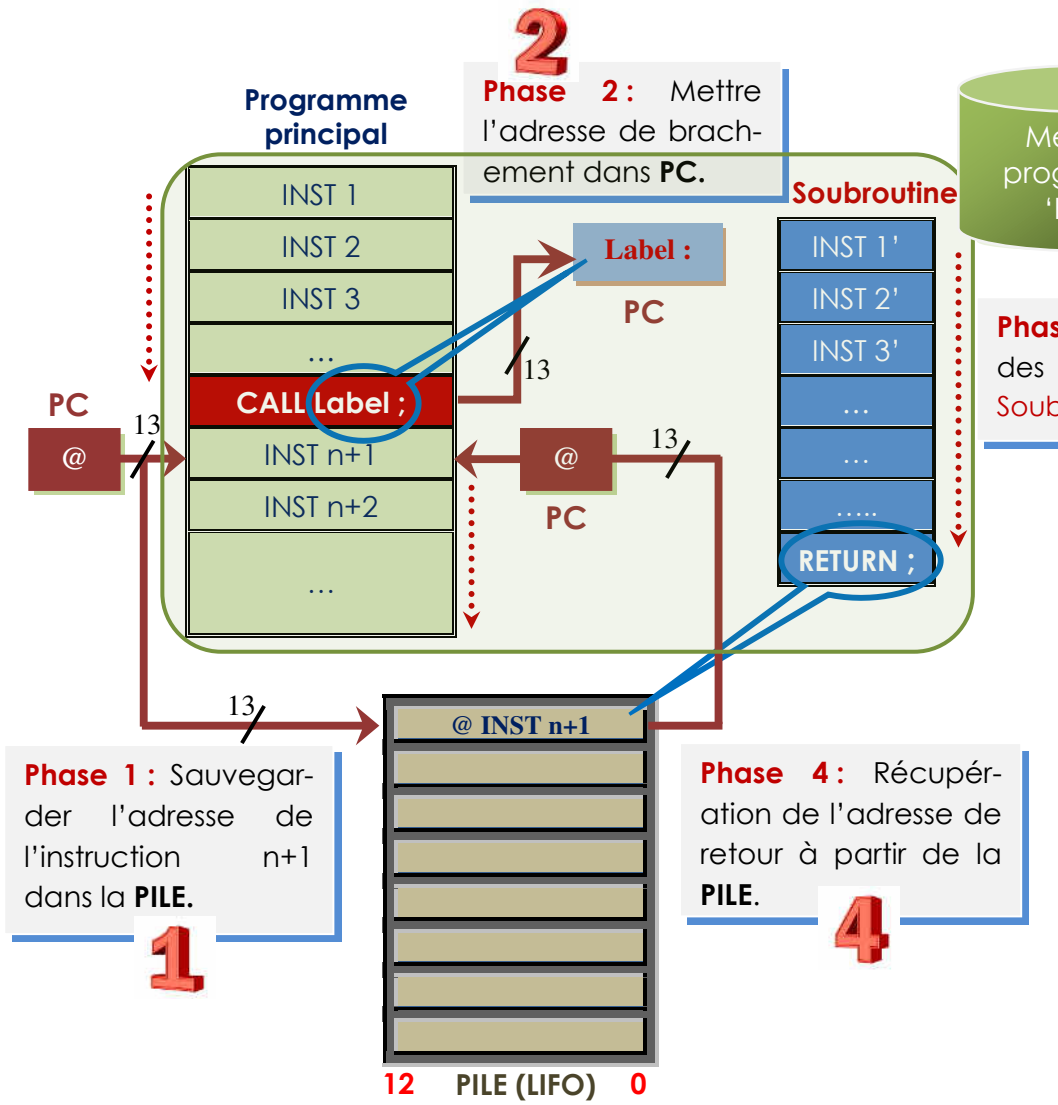
<table>
<tr><td></td><td></td><td>movf 0x2B, 0 ;
Transfert dans W la valeur contenue à l'adresse 2BH.
L'adresse 2BH peut correspondre à 2 registres en fonction de la banque choisie (0 ou 1).
Le bit RP0 permet ce choix, le bit RP1 étant réservé pour les futurs systèmes à 4 banques.</td></tr>
<tr><td>Adressage indirect</td><td colspan="2">L'adressage indirect se fait par l'intermédiaire des registres FSR et INDF. Le registre INDF n'est pas un vrai registre mais représente la case mémoire pointée par le registre d'index FSR.
Pour lire ou écrire dans une case mémoire en utilisant l'adressage indirect, on commence par :
✓ Placer l'adresse dans le registre FSR,
✓ Ensuite on lit/écrit dans le registre INDF.

movlw 0x50 ; W ← 0x50
movwf mavariable ; mavariable ← 0x50
movlw mavariable ; W ← 0x0E
movwf FSR ; On place l'adresse de destination dans FSR.
movf INDF, w ; w ← 0x50
L'adressage indirect permet d'acceder à plusieurs données sans spécifier leur adresse effective. L'interet est de permettre l'acces à des variables consecutives avec un minimum d'instructions.</td></tr>
</table>

* **FSR** est un registre spécial situé à l'adresse 04H (en banque 0) de la mémoire des données (Data RAM). Il s'agit d'un pointeur. Ce registre est également accessible en banque 1 (à l'adresse 84H).

Le rôle de la pile est de mémoriser l'adresse de retour d'une soubroutine. Donc, elle intervient seulement dans le mécanisme interne des instructions suivantes :

- ✓ **call ;**
- ✓ **return ;**
- ✓ **retlw ;**
- ✓ **retfie ;**

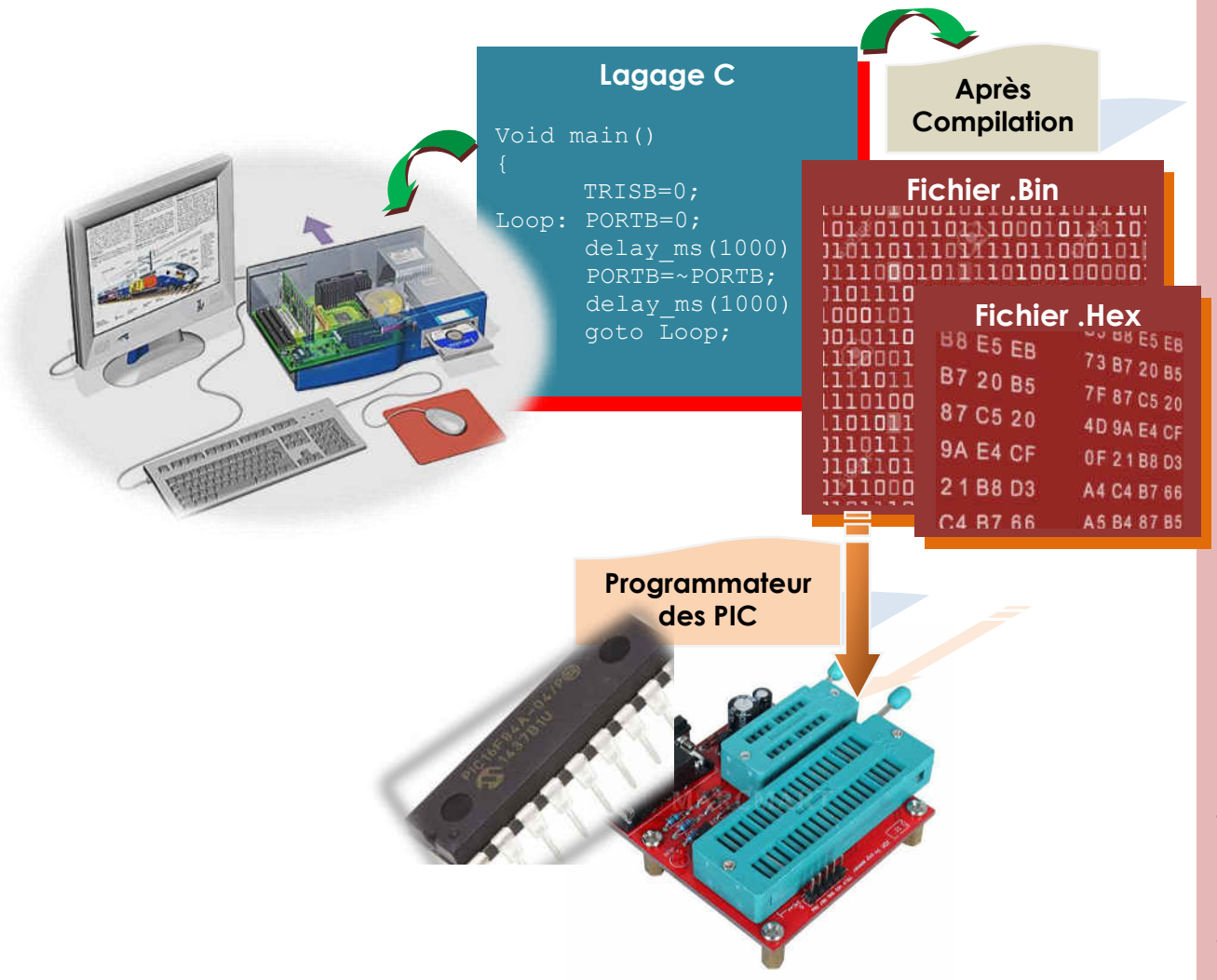


- Dans le cas d'une interruption ou de l'exécution de l'instruction **call**, l'adresse de l'instruction suivante (qui se trouve dans **PC**) est sauvegardée dans la pile.
- Quand une instruction **return**, **retlw** ou **retfie** apparaît, le PIC lit la dernière valeur sauvegardée dans la **PILE**, libère l'emplacement et se branche à l'adresse indiquée.
- Dans un programme, un sous-programme peut appeler un sous-programme (ou plutôt un sous-sous-programme). Donc, la pile réserve un autre emplacement.
- Pour le PIC 16F84, la taille de la PILE est limitée à **8 niveaux**. S'il y a plus de 8 sous programmes (routines) actives en même temps, le contenu de la pile est écrasé et le programme ne fonctionne plus correctement. Cela se traduit par
  - ✓ un plantage (le PIC ne fait plus rien) ou bien par
  - ✓ Un comportement plus ou moins incohérent (le PIC fait des choses bizarres).

Attention

EXERCICE 3.1

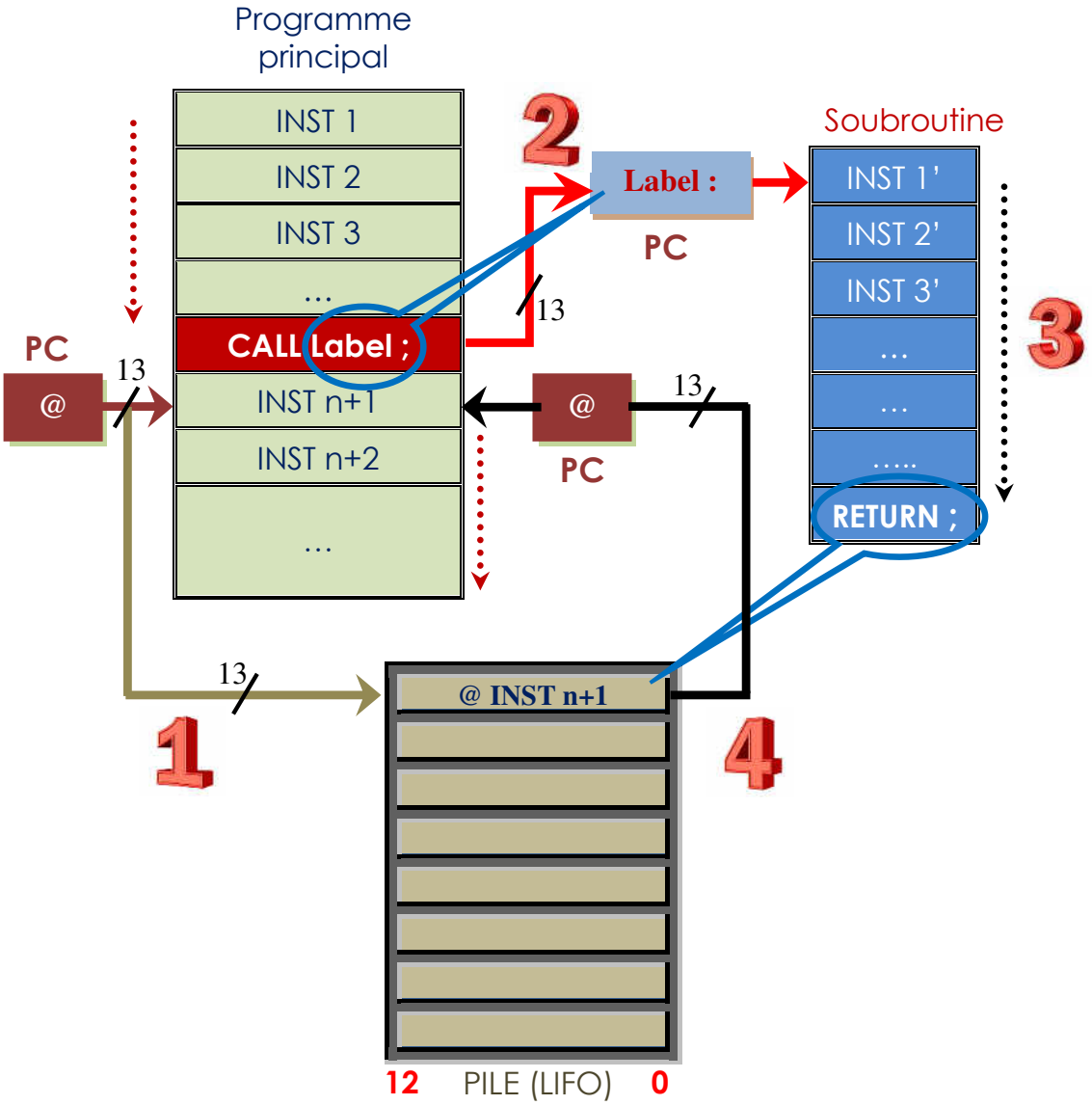
Expliquer le schéma de la figure suivante :



✓ Expliquer le processus d'exécution d'une instruction dans le cas d'un PIC16F84.

EXERCICE 3.2

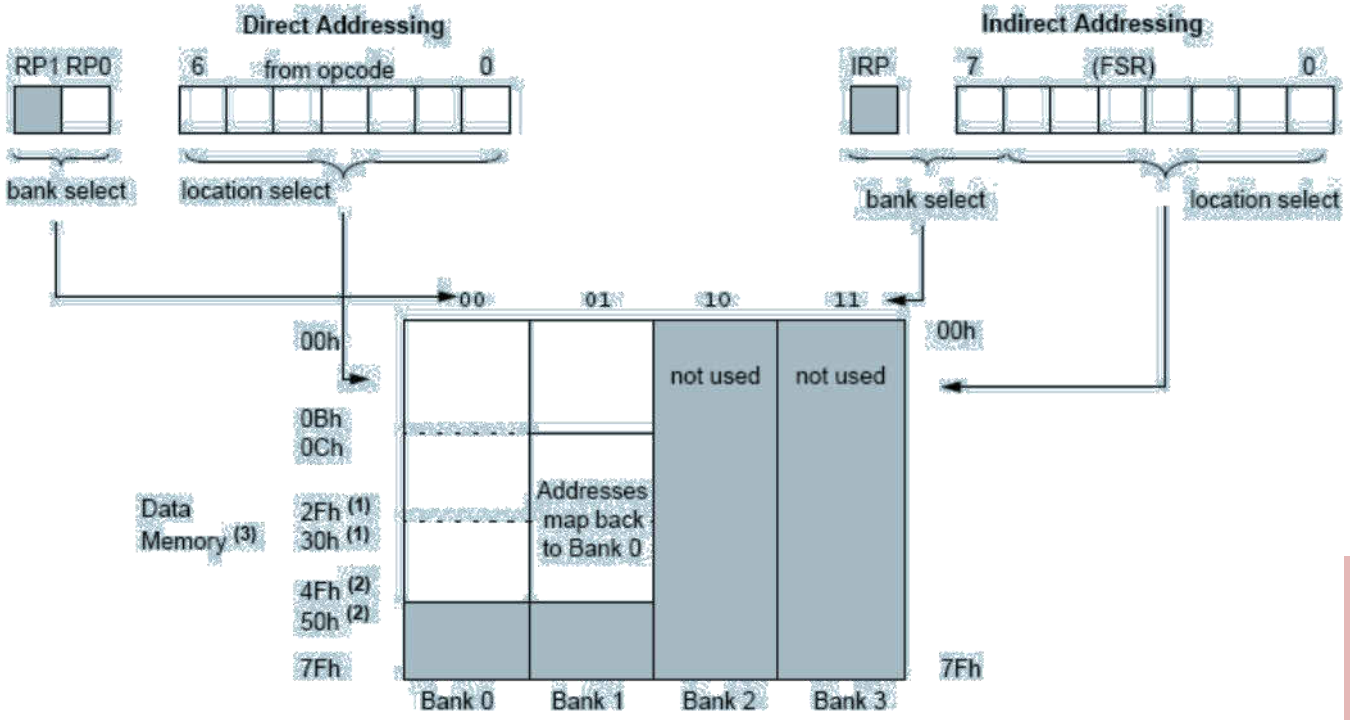
Expliquer le schéma de la figure suivante :



- ✓ Définir une interruption ?
- ✓ D'un point de vue logiciel, que fait généralement un microcontrôleur lorsqu'il détecte un signal 'evenement' ?
- ✓ Pourquoi un microcontrôleur se sert-il de la PILE lorsqu'une interruption se produit provenant d'un périphérique externe ?
- ✓ Combien de sources différentes peuvent interrompre le PIC 16F84 ?
- ✓ Etudier le processus d'exécution d'une interruption dans le cas du PIC 16F84. Discuter…

✓ EXERCICE 3.3

Expliquer avec des exemples les modes d'adressage offerts par le PIC 16F84. Commenter…



EXERCICE 3.4

Répondre aux questions suivantes:

1. What does the word "literal" mean?

- ☐ A number
- ☐ A value such as 0C3h
- ☐ A hexadecimal value
- ☐ A number loaded into W

2. Write the instruction to put 4Ch into W:

- ☐ MOVWF 4Ch;
- ☐ MOVLW 4Ch;
- ☐ MOLVW 4Ch;
- ☐ MOVWL 4Ch;

3. Write the instruction to move 4Ch from W to file 1B:

- ☐ MOVLW 1Bh;
- ☐ MOLVW 4Ch, 1;
- ☐ MOVLW 4Ch;
- ☐ MOVWF 1Bh;

4. Write the instruction to move 4Ch from file 1B to W:

- ☐ MOVF 1Bh, 0;
- ☐ MOVWF 4Ch, 1;

- ☐ MOVWF 1Bh, 1;
- ☐ MOVF 1B, 1;

5. Write the instruction to clear the value 4Ch in file 1B:

- ☐ CLRF 4Ch;
- ☐ CLRF 1Bh, 0;
- ☐ CLRF 4Ch, 0;
- ☐ CLRF 1Bh;

6. Write the instruction to move 4Ch from file 1A to file 1B:

- ☐ MOVF 1B, 1;
- ☐ MOVF 1C, 0;
- ☐ MOVWF 1B, 1;
- ☐ no instruction exists;

7. Write the instruction to decrement file 1B:

- ☐ DECFSZ 1B, 0;
- ☐ DECF 1B, 0;
- ☐ DECFSZ 1B, 1;
- ☐ DECF 1B, 1;

8. After the instruction: BCF 06,3 the file will contain:

- ☐ 0101 0000;
- ☐ 1111 1010;
- ☐ 1111 1000;
- ☐ 1010 1010;

9. The TRIS file is loaded with 0011 1011. Name the output line(s) created by this value.

- ☐ RB0, RB1, RB3, RB4, RB5
- ☐ RB1, RB2, RB4, RB5, RB6.
- ☐ None of the above

10. A LED is connected to the 3rd lowest output. Write the instruction to activate the LED:

- ☐ BSF 06, 3;
- ☐ BSF 06, 2;
- ☐ BSF 06, 4;

11. File 1B holds the value 0000 1100. After the following instruction: BSF 1B,6 the file will contain:

- ☐ 0010 0011;
- ☐ 0010 1100;

- ☐ 0100 1100;
- ☐ 0010 0000;

12. Write the instructions to make bit 3 of TRIS 06 an INPUT:

- ☐ MOVLW 03; and MOVWF 06;
- ☐ MOVLW 04; and MOVWF 06;
- ☐ MOVLW 08; and MOVWF 06;

13. For any file, what is the maximum hex value it can contain?

- ☐ 128;
- ☐ FF;
- ☐ 256h;
- ☐ 255h;

14. Write the instructions to make bit 3 of the input/output file HIGH:

- ☐ MOVLW 03;
- ☐ BSF 06, 3;
- ☐ MOVWF 06;
- ☐ MOVWF 06;
- ☐ MOVLW 11;

15. What value will be contained in file 1B after the instruction: BSF 1B,4:

- ☐ 0000 1000;
- ☐ 0000 1111;
- ☐ 0000 0100;
- ☐ unknown;

16. Write the two instructions to carry out the following: Put 8E into file 1C

- ☐ MOVWF 8Eh; and MOVWF 1Ch;
- ☐ MOVLW 8Eh; and MOVWF 1Ch;
- ☐ MOVLW 8Eh; and MOVLW 1Ch;
- ☐ MOVWF 8Eh; and MOVLW 1Ch;

17. A file containing 0000 1001 is AND with 1100 1010. The result is:

- ☐ 0000 1001;
- ☐ 1000 1000;
- ☐ 0000 1000;
- ☐ 1100 1000;

18. Write the "bit set file" instruction for bit 5 of file 06:

- ☐ BSF 05, 6;

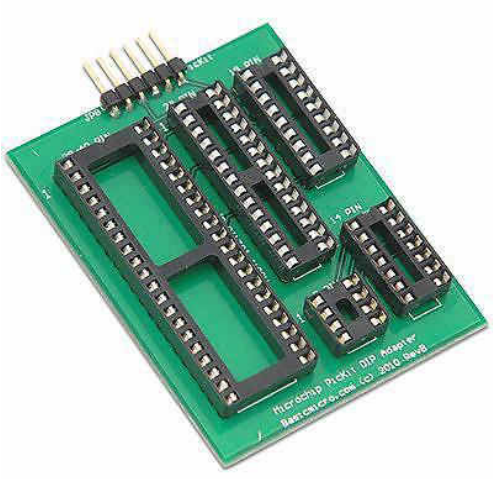
- ☐ BSF 06, 5;
- ☐ BSF 05;
- ☐ BSF 06;

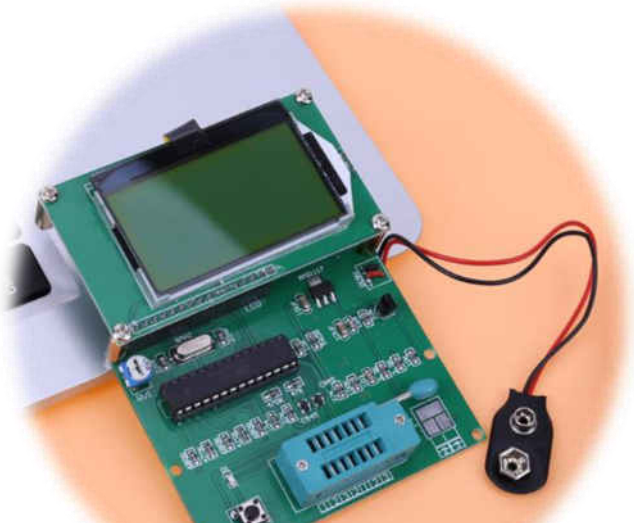
19. A file containing 0110 1001 is XORed with 1100 1001. The result is:

- ☐ 0110 1001;
- ☐ 1001 0110;
- ☐ 1010 0000;
- ☐ 0001 1111;

20. The result of BCF 1B, 3; is:

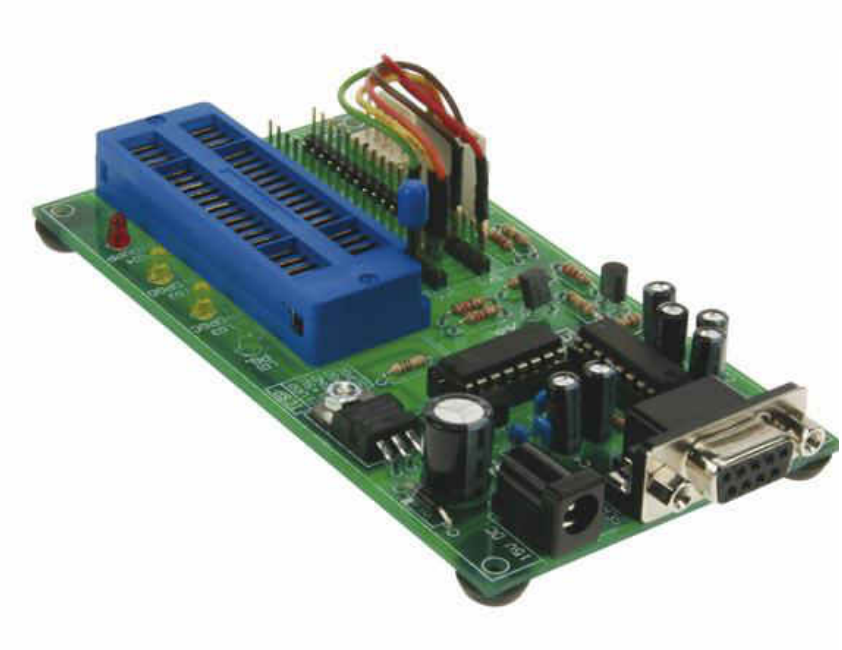
- ☐ 0000 0000;
- ☐ 1111 1100;
- ☐ 1111 1111;
- ☐ None of the above;

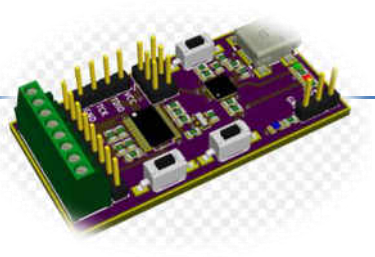




CHAPITRE **4**

Programmation assembleur du PIC 16F84





CHAPITRE 4 Programmation assembleur du PIC 16F84

Objectifs

L'objectif est de

- ✓ Présenter d'une manière profonde les Outils nécessaires à la programmation du PIC 16F84.
- ✓ Décrire la Structure d'un programme assembleur du PIC 16F84.
- ✓ Etudier des exemples des programmes assembleur du PIC 16F84.

Avant Propos

Le logiciel de programmation doit être capable de:

- ✓ Editer un fichier programme en assembleur ou un fichier texte.
- ✓ Compiler un programme écrit en assembleur.
- ✓ Transférer le programme dans le circuit.
- ✓ Simuler le programme afin de vérifier son fonctionnement.

Le compilateur assembleur est construit de façon à pouvoir s'adapter à divers codes assembleurs (Fichier qui contient la description des instructions).

- On peut ainsi programmer pour différent microcontrôleur à condition d'effectuer quelques adaptations du logiciel.
- Le compilateur est en parti compatible avec le langage de MPASM disponible gratuitement sur le site de Microchip.

- Sachant que les PICs ne peuvent pas être programmés beaucoup de fois, il est préférable de simuler le fonctionnement du programme. La solution envisagée ici est une solution logicielle.

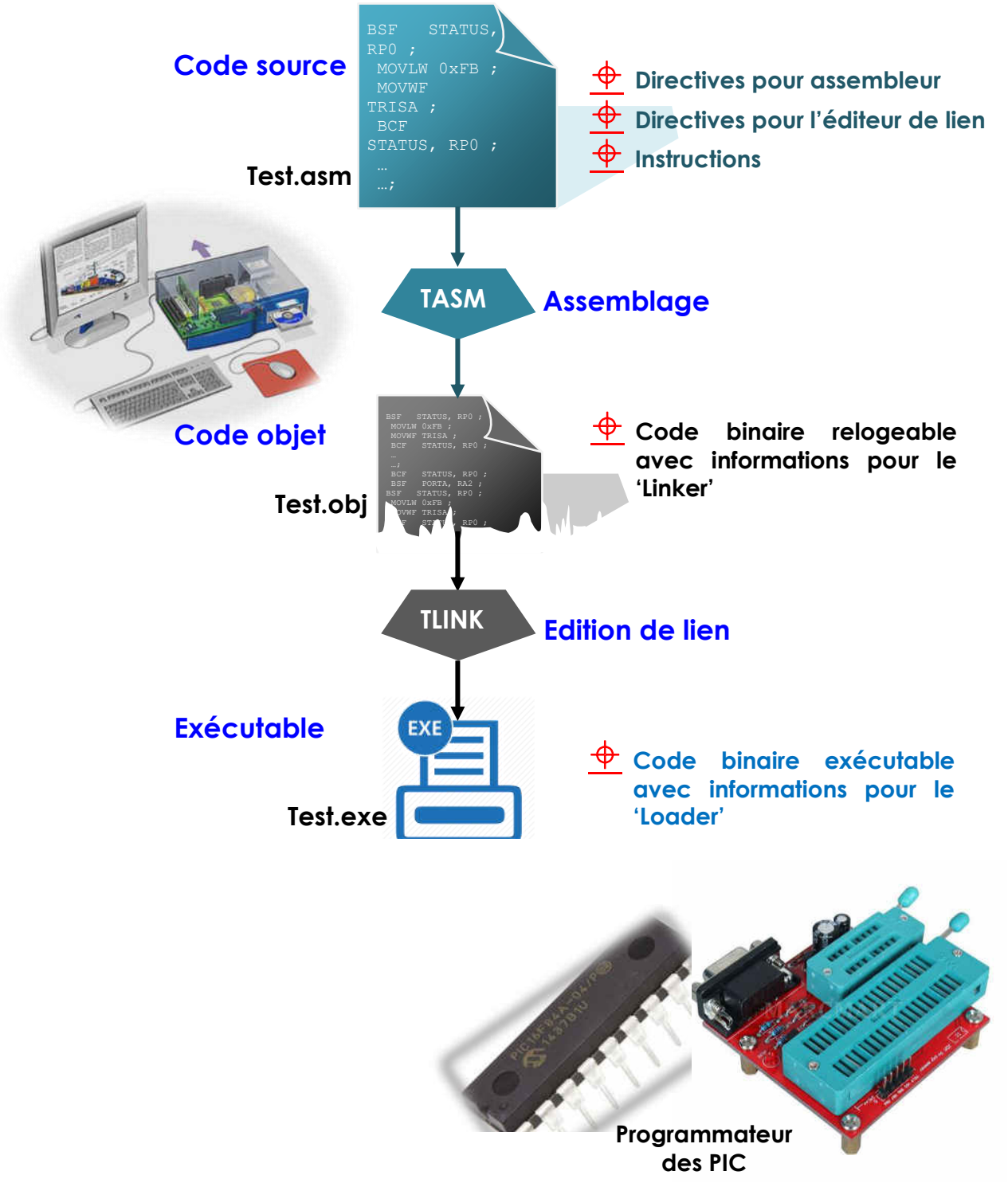


Figure 4.1 : *Chaine de programmation du PIC 16F84.*

4.1. Outils nécessaires à la programmation du PIC 16F84

4.1.1. Les circuits annexes

Afin de faire fonctionner le système à base du PIC 16F84 microcontrôleur, il faut associer 3 petits montages électriques assurant son bon fonctionnement. Il s'agit de :

- ✓ **Un régulateur de tension** : Permet au PIC de recevoir une alimentation très stable afin de ne pas le faire griller
- ✓ **Circuit Horloge** : permet de définir la vitesse d'exécution des instructions.

✓ **Un circuit RESET** : Permet la Remise à Zéro du PIC à chaque mise en tension.

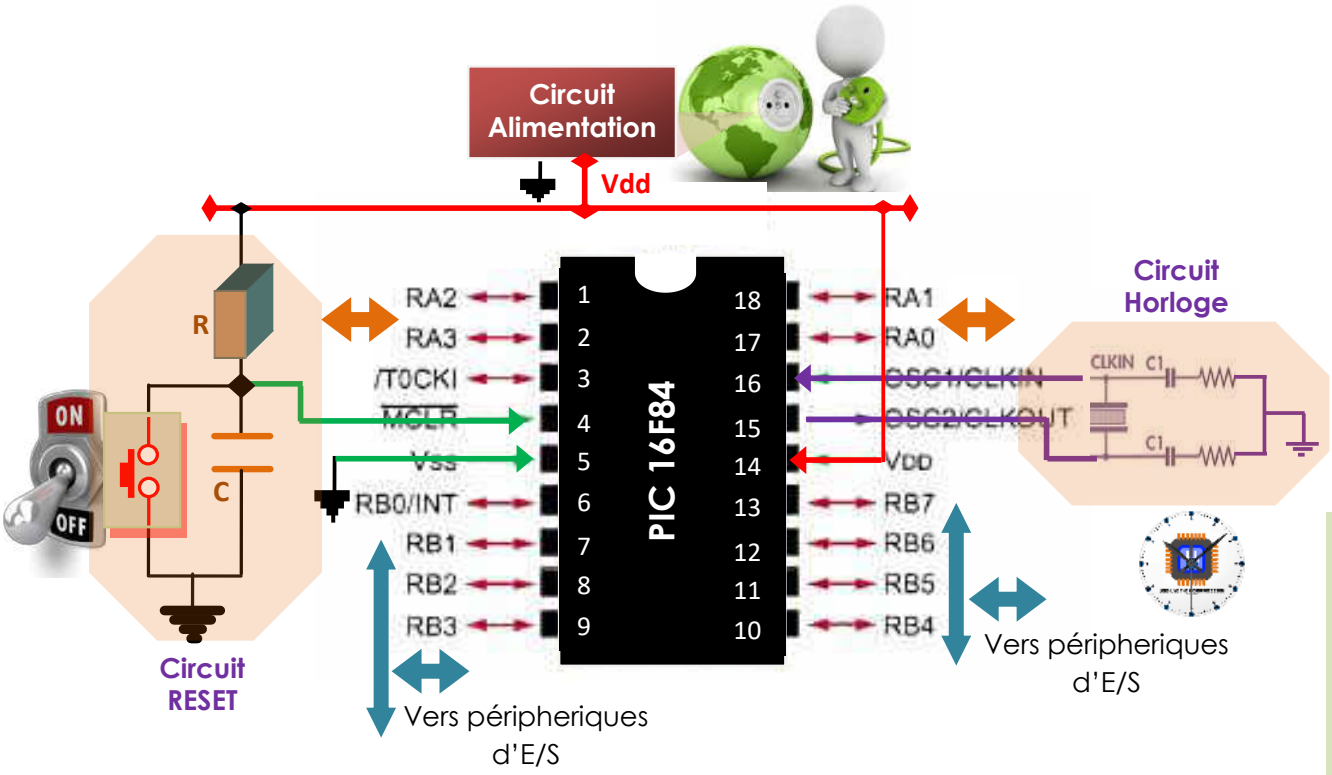


Figure 4.2 : *Circuits annexes du PIC 16F84.*

4.1.2. Systèmes de développement du PIC 16F84

Les processus de développement d'un programme à exécuter sur un PIC sont :

5. Ecrire un programme en langage assembleur du PIC dans un fichier texte et le sauvegarder avec l'extension **.asm**.
6. Compiler ce programme avec l'assembleur **MPASM** fourni par **Microchip**. Le résultat est un fichier exécutable avec l'extension **.hex** contenant une suite d'instruction compréhensible par le PIC.
7. Transplanter le fichier **.hex** dans la mémoire programme du PIC (mémoire **FLASH**) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de **Microchip** ou tout autre programmateur acheté ou réalisé par soit même.
8. Mettre le PIC dans son montage final puis mettre sous tension.

Info

- L'utilisation et la mise en oeuvre très simple des PICs les a rendus extrêmement populaire au point que la société qui les fabrique (**Microchip**) est en passe de devenir le leader mondial dans le domaine des microcontrôleurs devant MOTOROLA et INTEL.
- Il est à noter que **Microchip** propose gratuitement (Téléchargeable à partir du l'URL : http://www.microchip.com/) l'outil de développement MPLAB qui regroupe
  - ✓ **L'éditeur** de texte,
  - ✓ Le **compilateur** MPASM,

- ✓ Un **outil de simulation** et
- ✓ Le **logiciel de programmation**.

Le programmateur lui-même, n'est malheureusement pas gratuit.

Figure 4.3 : *Kit de développement (MPLAB + Programmateur des PIC).*

4.1.3. Mise en œuvre

Une fois le montage est réalisée, il suffit ensuite d'écrire le programme en langage assembleur ou en C sur un ordinateur grâce au logiciel MPLAB de **Microchip** puis de le compiler pour le transformer en langage machine et le transférer dans le PIC grâce à un programmateur du PIC.

Lors de la mise sous tension, tous les registres spécifiques (**SFR**) sont placés dans un état déterminé comme montre la figure ci-dessous.

Tableau 4.1 : *Etats des SFR lors d'un RESET dans le PIC 16F84.*

Registre	Adresse	Valeurs après RESET
INDF	00H - 80H	---- ----
TMR0	01H	xxxx xxxx
PCL	02H – 82H	0000 0000
PCLATH	0AH – 8AH	---0 0000
STATUS	03H – 83H	0001 1xxx
FSR	04H – 84H	xxxx xxxx
PORTA	05H	---x xxxx
PORTB	06H	xxxx xxxx
EEDATA	08H	xxxx xxxx
EEADR	09H	xxxx xxxx

INTCON	0BH - 8BH	0000 000x
OPTION	81H	1111 1111
TRISA	85H	---1 1111
TRISB	86H	1111 1111
EECON1	88H	---0 x000
EECON2	89H	---- ----

*. **–** Non implémenté.
*. **x** = '0' ou '1'.

4.2. Structure d'un programme assembleur du PIC 16F84

Les programmes sources seront écrits en langage assembleur du PIC 16F84. Un programme écrit en assembleur doit respecter une certaine syntaxe et un certain nombre de règles afin qu'il soit facile à lire et à débuguer. Le format utilisé divise chaque ligne en langage assembleur selon 4 champs, à savoir :

- Champ étiquette.
- Champ Code opération.
- Champ opérandes ou adresses des opérandes.
- Champ commentaires.

Etiquette	Code mnémonique		Description linguistique du progrmme
Label :	**Code Opération (COP)**	**Opérande (s) ;**	**Commentaires**

; Initialisation / Configuration :
- Initialisation des registres ;
- Choix de l'horloge ;
- Utilisation de WatchDog ;
- Utilisation de broche d'initialisation (MCLR) ;
- Possibilité de protéger les codes ;
- … etc.
- Initialisation des ports d'entrées / sorties ;
- Déclaration des bibliothèques ;
- Déclaration des variables ;
- Déclaration des fonctions ;
- … etc.

Registres de configuration

Directives d'assemblage

Label	Code Opération	Opérande (s) ;	Commentaires
	---	---	---
	---	---	---

```
                    ---------------     ---------------     ---------------

                    ---------------     ---------------     ---------------

                                        ---------------

                    end.
```

4.2.1. Configuration du PIC 16F84

Afin de garantir la parfaite compatibilité du microcontrôleur avec l'application développée, les PICs intègrent un registre de configuration spécifique (**CONFIG_**). Les modes de fonctionnement assuré par le registre **CONFIG_**, on trouve :

- Choix de l'horloge ;
- Utilisation de WatchDog ;
- Utilisation de broche d'initialisation (MCLR) ;
- Possibilité de protéger les codes ;

D'autres registres de configuration sont désignés dans la partie SFR du mémoire de données (Banque 0 et Banque 1). L'état des ces registres (y inclu le registre CONFIG_) est pris en compte uniquement au démarrage du PIC et ne peut être modifié en cours de programme. L'ensemble des registres de configuration sont:

- CONFIG_ ;
- STATUT ;
- TRISA ;
- TRISB ;
- INTCON ;
- OPTION ;
- EECON1.

Tableau 4.1 : *SFR de banque 0 dans le PIC 16F84.*

Bank 0									
Registre	**Adresse**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
INDF	**00H**	Utilise le contenu de **FSR** pour adresser la mémoire							
TMR0	**01H**	Horloge / Compteur en temps réel							
PCL	**02H**	Octet du poids faible de **PC**							

STATUS	**03H**	**IRP**	**RP1**	**RP0**	**TO#**	**PD#**	**Z**	**DC**	**C**
FSR	**04H**	Adressage indirect avec **INDF**							
PORTA	**05H**	-	-	-	RA4	RA3	RA2	RA1	RA0
PORTB	**06H**	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
	07H	Non implémenté, lu comme étant à '0'							
EEDATA	**08H**	Registre de données EEPROM							
EEADR	**09H**	Registre d'adresses EEPROM							
PCLATH	**0AH**	-	-	-	S'écrit sur les 5 bits de **PCH**				
INTCON	**0BH**	**GIE**	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Tableau 4.1 : *SFR de banque 1 dans le PIC 16F84.*

Bank 1									
Registre	**Adresse**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
INDF	**80H**	Utilise le contenu de **FSR** pour adresser la mémoire							
OPTION	**81H**	**RBPU**	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**
PCL	**82H**	Octet du poids faible de **PC**							
STATUS	**83H**	**IRP**	**RP1**	**RP0**	**TO#**	**PD#**	**Z**	**DC**	**C**
FSR	**84H**	Adressage indirect avec **INDF**							
TRISA	**85H**	-	-	-	Configuration du **PORT A**				
TRISB	**86H**	Configuration du **PORT B**							
	87H	Non implémenté, lu comme étant à '0'							
EECON1	**88H**				**EEIF**	**WRERR**	**WREN**	**WR**	**RD**
EECON2	**89H**	Registre de commande EEPROM							
PCLATH	**0AH**	-	-	-	S'écrit sur les 5 bits de **PCH**				
INTCON	**0BH**	**GIE**	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

4.2.2. Directives d'assemblage MPASM

Les directives de l'assembleur sont des 'instructions' qu'on ajoute dans le programme et qui seront interprétées par l'assembleur MPASM. Ce ne sont pas des instructions destinées au PIC mais des commandes destinées à l'assembleur MPASM. Le tableau ci-dessous illustre les différentes directives les plus utilisées par l'assembleur MPASM.

Tableau 4.5 : *Les directives les plus utilisées par l'assembleur MPASM.*

Directive	**Description**
Directives de configuration du PIC 16F84	

ORG	Définit la position dans la mémoire programme à partir de laquelle seront inscrites les instructions suivantes. **ORG** 0x000 ;
LIST	Permet de définir un certain nombre de paramètres comme ✓ Le processeur utilisé (**p**) ; ✓ La base par défaut pour les nombres (**r**) ; ✓ Le format du fichier hex à produire (**f**) ; ✓ … etc. **LIST** p=16F84A, r=dec, f=inhx8m
INCLUDE	Permet d'insérer un fichier source. Par exemple le fichier **p16f84A.inc** contient la définition d'un certain nombre de constante comme les noms des registres ainsi que les noms de certain bits; **INCLUDE** "p16f84A.inc"
CONFIG	Permet de définir les 14 fusibles de configuration qui seront copié dans l'EEPROM de configuration lors de l'implantation du programme dans le PIC (Protection de code, Type d'oscillateur, Chien de garde et temporisation lors de RESET) ✓ **__CONFIG B'11111111111001'** ✓ **__CONFIG H'3FF9'** Si le fichier p16f84.inc a été inséré, on peut utiliser les constantes prédéfinies : ✓ **__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC**

_CP_OFF	Aucune protection en lecture du pic (choix par défaut).
_WDT_ON	WatchDog en service par défaut (le mettre à OFF sur le montage).
_PWRTE_ON	Délai de démarrage du pic à la mise sous tension (OFF par défaut).
_HS_ON	Oscillateur à quartz à grande vitesse (_XT_OSC sur le montage: quartz lent ; _RS_OSC par défaut).
_BODEN_ON	Reset du pic si chute de tension (choix par défaut).
_LVP_ON	Utilisation de RB3 comme broche de programmation sous 5V (choix par défaut).
_CPD_OFF	Non protection en écriture de la zone EEPROM (choix par défaut).
_DATA_CP_OFF	Non protection en écriture de la zone RAM (choix par défaut).
_WRT_ON	Non protection de la mémoire FLASH (lié à _CP_) (choix par défaut).
_DEBUG_OFF	Non utilisation d'un débuggeur (avec RB6 et RB7) (choix par défaut).

Directives d'initialisation des variables de programme

<table>
<tr><td>EQU</td><td>
Permet de définir une constante ou une variable.
XX EQU 0x20
✓ Chaque fois que le compilateur rencontrera XX, il la remplacera soit par la constante 0x20.
✓ Ca peut être une constante s'il s'agit d'une instruction avec adressage immédiat, ou d'une adresse s'il s'agit d'une nstruction avec adressage direct.
FSR EQU H'04' ;
✓ MOVF FSR ; Ecriture de W dans FSR
</td></tr>
<tr><td>#DEFINE</td><td>
Définit un texte de substitution
#DEFINE fonct(x,y,z) (y-2z+x)
✓ Chaque fois que le compilateur rencontrera le texte fonct(x,y,z), il le remplacera par (y-2z+x).
#DEFINE MONBIT PORTA
✓ Chaque fois que le compilateur rencontrera le texte MONBIT, il le remplacera par PORTA.
bcf MONBIT; Met le bit 1 du port A à 0 (équivalent à BCF PORTA, 1)
</td></tr>
<tr><td>CBLOCK /ENDC</td><td>
Définit un bloc de constante
CBLOCK 0X0C ; var1=0x0C, var2=0x0D, k=0x0D
var1, var2 ;
var3 ;
ENDC.
</td></tr>
<tr><td>DE</td><td>
Pour déclarer des donnés qui seront stockée dans l'EEPROM de donnée au moment de l'implantation du programme sur le PIC.
ORG 0x2100
DE "Programmer un PIC, rien de plus simple", 70, 'Z'
</td></tr>
<tr><td>DT</td><td>
Pour déclarer un tableau RETLW
proc addwf PCL, f ; Saut à la position
DT "Programmer un PIC", 23, 0x47 ; L'assembleur remplacera cette ligne par la suite d'instructions :
RTLW 'P'
RTLW 'r'
RTLW 'o'
. . .
RTLW 'C'
RTLW 23
RTLW 0x47
</td></tr>
<tr><td>END</td><td>Indique la fin du programme.</td></tr>
</table>

*. Pour plus d'information on se réfèrera à l'aide **MPASM** dans **Mplab** (menu « **Help** » ou encore dans ledocument « **MPASM/MPLINK PICmicro MCU Quick Chart** » ou « **MPASM USER'S GUIDE** ».

Exemple


```
; Programme Assembleur PIC 16F84
; Allumer une LED par bouton poussoir
; Sur la broche RB2

LIST p=16F84                          ; Initialisation / Configuration du PIC
include "P16F84.inc"
__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC
        org 0x0000
            bsf STATUS, RP0           ; sélectionner bank 1
            movlw b'11111111'         ; Port B en entrée
            movwf TRISB ;
            movlw b'00000000'         ; Port A en sortie
            movwf TRISA
            bcf STATUS, RP0           ; Sélectionner bank 0
boucle :    btfsc PORTB, 2            ; Tester RB2, sauter si vaut 0
            bcf PORTA, 2              ; Sinon on allume la LED
            btfss PORTB, 2            ; Tester RB2, sauter si vaut 1
            bsf PORTA, 2              ; RB2 vaut 0, donc LED
            goto boucle ;
            end.                      ; Fin de programme
```

4.2.2.1. Format des nombres

Toutes les variables sont des variables 8 bits pour le cas du PIC 16F84, allant de 0 à 255 (00h à FFh). L'assembleur des PICs reconnaît les nombres en décimal, hexadécimal, binaire ou octal. Pour préciser la base il faut utiliser les **préfixes** précisés dans le tableau ci-dessous.

Tableau 4.6 : *Format des nombres pour le PIC 16F84.*

Base	Préfixe	Exemple (36_{10})
Décimal	**D'nnn'** **.nnn**	**D'36'** **.36**
Hexadécimal	**H'nn'** **0xnn** **nnh**	**H'24'** **0x24** **24h**
Octal	**O'nnn'**	**O'44'**

Binaire	B'....'	B'00100100'

En résumé

Il existe différentes écoles indiquant comment doit être organisé un programme. Voici un exemple d'organisation :

1. Quelques lignes de commentaire précisant la fonction du programme ;
2. Configuration ;

```
LIST p=16f84, f=inhx8m, r = dec
#INCLUDE "p16f84.inc"
__CONFIG H'3FF9'
```

- Fichiers d'en-tête
- Bibliothèque des instructions pour le PIC16F84

3. Définition des constantes et des variables ;

```
LED equ 0
x equ 0x0C
cblock 0x0D
    y,z
    u,v,w
endc
```

- Déclaration des variables et constantes :
  - LED = 00H ;
  - X = 0CH ;
  - Y = 0DH, Z = 0EH, U = 0FH, V = 10H, W = 11H ;

4. Si le programme utilise des interruptions, mettre à l'adresse 0000H (adresse du RESET) une instruction de branchement au début du programme principal ;

```
org 0x0000
goto debut
```

- Adresse de départ après un RESET (Initialisation).

5. Ecrire la routine d'interruption à l'adresse 0004H ;

```
ORG 0x0000
-----------
-----------
-----------
```

- Adresse de saut vers la routine de l'interruption.

6. **RETFIE**
   Ecrire les sous programmes (s'il y en a). Chaque procédure commence par une étiquette qui représente son nom, et se termine par l'instruction RETURN ;

```
Label :  .....;
         .....;
         .....;
         return ;
```

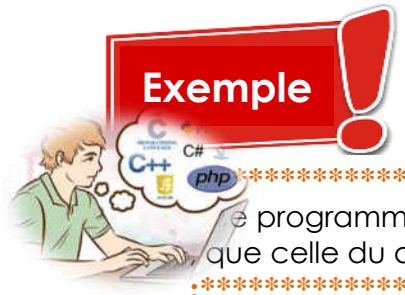
- Sous programme (Procédure) d'une tâche quelconque

7. Ecrire le programme principal (commençant par l'étiquette début: Si les étapes 4 et 5 sont présentés) ;

```
début :  ……;
         ……;
         ……;
         end ;
```

Programme principal, effectuant la tâche demandée.

8. Terminer avec la directive **end**.



Générateur d'un signal périodique de fréquence 24 fois plus faible que celle du quartz

Commentaires

```
;*****************************************************************
;  Ce programme génère une horloge en RB0 de fréquence 24 fois plus faible
;  que celle du quartz.
;*****************************************************************
```

Fichiers d'en-tête **(header files)**

```
LIST P=16F84A ;              Directive qui définit le processeur utilisé
#include <P16F84A.INC> ;     Fichier de définition des constantes
```

Bibliothèque des instructions pour le PIC16F84

Configuration materielle

```
;*****************************************************************
; Bits de configuration
;*****************************************************************
__CONFIG _HS_OSC & _WDT_OFF & _CP_OFF & _CPD_OFF & _LVP_OFF

; _XS_OSC          L'oscillateur est configuré en oscillateur à quartz haute fréquence
; _WDT_OFF         Le watchdog est désactivé
; _CP_OFF          Le code de protection de la mémoire programme est désactivé
; _CPD_OFF         Le code de protection de la mémoire EEPROM est désactivé
; _LVP_OFF         La programmation basse tension est désactivée
; Ces opérations sont nécessaires pour fonctionner en mode "debug"
;*****************************************************************
; Démarrage sur RESET
;*****************************************************************
```

Commentaires

```
        org 0x00 ;           Adresse de départ après reset
        goto début
        org 0x10 ;           Adresse de début du programme
;*****************************************************************
; Initialisation
;*****************************************************************
; Initialisation du PORTB en sortie
```

Etiquette

```
début : bcf STATUS, RP0 ;
        bcf STATUS, RP1 ;    Passage en banque 0
        clrf PORTB ;         RAZ des bascules D
        bsf STATUS, RP0 ;    Passage en banque 1
        movlw b'00000000' ;
        movwf TRISB ;        PORTB en sortie
        bcf STATUS, RP0 ;    Retour en banque 0
;*****************************************************************
```

Etiquette

; Programme principal
;***
boucle : **bsf** PORTB,0 ; Mise à 1 de la sortie
nop ; 2 temps morts pour compenser le saut
nop ;
bcf PORTB,0 ; Mise à 0 de la sortie
goto boucle ; Rebouclage
end ; Directive signalant la fin du programme
;***

Etiquette

Fin de programme

Exemple

Programme d'affichage d'une LED

;***
; Programme de commande l'affichage d'une LED. On connecte un interrupteur sur
; RB0 (entrée) et une LED sur RB1 (sortie)
; Si on place l'interrupteur à 1, la LED doit s'allumer, si on le met à zéro, elle
; doit s'éteindre
;***
LIST p=16f84A ; Définition de processeur
#INCLUDE "p16f84A.inc" ; Définitions de variables
;***
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF ; Fixe les fusibles
;***

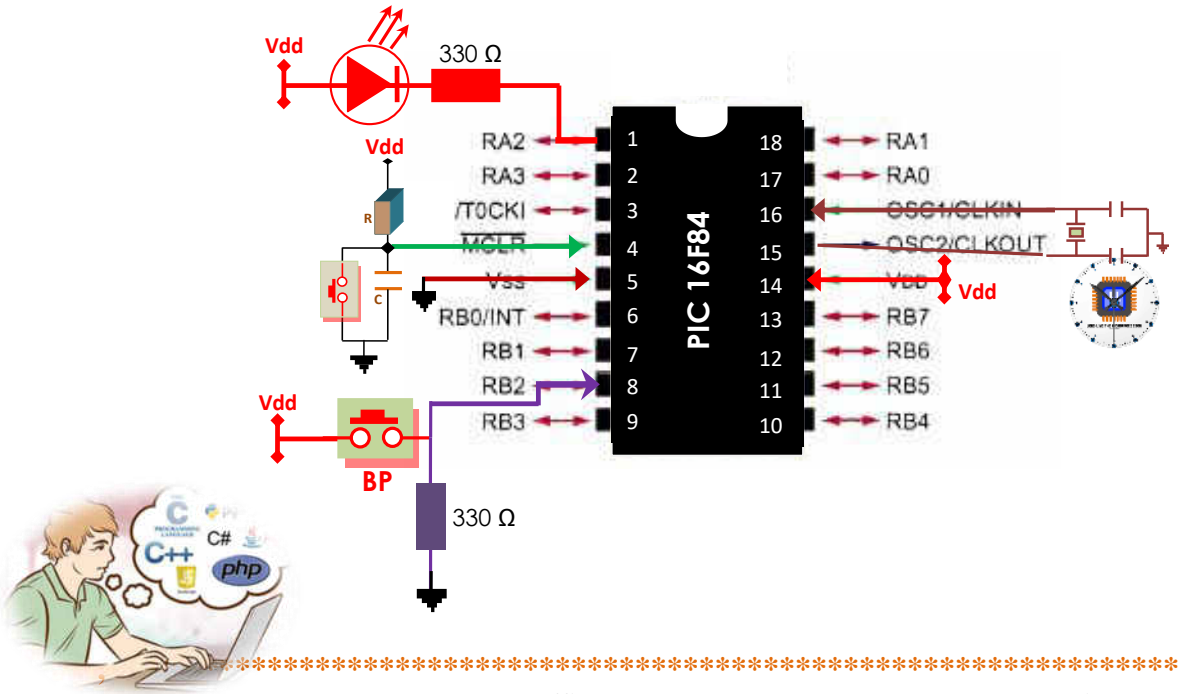
bsf STATUS, RP0 ; Bank 1
movlw B'00000001' ; Charger w par 01h
movwf TRISB ; Configurer RB0 en entrée et les autres en sortie
bcf STATUS, RP0 ; Bank 0

test : **btfss** PORTB, 0 ; Teste si RB0 = 1
goto off ; Sinon goto off
bsf PORTB, 1 ; Si oui mettre RB1=1
goto test ; et goto tst

off : **bcf** PORTB,1 ; Mettre Rb1=0
goto test ; Revenir au test
end ; Fin du programme
;***

Exemple

Soit l'exemple du montage ci-dessous qui permet d'allumer une LED par un bouton poussoir.




```
;*****************************************************************************
; Programme de commande l'affichage d'une LED. On connecte un interrupteur sur
; RB2 (entrée) et une LED sur RA2 (sortie)
; Si on place l'interrupteur à 1, la LED doit s'allumer, si on le met à zéro, elle
; doit s'éteindre
;*****************************************************************************
LIST p=16f84A ;                 Définition de processeur
#INCLUDE "p16f84A.inc" ;        Définitions de variables
;*****************************************************************************
__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC;    Fixe les fusibles
;*****************************************************************************
            org 0x0000 ;
            bsf STATUS, RP0 ;           Sélectionner bank 1
            movlw b'11111111' ;         Port B en entrée
            movwf TRISB ;
            movlw b'00000000' ;         Port A en sortie
            movwf TRISA ;
            bcf STATUS, RP0 ;           Sélectionner bank 0
boucle :    btfsc PORTB, 2 ;            Tester RB2, sauter si vaut 0
            bcf PORTA, 2 ;              Sinon on allume la LED
            btfss PORTB, 2 ;            Tester RB2, sauter si vaut 1
            bsf PORTA, 2 ;              RB2 vaut 0, donc LED
            goto boucle ;
            end.
;*****************************************************************************
```

4.3. Programmation des entrées/sorties du PIC 16F84

4.3.1. Ports d'Entrées / Sorties

Pour dialoguer avec l'extérieur (application) le PIC 16F84 vous met à disposition 13 Entrées / Sorties programmables individuellement soit en entrée soit en sortie. Ces 13 E/S sont

issues de 2 ports nommés PORT A pour les 5 E/S RA0 à RA3 et PORT B pour les 8 E/S RB0 à RB7.

- Le port A est un port d'E/S de 5 bits (RA*i* (**i** = 0, 1, 2, 3, 4)).
  - ✓ La configuration de direction pour chaque bit du port est déterminée avec le registre TRISA.
    - Bit *i* de TRISA = 0 → RA*i* de PORTA configuré en **sortie**.
    - Bit *i* de TRISA = 1 → RA*i* de PORTA configuré en **entrée**.
  - ✓ La broche RA4 est multiplexée avec l'entrée horloge du Timer TMR0. Elle peut être utilisée soit:
    - Comme E/S normale du port A.
    - Comme entrée horloge pour le Timer TMR0.
    - Le choix se fait à l'aide du bit **T0CS** du registre **OPTION**.

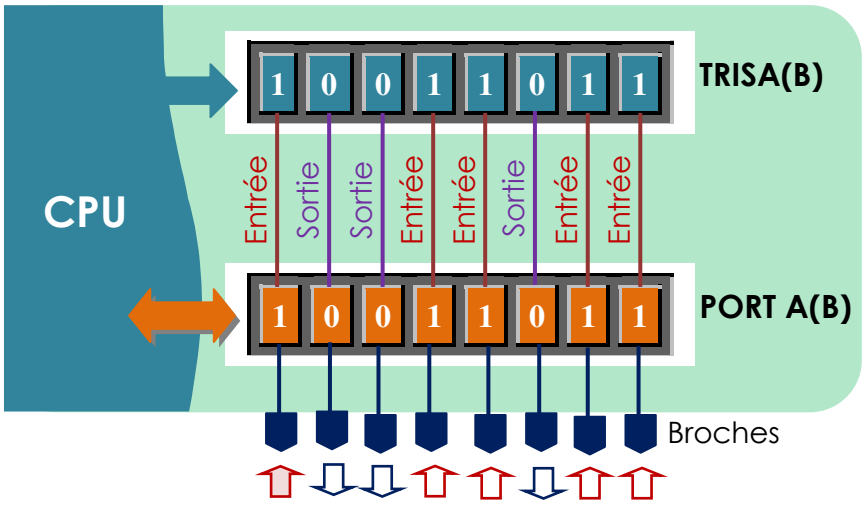


Figure 2.21 : *Exemple de configuration des ports d'E/S.*

- Le port B désigné par PORTB est un port bidirectionnel de 8 bits ((RB*i* (**i** = 0, 1, …,7)).
  - ✓ La configuration de direction se fait à l'aide du registre TRISB.
    - Même configuration que le PORTA.
  - ✓ La broche RB0 peut aussi servir d'entrée d'interruption externe INT.
  - ✓ Les quatre bits de poids fort (RB7-RB4) peuvent être utilisés pour déclencher une interruption RBI sur changement d'état.
  - ✓ Toutes les broches sont compatibles TTL.

Exemple

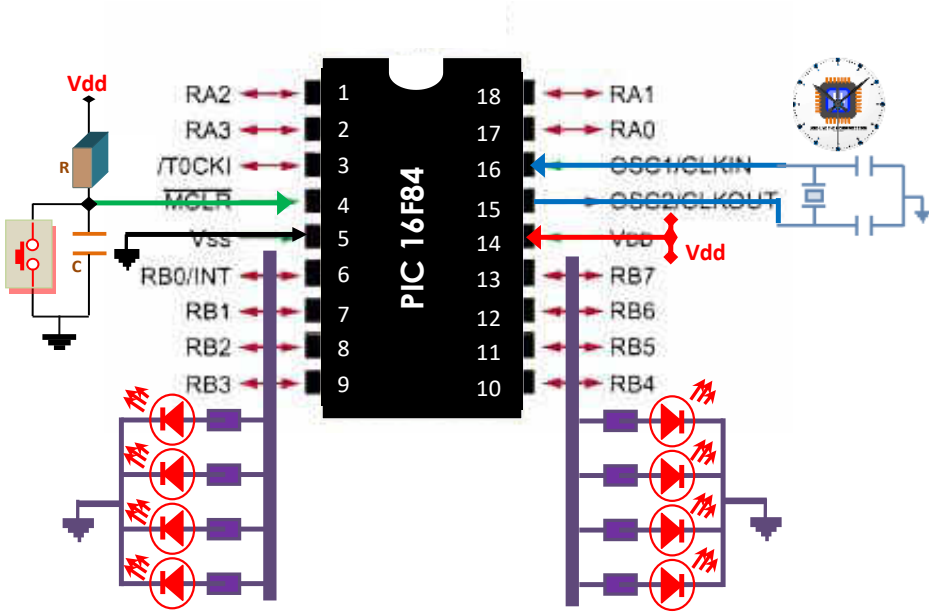
Programme qui permet de recopie d'un port en entrée sur un autre comme sortie.

```
;       Recopier le port A sur le Port B
;*****************************************************************************
LIST p=16f84A ;                     Définition de processeur
#INCLUDE "p16f84A.inc" ;            Définitions de variables
;*****************************************************************************
__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC;    Fixe les fusibles
;*****************************************************************************
; Configuration des registres
;*****************************************************************************

        PORTA   equ     0x0005  ;       Adresse du PORTA
        PORTB   equ     0x0006  ;       Adresse du PORTB
        TRISA   equ     0x0085  ;       Adresse du registre de direction du PORTB
        TRISB   equ     0x0086  ;       Adresse du registre de direction du PORTA
        STATUS  equ     0x0003  ;       Le bit 5 permet d'acceder à la Banque 1
                                ;       ou 0 ce qui donne acces au TRIS ou au
                                ;       PORT

;*****************************************************************************
; Initialisation
;*****************************************************************************

        org     00H;            Apres le reset le PC pointe l'adresse 00
        goto    debut;          On saute les 5 premiers octets car à l'adresse
                        ;       o4H a l'adresse d'interruption. On prend
                        ;       l'habitude de ne pas ecrire sur ce segment
                        ;       en sautant simplement jusqu'apres ce segment


        org     05H ;

debut :     clrf    PORTB;      Mise a zero des latches de sorties
            clrf    PORTA;      Mise a zero des latches de sorties
            bsf     STATUS, 05; Selection de Bank 1 pour l'accès au TRIS
            movlw   00H ;
            movwf   TRISB;      Declaration du portb en sortie
            movlw   1fH;
            movwf   TRISA ;     Declaration du porta en entree
            bcf     STATUS, 05; Selection de Bank 0 pour l'accès au
                            ;   PORT
;*****************************************************************************
; Programme principal
;*****************************************************************************

boucle :    movf    PORTA, 0;   Chargement dans le registre Work du
                            ;   Port A
            movwf   PORTB;      Chargement du contenu de w dans le
                        ;       fichier PORTB
            goto    boucle ;    Recopie permanente
```

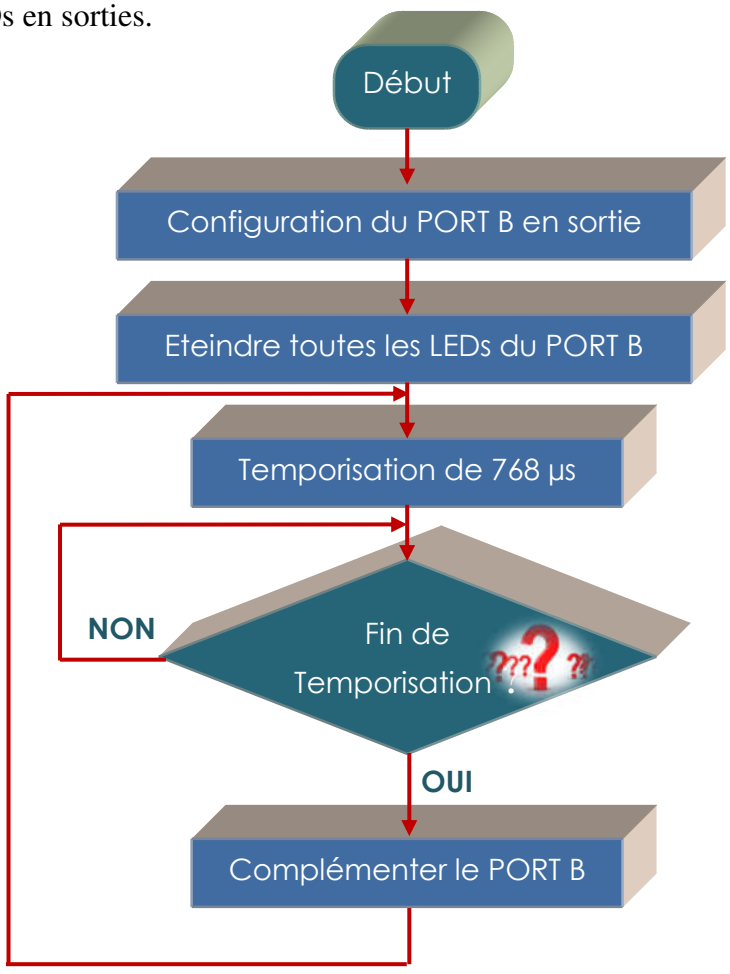
```
                end.
;*****************************************************************************
```

Exemple

Clignotement de toutes les LED sur un port en sortie à l'aide d'une temporisation par boucle. Le montage ci-dessous schématise le mode de fonctionnement du système désiré.



L'organigramme ci-dessous illustre le processus de configuration du PORT B pour la commande des LEDs en sorties.




```
;*****************************************************************************
; Programme de clignotement des LEDs
;*****************************************************************************
LIST p=16f84A ;                 Définition de processeur
#INCLUDE "p16f84A.inc" ;        Définitions de variables
;*****************************************************************************
__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC;    Fixe les fusibles
;*****************************************************************************
        PORTB   equ     0x0006;         Adresse du PORT B
        TRISB   equ     0x0086;         Adresse du registre de direction du PORTA
        STATUS  equ     0x0003;         Le bit 5 permet d'acceder à la Banque 1
                                ;       ou '0' ce qui donne acces au TRIS ou au
                                ;       PORT
;*****************************************************************************
; Reservation memoire *
; Les registres occupent la RAM jusqu'en OBh
;*****************************************************************************
        COMPT equ       0Ch
;*****************************************************************************
; Initialisation
;*****************************************************************************

        org     00h;            Apres le reset le PC pointe l'adresse 00
        goto    debut;          On saute les 5 premiers octets car à l'adresse
                ;               04H on a l'adresse d'interruption On prend
                ;               l'habitude de ne pas ecrire sur ce segment en
                ;               sautant simplement jusqu'apres ce segment


debut :         clrf    PORTB;          Mise a zero des latches de sorties
                bsf     STATUS, 05;     Selection de Bank 1 pour l'accès au TRIS
                movlw   00h;
                movwf   TRISB;          Declaration du portb en sortie
                bcf     STATUS, 05;     Selection de Bank 0 pour l'accès au
                                ;       PORT
;*****************************************************************************
; Programme principal
;*****************************************************************************
                clrf    PORTB;          Extinction de toutes les LEDS
boucle :        movlw   0x00ff;         Chargement de COMPT de la valeur
                                ;       maximale
                movwf   COMPT ;
tempo :         decfsz  COMPT, 1;       Le temps de decompter 256 valeurs
                goto    tempo;          Calcul du temps : 1 instruction
                                ;       = Tquartz/4 = 1µs = 1 cycle
                                ;       l'instruction decfz est de 2 cycles
                                ;       2 instructions = 3µs repeter 256 fois
                                ;       t = 768µs
                comf    PORTB, 1;       on inverse l'etat des LEDs
```

```
        goto    boucle;         Recopie permanente
        end.
;*****************************************************************************
```

4.3.2. Mise en veille du PIC 16F84

Le mode sleep permet au PIC de se mettre en sommeil afin de limiter la consommation d'energie. Ceci est réalisé à l'aide de l'instruction **SLEEP**. Le PIC est placé alors en sommeil et cesse d'exécuter le programme:

- ✓ Le Timer du Watchdog est remis à '0'.
- ✓ Le bit **TO** du registre **STATUS** est mis à '1'.
- ✓ Le bit **PD** du registre **STATUS** est mis à '0'.
- ✓ L'oscillateur est arrêté.

Le Watchdog continue de compter s'il est mis en service. Le passage en mode « Sleep » n'a réellement d'intérêt que s'il est possible d'en sortir. Les seuls événements susceptibles de réveiller le PIC:

- ✓ Niveau bas sur le pin **MCLR** ce qui provoque un reset à l'adresse 0000H.
- ✓ Ecoulement du temps du Timer du Watchdog s'il est mis en service. Dans ce cas, le PIC est seulement réveillé, et il exécutera l'instruction qui suit l'instruction **SLEEP**.
- ✓ Une interruption **RB0/INT**, **RB** ou **EEPROM**, si elles sont programmées et que le bit **GIE** n'est pas positionné (on positionne juste les bits des interruptions). Dans ce cas le PIC se réveille est exécute l'instruction suivante.

Exemple

Programme de mise en veille du PIC.

```
;*****************************************************************************
;                   Programme de mise en veille du PIC 16F84
;*****************************************************************************
LIST p=16f84A ;                 Définition de processeur
#INCLUDE "p16f84A.inc" ;        Définitions de variables
;*****************************************************************************
__CONFIG _CP_ON & _WDT_ON & _PWRTE_ON & _HS_OSC;    Fixe les fusibles
;*****************************************************************************
        OPTIONVAL equ H'8E' ;   Valeur à mettre dans le registre option ;
                        ;       Préscaler wdt à 64 (1152ms nom et 448ms
                        ;       min)
;*****************************************************************************
#DEFINE LED PORTA, 2 ;          LED de sortie
        bcf LED ;               LED éteinte
;*****************************************************************************
; Programme principal
;*****************************************************************************
Start : bsf LED ;               Allumage de LED
                sleep ;         Mise en sommeil
                bcf LED ;       Extinction de la LED
                sleep ;         Mise en sommeil
                goto Start ;    Boucler
```

```
end ;                          Fin
;*****************************************************************************
```

4.3.3. Traitement des interruptions du PIC 16F84

Dans le cas du PIC 16F84, les différentes sources d'interruption sont au nombre de 4, à savoir :

- ✓ **INT**: Interruption externe broche **RB0/INT**.
- ✓ **TMR0**: Interruption interne fin du comptage.
- ✓ **PORTB**: Interruption externe changement d'état du **PORTB** (**RB4 à RB7**).
- ✓ **EEPROM**: Interruption interne fin d'écriture en **EEPROM**.

La séquence classique de fonctionnement d'une interruption est la suivante :

- ✓ Détection de l'événement déclencheur.
- ✓ Fin de l'instruction en cours.
- ✓ Sauvegarde de l'adresse de retour.
- ✓ Déroutement vers la routine d'interruption.
- ✓ Sauvegarde du contexte.
- ✓ Identification de l'événement survenu.
- ✓ Traitement de l'interruption correspondante.
- ✓ Restauration du contexte.
- ✓ Retour au programme principal.

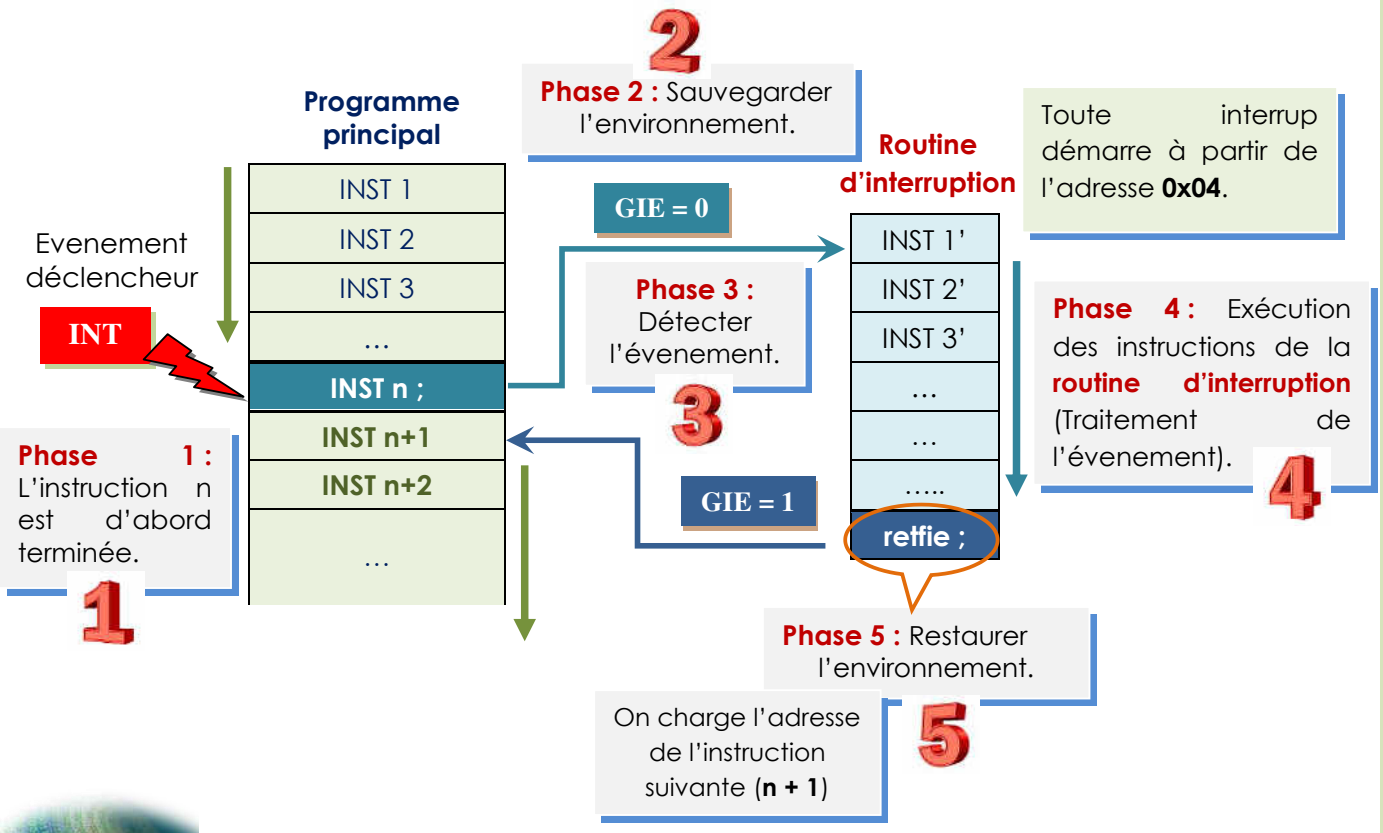


Figure 4.5 : *Séquence classique de fonctionnement d'une interruption.*

La structure générale d'un programme avec interruption est donnée comme suit :


```
;*****************************************************************************
; Structure d'un programme avec interruption
;*****************************************************************************
; Configuration du PIC
;*****************************************************************************
list p=16f84, f=inhx8m ;          Type de PIC et format de fichier
__config B'11111111110001' ;      Configuration du PIC
#include "p16f84.inc" ;           Bibliotheque des instructions pour le PIC16F84
;*****************************************************************************
; Déclaration des variables et constantes du programme
;*****************************************************************************

        Son equ D'129' ;          Définition des constantes
        Note equ H'0C' ;          Définition des Variables

;*****************************************************************************
; Routine d'interruption
;*****************************************************************************

        org H'00' ;               Début du programme (non obligatoire)
        goto début ;

;*****************************************************************************
        org H'04' ;               Adresse d'interruption

{Programme d'interruption}

        retfie ;                  Retour à l'endroit ou le programme s'est interrompu
;*****************************************************************************
; Programme principal
;*****************************************************************************

début :

{les instructions du programme}

        End.
;*****************************************************************************
```

En Résumé

La structure de base d'une routine d'interruption s'effectue de la manière suivante :

```
;*****************************************************************************
                    ; Routine d'interruption
;*****************************************************************************
                    ; Sauvegarde des registres
        Org 0x04 ;          Adresse des interruptions
        movwf w_temp ;      Sauvegarder W
        swapf STATUS , w ;  Sauvegarder STATUS
;*****************************************************************************
        movwf status_temp ; ---------------------
                    ;       On teste l'origine de l'interruption (on test les bist signal)
                            On traite l'interruption, on efface son bit de signal (flag ou drapeau)
                            On restaure les registres
;*****************************************************************************
        swapf status_temp , w ; Restaurer STATUS
        movwf STATUS ;          -- -- -- -- -- -- -- --
        swapf w-temp , f ;      Restaure W
        swapf w_temp , W ;      -- -- -- -- -- --
        retfie ;                Retour de l'interruption (GIE =1 automatiquement)
;*****************************************************************************
```

Il faut sauvegarder les registres qui permettent au pg principal de fonctionner correctement après interruption.

Instructions de traitement de l'interruption

Exemple

Programme qui permet d'inverser l'allumage d'une LED à chaque pression sur un bouton poussoir **BP**.

- ✓ Pour cela on active les résistances internes de rappel du **PORTB**.
- ✓ L'entrée **RB0/INT** est à '1' et lorsqu'on appui sur BP elle passe à 0 et lorsqu'on relâche elle revient à '1'.

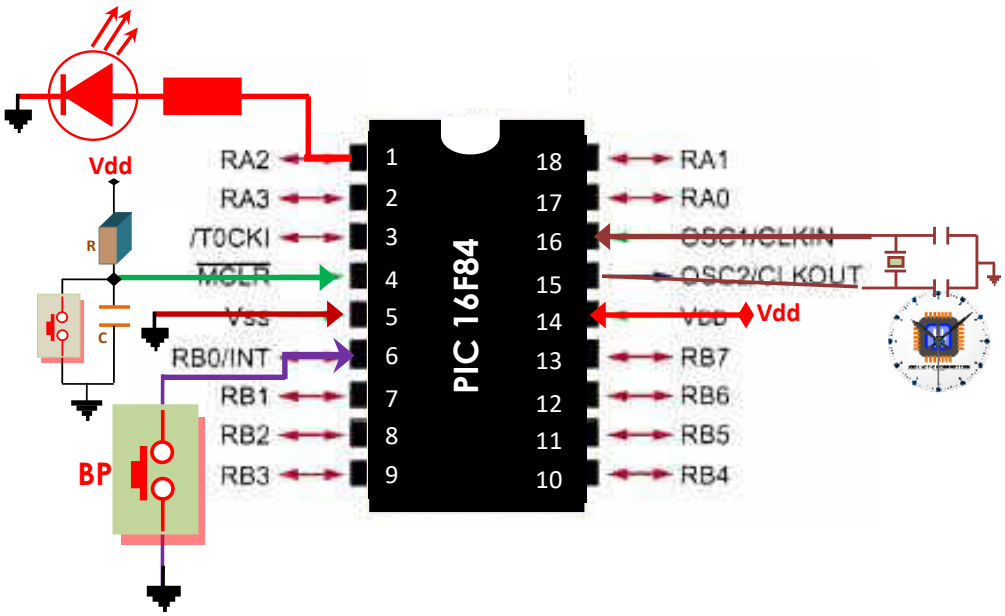


Figure 4.6 : *Montage pour la commande d'une LED.*

La première étape consiste à configurer les registres de contrôle intervenant dans la bonne gestion des interruptions.

INTCON :

GIE =1	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
7	6	5	4	3	2	1	0

OPTION :

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
7	6	5	4	3	2	1	0

1. Calculons la valeur à envoyer dans le registre **OPTION**:

- ✓ **Bit 7 = 0 :** Pour utiliser les résistances de rappel.
- ✓ **Bit 6 = 0 :** Pour fixer le front de l'interruption (**Front descendant**).
- ✓ **Bit 5 – Bit 0 = 0** : Aucune importance concernant le Timer et Watchdog.

La valeur à mettre dans **OPTION** est donc : **00H.**

OPTION :

0	0	0	0	0	0	0	0

2. Pour le registre **INTCON**:

- ✓ **Bit 7 = 1 :** Pour valider les interruptions.
- ✓ **Bit 6 = 0 :** Pas d'interruption EEPROM.
- ✓ **Bit 5 = 0 :** Pas d'interruption TMR0.
- ✓ **Bit 4 = 1 :** Interruption **RB0/INT** activée.
- ✓ **Bit 3 = 0 :** Pas d'interruption PORTB (RB4 - RB7).
- ✓ **Bit 2 = 0 :** Efface flag du TMR0.
- ✓ **Bit 1 = 0 :** Efface flag de RB0/INT.
- ✓ **Bit 0 = 0 :** Efface flag du PORTB.

La valeur à mettre dans **INTCON** est donc : **90H.**

INTCON :

1	0	0	1	0	0	0	0

```
;*****************************************************************************
; Ce programme n'est que qu'une démonstration d'une Interruption par bouton-
; poussoir sur la broche RB0 une horloge en RB0 de fréquence 24 fois plus faible
;*****************************************************************************

LIST P=16F84A ;              Directive qui définit le processeur utilisé
#include <P16F84A.INC> ;     Fichier de définition des constantes
```


```
;*****************************************************************************
; Bits de configuration
;*****************************************************************************

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

; _CP_OFF          Code protection OFF
; _PWRTE_ON        Timer reset sur power on en service
; _PWRTE_OFF       Timer reset hors-service
; _WDT_ON          Watch-dog en service
; _WDT_OFF         Watch-dog hors service
; _LP_OSC          Oscillateur quartz basse vitesse
; _XT_OSC          Oscillateur quartz moyenne vitesse
; _HS_OSC          Oscillateur quartz grande vitesse
; _RC_OSC          Oscillateur à réseau RC

;*****************************************************************************
        OPTIONVAL   EQU   H'0000' ;    Valeur registre option
        INTERMASK   EQU   H'0090' ;    Masque d'interruption, interruptions sur RB0
;*****************************************************************************
#DEFINE     Bouton PORTB, 0 ;       Bouton poussoir
#DEFINE     LED PORTA, 2 ;          LED
;*****************************************************************************

        BANK0       macro
                bcf STATUS, RP0 ;       Passer en banque 0
                endm

        BANK1       macro
                bsf STATUS, RP0 ;       Passer en banque1
                endm
;*****************************************************************************
                        ; Déclarations de variables
;*****************************************************************************
        CBLOCK 0x00C ;              Début de la zone variables
                    w_temp :1 ;     Sauvegarde de W dans interruption
                    status_temp : 1 ;   Sauvegarde de STATUS dans interrupt
                    cmpt1 : 1 ;     Compteur de boucles 1 dans tempo
                    cmpt2 : 1 ;     Compteur de boucles 2 dans tempo
                    flags : 1 ;     Un octet pour 8 flags
                            ;       Réservons b0 pour le flag tempo
                ENDC ;              Fin de la zone
```

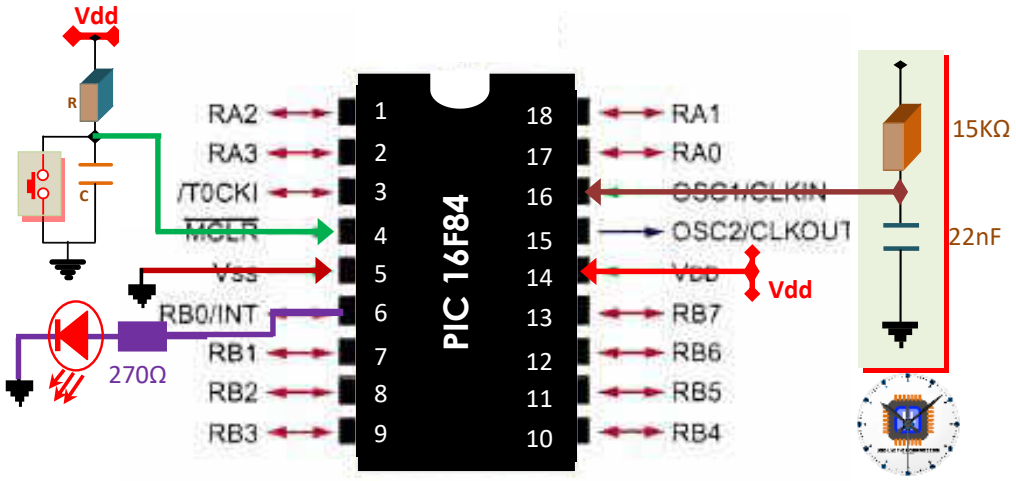
```
        #DEFINE tempoF flags, 0 ;       Définition du flag tempo
;*****************************************************************************

;*****************************************************************************
; Démarrage sur RESET
;*****************************************************************************
            org 0x000 ;                 Adresse de départ après reset
            goto init ;                 Adresse 0 → Vecteur d'initialisation
;*****************************************************************************
                                ; Routine d'interruption
;*****************************************************************************
                                ; Sauvegarder registres
                               ;----------------------------

            org 0x004 ;                 Adresse d'interruption

            movwf w_temp ;              Sauver registre W
            swapf STATUS, w ;           Swap status avec résultat dans w
            movwf status_temp ;         Sauver status swappé

                                ; Appel de l'interruption
                               ;-----------------------------

            call intrb0 ;               Traiter l'interruption sur la broche RB0

                                ; Restaurer registres
                               ;-----------------------

            swapf status_temp,w ;       Swap ancien status, résultat dans w
            movwf STATUS ;              Restaurer status
            swapf w_temp, f ;           Inversion L et H de l'ancien W sans modifier
                            ;           Z
            swapf w_temp, w ;           Réinversion de L et H dans W, W restauré
                            ;           sans modifier status
            retfie ;                    return from interrupt

;*****************************************************************************
; INTERRUPTION RB0/INT
;*****************************************************************************
; Inverse le niveau de RA2 à chaque passage, interdit toute nouvelle interruption
; et valide le flag tempo
;*****************************************************************************
intrb0 :    movlw B'00000100' ;         bit positionné = bit inversé
            BANK0 ;                     se placer en bank0
            xorwf PORTA f ;             inverser RA2
            bcf INTCON, INTF ;          effacer flag INT/RB0
            bcf INTCON, INTE ;          interdire autre inter. RB0
            bsf tempoF ;                positionner flag tempo
```

```
                return ;                fin d'interruption RB0/INT
;*****************************************************************************
; Initialisations
;*****************************************************************************
init :          clrf PORTA ;            Sorties portA à 0
                clrf PORTB ;            Sorties portB à 0
                clrf EEADR ;            Permet de diminuer la consommation
                BANK1 ;                 Passer banque1
                movlw OPTIONVAL ;       Charger masque
                movwf OPTION_REG ;      Initialiser registre option

                                        ; Effacer RAM
                                        ; ----------------
                movlw 0x0c ;            Initialisation pointeur
                movwf FSR ;             Pointeur d'adressage indirect

init1 :         clrf INDF ;             Effacer RAM
                incf FSR, f ;           Pointer sur suivant
                btfss FSR, 6 ;          Tester si fin zone atteinte (>=40)
                goto init1 ;            Si Non, boucler vers l'étiquette init1

                btfss FSR, 4 ;          Tester si fin zone atteinte (>=50)
                goto init1 ;            Si Non, boucler vers l'étiquette init1

                                        ; Configuer PORTS
                                        ; ---------------------

                bcf LED ;               RA2 en sortie (TRISA)
                BANK0 ;                 Passer banque0
                movlw INTERMASK ;       Masque interruption
                movwf INTCON ;          Charger interrupt control
                goto start ;            Sauter programme principal
;*****************************************************************************
; Sous-routine de temporisation
;*****************************************************************************
; Cette sous-routine introduit un retard
; Elle ne reçoit aucun paramètre et n'en retourne aucun
;*****************************************************************************
tempo :         clrf cmpt2 ;            effacer compteur2
boucle2 :       clrf cmpt1 ;            effacer compteur1
boucle1 :       decfsz cmpt1, f ;       décrémenter compteur1
                goto boucle1 ;          Si pas 0, boucler
                decfsz cmpt2, f ;       Si 0, décrémenter compteur 2
                goto boucle2 ;          Si cmpt2 pas 0, recommencer boucle1
                return ;                Retour de la sous-routine
;*****************************************************************************
; Programme principal
;*****************************************************************************
start :         btfss tempoF ;          Tester si tempo flag mis
```

```
goto start ;            Non, attendre qu'il soit mis
call tempo ;            Oui, exécuter tempo
bcf tempoF ;            Effacer flag tempo
bcf INTCON, INTF ;      Effacer flag INT
bsf INTCON, INTE ;      Remettre interrupts INT en service
goto start ;            Boucler
END ;                   Directive fin de programme
;*****************************************************************************
```

Exemple

Programme qui permet clignotement d'une LED à l'aide du **chien de garde**.




```
;*****************************************************************************
;Ce programme de clignotement d'une LED à l'aide du chien de garde
;*****************************************************************************

LIST P=16f84 ;                  Directive qui définit le processeur utilisé
f =inhx8m ;
#include <P16F84A.INC> ;        Fichier de définition des constantes

;*****************************************************************************
; Bits de configuration
;*****************************************************************************
__config B'11111111110111' ;
;*****************************************************************************
            bsf STATUS, RP0 ;
            movlw B'00001101' ;
            movwf OPTION_REG ;
            movlw B'11111110' ;
            movwf TRISB ;
            bcf STATUS, RP0 ;
Boucle :    sleep ;
            comf PORTB, 1 ;
```

```
                goto Boucle ;
                end.
;*****************************************************************************
```

Exemple

Programme qui permet clignotement d'une LED à l'aide **des interruptions du TIMER 0**.

```
;*****************************************************************************
;Le programme de clignotement d'une LED à l'aide des interruptions du TIMER 0
;*****************************************************************************
LIST P=16f84 ;                  Directive qui définit le processeur utilisé
f =inhx8m ;
#include <P16F84A.INC> ;        Fichier de définition des constantes

;*****************************************************************************
; Bits de configuration
;*****************************************************************************
__config B'11111111110001' ;
;*****************************************************************************

                temps       equ   H'0C'

                org H'00' ;
                goto Debut ;
;*****************************************************************************
; Routine  de l'interruption du TIMER 0
;*****************************************************************************
                org H'04' ;
                movlw D'012' ;
                movwf TMR0 ;                Autoriser le mode TMR0
                bcf INTCON, T0IF ;
                decfsz temps, 1 ;
                retfie ;
                comf PORTB, 1 ;
                movlw D'008' ;
                movwf temps ;
                retfie ;
;*****************************************************************************
; Programme principal
;*****************************************************************************
Debut :         bsf STATUS, RP0 ;
                movlw B'10000111' ;
                movwf OPTION_REG ;
                bcf TRISB, 0 ;
```

```
                bcf STATUS, RP0 ;
                movlw D'008' ;
                movwf temps ;
                movlw B'10100000' ;
                movwf INTCON ;
Boucle :        goto Boucle ;
                end.
;*****************************************************************************
```

4.3.4. Lecture-Ecriture dans la mémoire EEPROM

La structure technologique de la mémoire EEPROM (mémoire dedonnées) du PIC 16F84 autorise l'accès en lecture comme en écriture selon deux procédures spécifiques.

4.3.4.1. Accès en lecture dans la mémoire EEPROM

Comment peut-on **LIRE une donnée** de la mémoire **EEPROM** de données ?

1. Placer l'adresse relative dans **EEADR**.
2. Mettre le bit **RD** de **EECON1** à **'1'**.
3. Lire le contenu du registre **EEDATA**.

Exemple

Lecture de l'emplacement mémoire situé à l'adresse 0x10.


```
;*****************************************************************************
bcf STATUS, RP0 ;       Passage en banque 0
movlw 0x10 ;
movwf EEADR ;           0x10 est l'adresse de l'emplacement mémoire
bsf STATUS, RP0 ;       Passage en banque 1
bsf EECON1, RD ;        Lecture de l'EEPROM
bcf STATUS, RP0 ;       Passage en banque 0
movf EEDATA, W ;        La valeur lue dans l'EEPROM est placée dans
                        l'accumulateur
;*****************************************************************************
```

*. Ne pas oublier les changements de banques.

4.3.4.1. Accès en ecriture dans la mémoire EEPROM

Comment peut-on **ECRIRE une donnée** sur la mémoire **EEPROM** de données ?

1. L'écriture dans L'EEPROM doit être autorisée : **WREN = '1'**.
2. Placer l'adresse relative dans **EEADR**.
3. Placer la donnée à écrire dans **EEDATA**.
4. Placer **55H** dans **EECON2**.
5. Placer **AAH** dans **EECON2**.
6. Démarrer l'écriture en positionnant le bit **WR**.
7. Attendre la fin de l'écriture, (10 ms) (**EEIF = '1'** ou **WR = '0'**).
8. Recommencer au point 2, si on a d'autres données à écrire.

Exemple

Ecriture de la donnée 0xE3 dans l'emplacement mémoire situé à l'adresse 0x10

```
;*********************************************************************************
bcf STATUS, RP0 ;       Passage en banque 0
movlw 0x10 ;
movwf EEADR ;           0x10 est l'adresse de l'emplacement mémoire
movlw 0xE3 ;
movwf EEDATA ;          0xE3 est la donnée à écrire dans l'emplacement mémoire
bsf STATUS, RP0 ;       Passage en banque 1
bcf INTCON, GIE ;       Désactivation de toutes les interruptions (recommandation de Microchip)
bsf EECON1, WREN ;      Autorisation de l'écriture
movlw 0x55 ;            Séquence spécifique (c'est comme ça, il faut le savoir)
movwf EECON2 ;          Séquence spécifique
movlw 0xAA ;            Séquence spécifique
movwf EECON2 ;          Séquence spécifique
bsf EECON1, WR ;        Lance une opération d'écriture
bsf INTCON, GIE ;       Autorisation de toutes les interruptions
bcf EECON1, WREN ;      Interdiction de l'écriture
;*********************************************************************************
```

Info

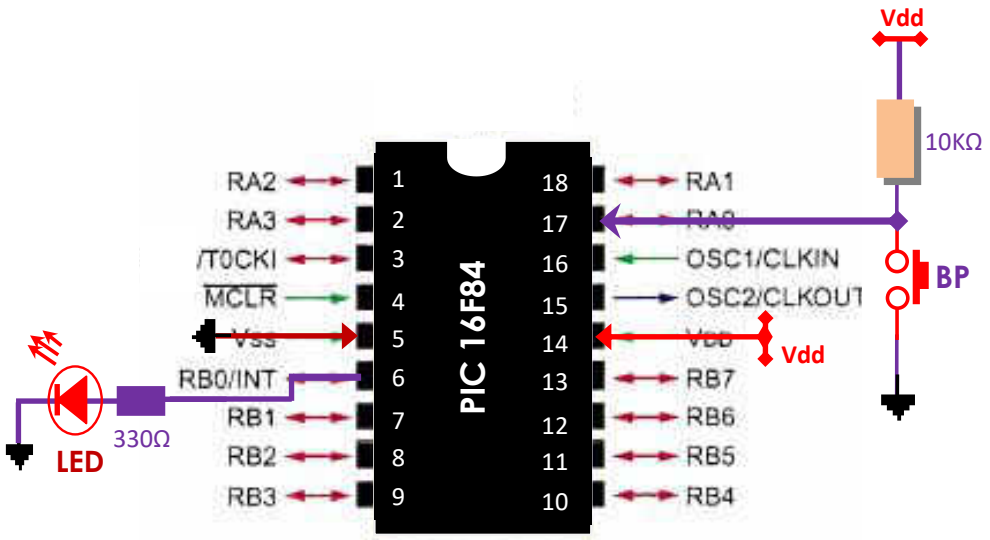
- Microchip recommande que cette procédure spécifique ne soit pas interrompue par une interruption.
- Avant d'écrire sur l'EEPROM, Il faut toujours vérifier qu'une autre écriture n'est en cours.
- A la fin de la procédure d'écriture, la donnée sera enregistrée dans l'EEPROM approximativement 10ms plus tard.
- La fin de l'écriture peut être constatée par:
  - ✓ La génération d'une interruption (si le bit **EEIE** est positionné)
  - ✓ La lecture du flag **EEIF** (s'il avait été remis à 0 avant l'écriture)
  - ✓ La consultation du bit **WR** qui est à 1 durant tout le cycle d'écriture.
- Le nombre de cycle d'écritures en EEPROM est limité environ 10 millions de cycles.
- L'écriture complète des 64 octets de l'EEPROM demande environ 1/4 de seconde !

EXERCICE 4.1

Ecrire un programme en langage assembleur permettant de simuler le fonctionnement d'un télerupteur :

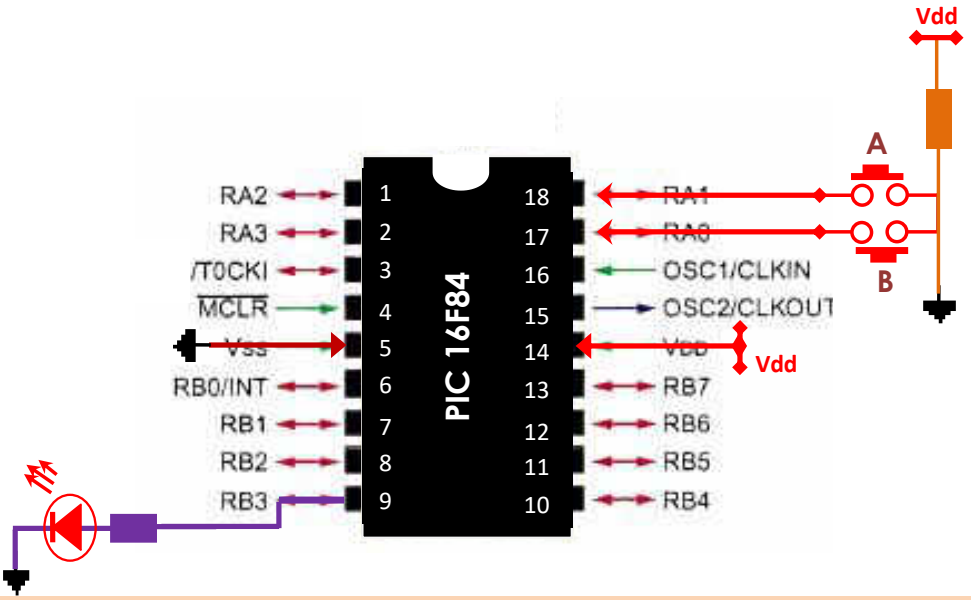
- ✓ L'action sur le bouton poussoir BP allume la LED.
- ✓ La nouvelle action sur BP éteint la LED.

On note que la résolution du problème des rebondissements du bouton poussoir **BP** consiste à l'utilisation d'une routine de temporisation.



EXERCICE 4.2

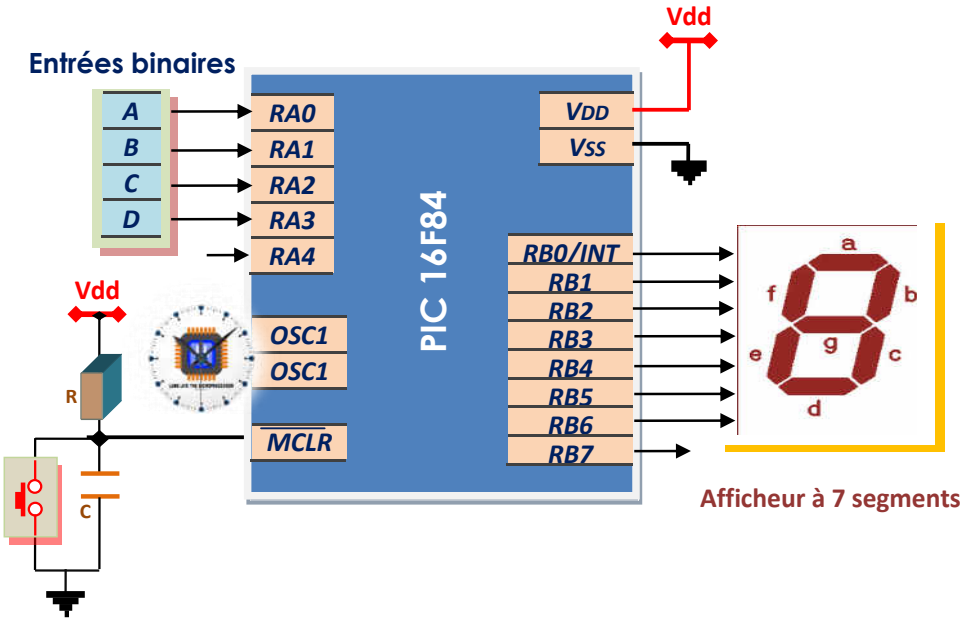
Ecrire un programme assembleur permettant de réaliser la fonction logique : **S=A⊕B** et d'afficher le résultat sur la LED (broche RB3).



EXERCICE 4.3

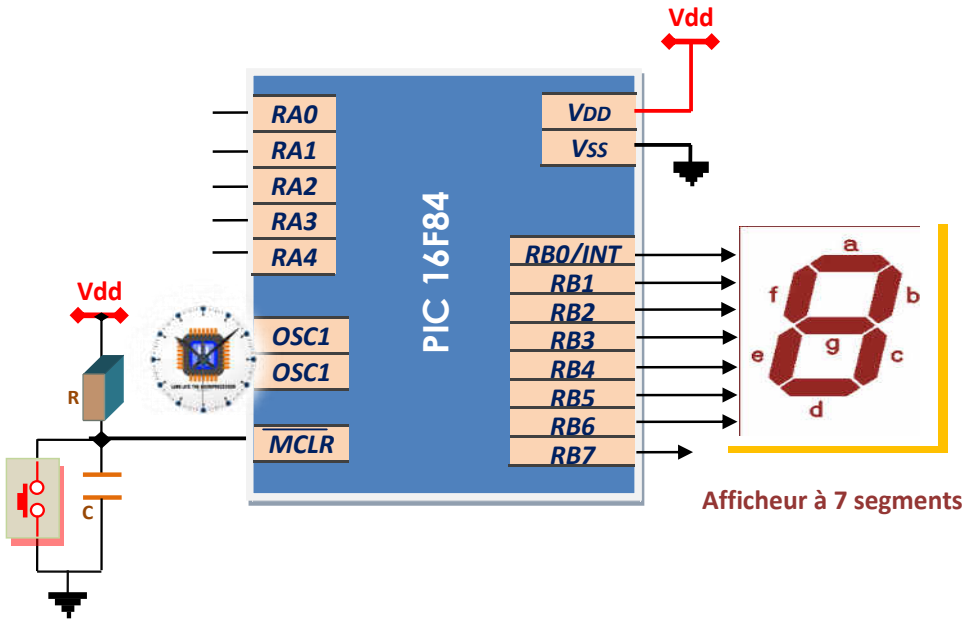
Ecrire un programme assembleur qui permet de générer le code BCD du mot binaire présent à l'entrée du port A. Ce code est disponible en sortie du port B pour être affiché sur un afficheur 7 segments.

On aura le soin de dresser un tableau d'équivalence entre le mot binaire d'entrée et le code BCD associé de sortie.



EXERCICE 4.4

Ecrire un programme assembleur permettant de concevoir un compteur BCD modulo 10.

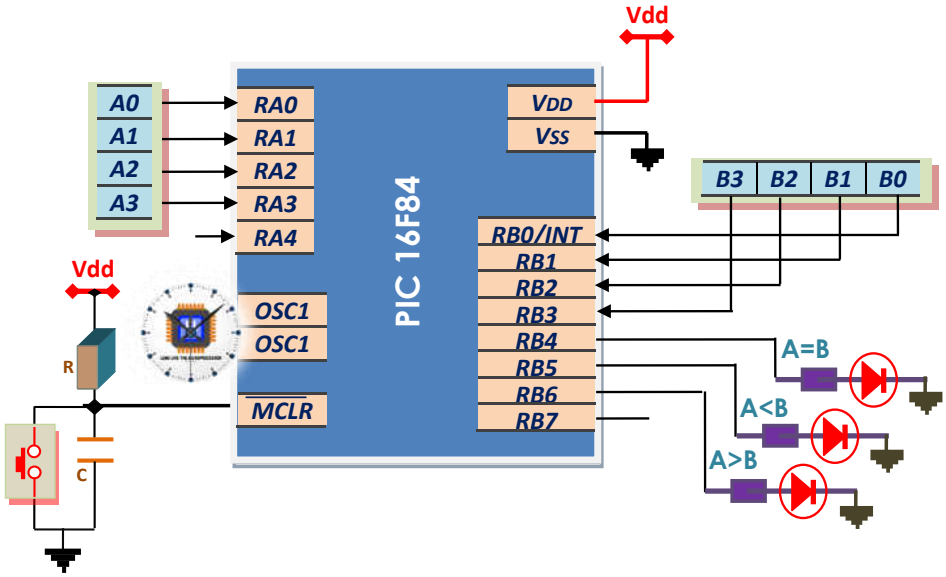


EXERCICE 4.5

Ecrire un programme assembleur qui permet de réaliser un comparateur de deux mots binaires de 4 bits :

Pour faire une comparaison entre F et W, on réalise la soustraction (F-W) et on teste les bits C et Z du registre STATUS :

- **Z** = 1 ⇒ F = W.
- **C** = 1 ⇒ F >= W.
- **C** = 0 ⇒ F < W.



EXERCICE 4.6

❶ On veut multiplie par additions successives le contenu du registre 20h par le contenu du registre 21h. Le résultat sera stocké dans le registre 22h. (On suppose que le résultat est donné sur un seul octet).

$$A*B = \underbrace{A + A + \ldots + A}_{B \text{ fois}}$$

✓ Donner l'organigramme pour réaliser ces opérations.

✓ Ecrire le programme correspondant en assembleur pour PIC.

❶ Compléter le programme suivant:

```
        org 0X04                ; .....
Comp :  .....                   ; Charger 5 dans w
        .....                   ; XOR w avec le registre 0F
        .....                   ; Tester le bit z du registre d'état
        .....                   ; Aller à Comp
        .....                   ; Retour de l'interruption
```

EXERCICE 4.7

Soit le programme suivant écrit en assembleur pour MPLAB:

```
__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF

N1 EQU 0x0D
N2 EQU 0x0E
N3 EQU 0x0F

        bsf STATUS, RP0 ;
        clrf TRISB ;
        bcf STATUS, RP0 ;
loop:   comf PORTB, f ;
        movlw 3 ;
        call tempo ;
        goto loop ;
tempo:  movwf N3 ;
tmp :   decfsz N1, f ;
        goto tmp ;
        decfsz N2, f ;
        goto tmp ;
        decfsz N3, f ;
        goto tmp ;
        return ;
```

✓ Commenter le programme instruction par instruction.

✓ Donner l'organigramme correspondant.

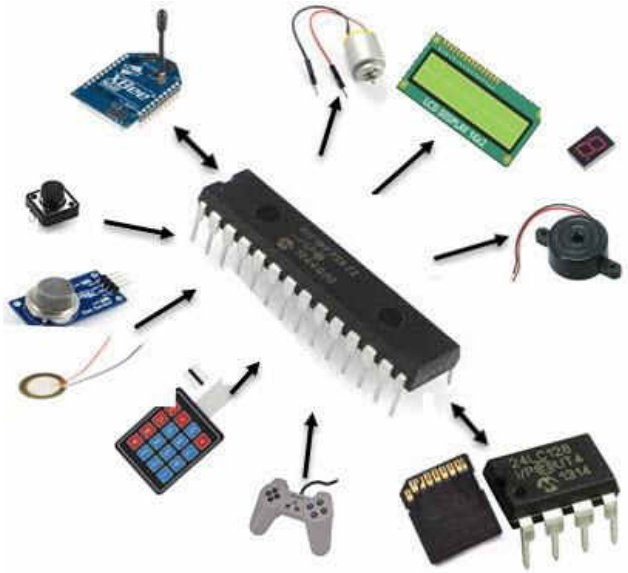
✓ Quelle est la tache réalisée par ce programme.

EXERCICE 4.8

Soit le programme suivant écrit en assembleur pour MPLAB:

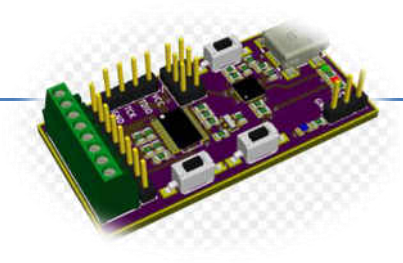
```
LIST p=16f84, f=inhx8m, r=dec
__CONFIG 0x3FF5
#INCLUDE "p16f84.inc"

        ORG 0x00 ;
        goto Main ;

            ORG 0x04 ;
            bcf INTCON, T0IF ;
            clrwdt ;
            retfie ;

Main:       bsf STATUS, RP0 ;
            clrf TRISB ;
            movlw B'00110100' ;
            movwf OPTION_REG ;
            bcf STATUS, RP0 ;
            movlw B'10100000' ;
            movwf INTCON ;
            clrf PORTB ;
            clrf TMR0 ;
Loop:       movf TMR0, w ;
            movwf PORTB ;
            goto Loop ;
            end ;
```

- ✓ Commenter le programme instruction par instruction.
- ✓ Donner l'organigramme correspondant.
- ✓ Quelle est la tache réalisée par ce programme.



CHAPITRE **5**

Système de déveoppement du PIC 16F84 - MPLAB -



CHAPITRE 5 Système de Développement du PIC 16F84 ~ MPLAB ~

Objectifs

L'objectif est de
- ✓ Présenter d'une manière profonde le système de développement MPLAB.
- ✓ Décrire les fonctionnalités et les concepts associés au MPLAB IDE v8.80.

Avant Propos

L'utilisation et la mise en oeuvre très simple des PICs les a rendus extrêmement populaire au point que la société qui les fabrique (**Microchip**) est en passe de devenir le leader mondial dans le domaine des microcontrôleurs devant MOTOROLA et INTEL.

Il est à noter que **Microchip** propose gratuitement (Téléchargeable à partir du l'URL : http://www.microchip.com/) l'outil de développement MPLAB qui regroupe
- ✓ **L'éditeur** de texte,
- ✓ Le **compilateur** MPASM,
- ✓ Un **outil de simulation** et
- ✓ Le **logiciel de programmation**.

Le programmateur lui-même, n'est malheureusement pas gratuit.

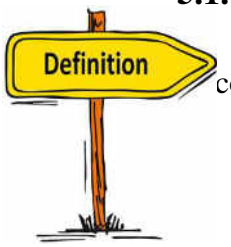
Figure 5.1 : *Kit de développement (MPLAB + Programmateur des PIC).*

Les outils de développement **Microchip** sont une suite de logiciels et matériels permettant de concevoir des applications à base des composants **Microchip** (www.microchip.com).

Microchip a un grand ensemble d'outils de développement logiciels et matériels intégrés au sein d'un logiciel appelé **MPLAB-IDE** (Integrated Development Environment). D'autres sociétés fournissent également des outils qui fonctionnent dans ce cadre.

- Avec l'environnement de **MPLAB** il est possible de réaliser un fichier source en langage assembleur (fichier **.asm**).
- L'avantage de **MPLAB** c'est de réaliser des programmes en **langage C**.
- MPLAB peut utiliser un langage de programmation évolué pour le développement en électronique.
- Après avoir réalisé le programme d'un fichier source en assembleur ou en c, il est possible de transformer ce dernier en fichier .hex. Ça le rend prêt à être chargé dans le microcontrôleur.

5.1. Environnements de Développement Intégrés (IDE)



L'IDE (Integrated Development Environment) est une interface qui permet de développer, compiler et exécuter un programme dans un langage donné.

- La plupart des langages de programmation (Java, Langage C, …etc.) sont associés à un outil permettant de
  - ✓ Saisir du code,
  - ✓ Compiler, Débugger et
  - ✓ Exécuter des programmes.
- Cet outil combine les fonctionnalités d'un éditeur de texte, d'un compilateur et d'un débugger. Il rend plus pratique la programmation, peut disposer de fonctions de complétion automatique de textes pour accélérer la saisie. Il dispose en général d'une documentation sur les outils et méthodes du langage concerné et facilite l'accès aux propriétés des librairies communes.
- Les IDE sont souvent puissants et très efficaces mais gourmands en ressources système.
- Exemples d'IDE :
  - ✓ Microsoft .NET pour C, C++, C#.
  - ✓ NetBeans pour Java.

- Les meilleurs environnements de développement Intégrés :
  - **MPLAB (Microchip)**
  - **Scilab**
  - **Visual Studio**
  - **Visual Studio Express Edition**
  - **Eclipse**
  - **NetBeans**
  - **Delphi**

- Code::Blocks
- MonoDevelop
- SharpDevelop
- KDevelop
- Access
- C++ Builder
- MATLAB
- LabVIEW
- Dev-C++
- Qt Creator
- XCode
- Lazarus
- WinDev
- 4D
- Dreamweaver
- Zend Studio

Source : http://general.developpez.com/edi/

Source : http://general.developpez.com/edi/

- L'avantage de **MPLAB** c'est de réaliser des programmes en **langage C**.
- MPLAB peut utiliser un langage de programmation évolué pour le développement en électronique.
- Après avoir réalisé le programme d'un fichier source en assembleur ou en c, il est possible de transformer ce dernier en fichier .hex. Ça le rend prêt à être chargé dans le microcontrôleur.

5.2. Présentation générale de l'environnement intégré de développement MPLAB

MPLAB est en environnement de développement (IDE) dédié aux microcontrôleurs PIC de Microchip (**www.microchip.com**). Il intègre :

- ✓ **Un éditeur** qui permet la création des fichiers de programme en langage C ou en assembleur.
- ✓ **Un compilateur** qui dépend de l'installation effectuée, ainsi que de la famille de microcontrôleurs PIC utilisée.
- ✓ **Un assembleur** qui dépend de l'installation effectuée, ainsi que de la famille de microcontrôleurs PIC utilisée.
- ✓ **Un linker** qui dépend de l'installation effectuée, ainsi que de la famille de microcontrôleurs PIC utilisée.
- ✓ **Un simulateur**.

Après avoir réalisé le programme d'un fichier source en assembleur ou en C, il est possible de transformer ce dernier en fichier **.hex**. Ça le rend prêt à être chargé dans le microcontrôleur par le biais d'un programmateur des PIC.

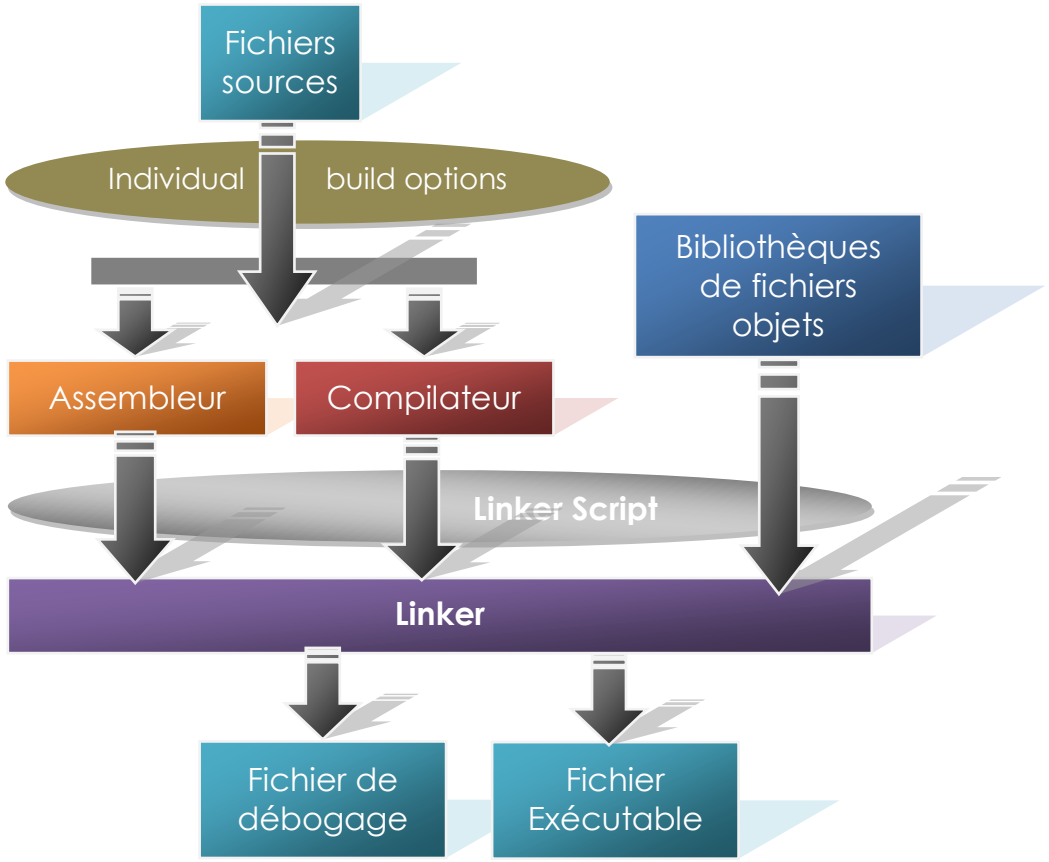


Figure 5.2 : *Gestion d'un projet MPLAB.*

La figure ci-dessous donne un aperçu de MPLAB-IDE et les catégories d'outils logiciels et matériels qui y sont intégrés.

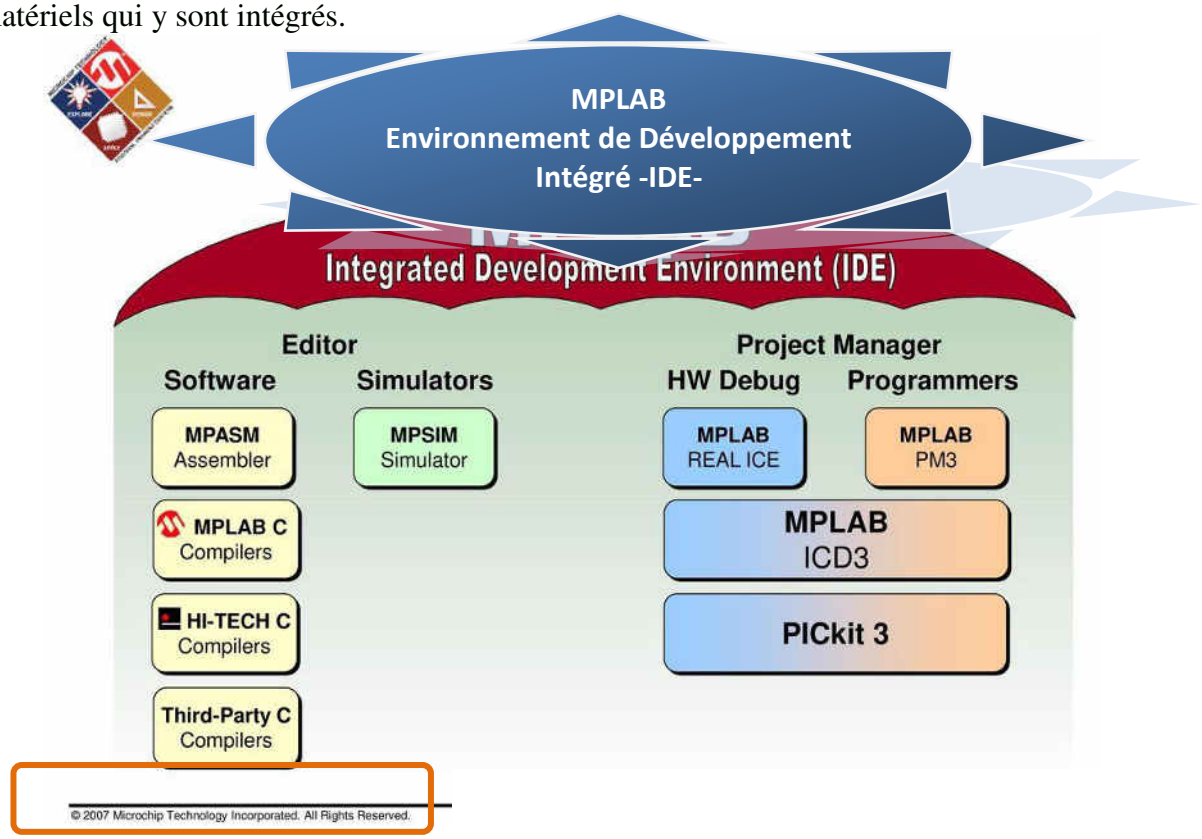


Figure 5.3 : *Aperçu de MPLAB-IDE.*

- **MPLAB ® IDE** - L'environnement de développement intégré sert d'application principale à partir de laquelle d'autres outils sont déployés.
- Le logiciel **MPLAB** est un outil de développement pour programmer des microcontrôleurs de type PIC de la famille **Microchip**. Il est mis au point par la société **Microchip**, et est entièrement gratuit téléchargeable à partir du site www.microchip.com.
- Ce logiciel permet de créer un programme, de l'assembler, et de le simuler.
- Enfin, le programme réalisé sous MPLAB peut être transférer pour le mettre sur votre PIC à travers un programmateur du PIC.

5.2.1. Versions de MPLAB

Ci-dessous différentes versions des systèmes de développement (IDE) MPLAB exécutables sur différents systèmes d'exploitation.

Windows (x86/x64)	Mac (10.X)	Linux (32/64 bit)
MPLAB IDE X v1.00a	MPLAB IDE X v1.00a	MPLAB IDE X v1.00a
MPLAB IDE X v1.10	MPLAB IDE X v1.10	MPLAB IDE X v1.10
MPLAB IDE X v1.20	MPLAB IDE X v1.20	MPLAB IDE X v1.20
MPLAB IDE X v1.30	MPLAB IDE X v1.30	MPLAB IDE X v1.30
MPLAB IDE X v1.41	MPLAB IDE X v1.41	MPLAB IDE X v1.41
MPLAB IDE X v1.51	MPLAB IDE X v1.51	MPLAB IDE X v1.51
MPLAB IDE X v1.60	MPLAB IDE X v1.60	MPLAB IDE X v1.60
MPLAB IDE X v1.70	MPLAB IDE X v1.70	MPLAB IDE X v1.70
MPLAB IDE X v1.80	MPLAB IDE X v1.80	MPLAB IDE X v1.80
MPLAB IDE X v1.85	MPLAB IDE X v1.85	MPLAB IDE X v1.85
MPLAB IDE X v1.90	MPLAB IDE X v1.90	MPLAB IDE X v1.90
MPLAB IDE X v1.95	MPLAB IDE X v1.95	MPLAB IDE X v1.95
MPLAB IDE X v2.00	MPLAB IDE X v2.00	MPLAB IDE X v2.00
MPLAB X v2.05	MPLAB X v2.05	MPLAB X v2.05
MPLAB X v2.10	MPLAB X v2.10	MPLAB X v2.10
MPLAB X v2.15	MPLAB X v2.15	MPLAB X v2.15
MPLAB X v2.20	MPLAB X v2.20	MPLAB X
MPLAB X v2.26	MPLAB X v2.26	MPLAB X
MPLAB X v2.30	MPLAB X v2.30	MPLAB X
MPLAB X v2.35	MPLAB X v2.35	MPLAB X
MPLAB X v3.00	MPLAB X v3.00	MPLAB X v
MPLAB X v3.05	MPLAB X v3.05	MPLAB X v3.05
MPLAB X v3.10	MPLAB X v3.10	MPLAB X v3.10

~~Source :~~
https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive
[06/04/2019]

MPLAB X v3.15	MPLAB X v3.15	MPLAB X v3.15
MPLAB X v3.20	MPLAB X v3.20	MPLAB X v3.20
MPLAB X v3.26	MPLAB X v3.26	MPLAB X v3.26
MPLAB X v3.30	MPLAB X v3.30	MPLAB X v3.30
MPLAB X v3.35	MPLAB X v3.35	MPLAB X v3.35
MPLAB X v3.40	MPLAB X v3.40	MPLAB X v3.40
MPLAB X v3.45	MPLAB X v3.45	MPLAB X v3.45
MPLAB X v3.50	MPLAB X v3.50	MPLAB X v3.50
MPLAB X v3.55	MPLAB X v3.55	MPLAB X v3.55
MPLAB X v3.61	MPLAB X v3.61	MPLAB X v3.61
MPLAB X v3.65	MPLAB X v3.65	MPLAB X v3.65
MPLAB X v4.01	MPLAB X v4.01	MPLAB X v4.01
MPLAB X v4.05	MPLAB X v4.05	MPLAB X v4.05
MPLAB X v4.10	MPLAB X v4.10	MPLAB X v4.10
MPLAB X v4.15	MPLAB X v4.15	MPLAB X v4.15
MPLAB X v4.20	MPLAB X v4.20	MPLAB X v4.20
MPLAB X v5.00	MPLAB X v5.00	MPLAB X v5.00
MPLAB X v5.05	MPLAB X v5.05	MPLAB X v5.05
MPLAB X v5.10	MPLAB X v5.10	MPLAB X v5.10

16-bit Windows	32-bit Windows	32-bit Windows
MPLAB IDE v5.40	MPLAB IDE v6.10	MPLAB IDE v7.00
MPLAB IDE v5.70.40	MPLAB IDE v6.20	MPLAB IDE v7.10
	MPLAB IDE v6.30	MPLAB IDE v7.10
	MPLAB IDE v6.40	MPLAB IDE v7.20
	MPLAB IDE v6.50	MPLAB IDE v7.30
	MPLAB IDE v6.60	MPLAB IDE v7.31
		MPLAB IDE v7.40
		MPLAB IDE v7.50
		MPLAB IDE v7.60
		MPLAB IDE v8.00
		MPLAB IDE v8.10
		MPLAB IDE v8.14
		MPLAB IDE v8.15

MPLAB IDE v8.20a

MPLAB IDE v8.30

MPLAB IDE v8.33

MPLAB IDE v8.36

MPLAB IDE v8.40

MPLAB IDE v8.43

MPLAB IDE v8.46

MPLAB IDE v8.50

MPLAB IDE v8.53

MPLAB IDE v8.56

MPLAB IDE v8.60

MPLAB IDE v8.63

MPLAB IDE v8.66

MPLAB IDE v8.70

MPLAB IDE v8.73a

MPLAB IDE v8.76

MPLAB IDE v8.80

MPLAB IDE v8.83

MPLAB IDE v8.84

MPLAB IDE v8.85

MPLAB IDE v8.86

MPLAB IDE v8.87

MPLAB IDE v8.88

MPLAB IDE v8.89

MPLAB IDE v8.90

MPLAB IDE v8.91

MPLAB IDE v8.92

5.2.2. MPLAB IDE 8.8

L'environnement de développement MPLAB, quelques soit la version utilisée, regroupe tous les outils nécessaires à la mise au point d'une application à base des microcontrôleurs du type PIC de Microchip :

- Editeur de texte interactif.
- Compilateur C (et assembleur)
- Simulateur.
- Debugger si on dispose de l'équipement nécessaire.

Deux phases sont nécessaires pour bien exploiter l'outil de développement MPLAB pour microcontrôleurs Microchip :

1. Création et configuration d'un projet.
2. Utilisation du simulateur.

~~Source :~~
http://ww1.microchip.com/downloads/en/devicedoc/51519a.pdf [04/05/2019]
https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive [04/05/2019]
https://www.lycee-ferry-versailles.fr:8081 [04/05/2019]

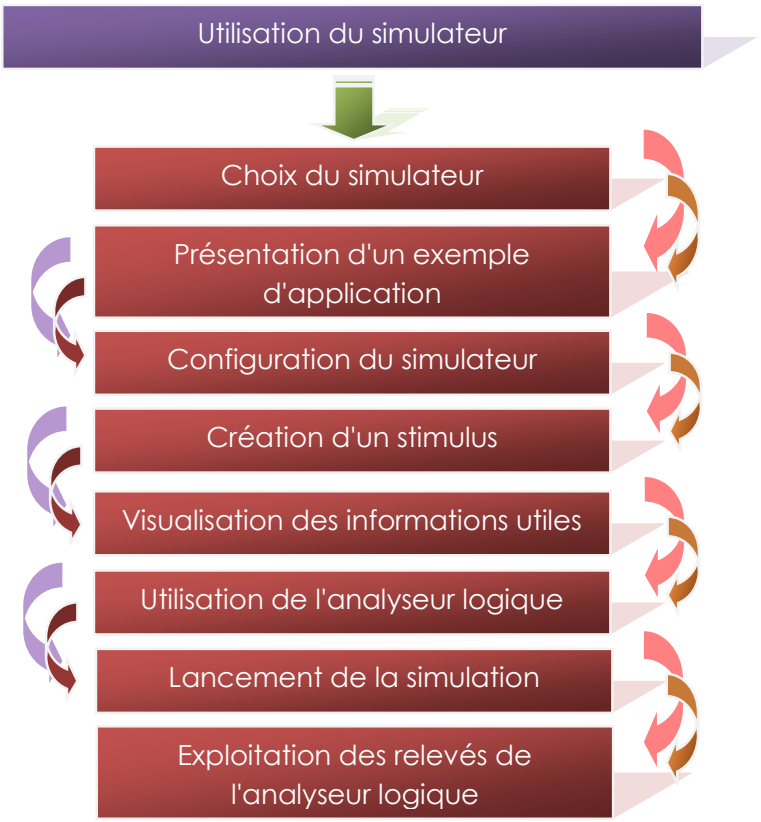
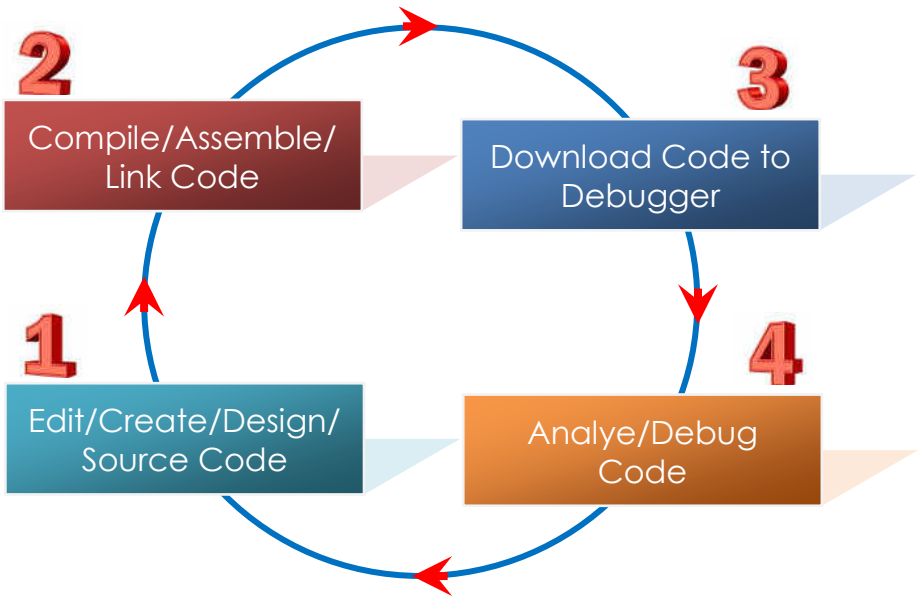
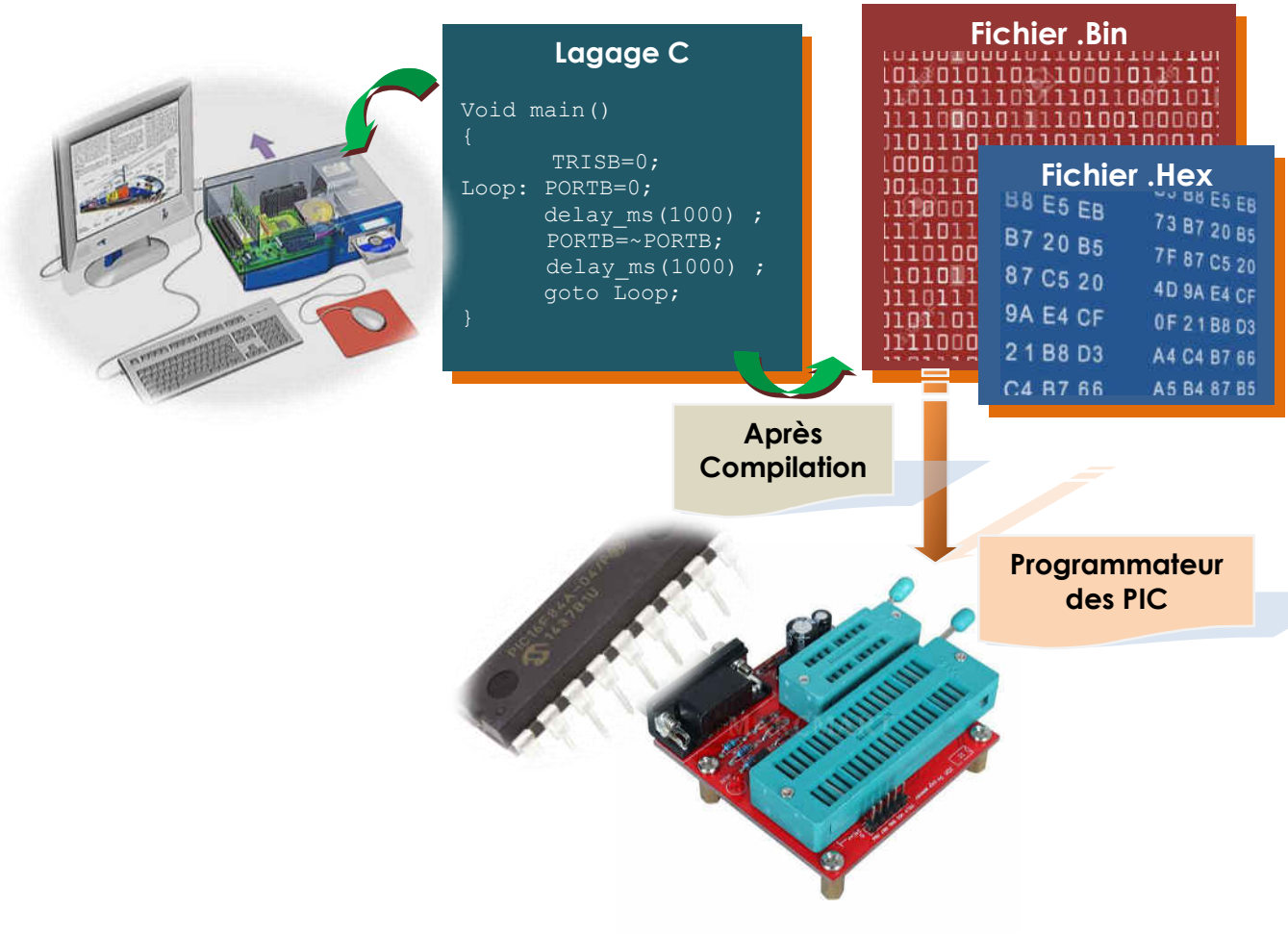


Figure 5.4 : *Phases d'exploitation d'outil de développement MPLAB pour microcontrôleurs Microchip.*

Après téléchargement à partir du lien https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive , l'outil de développement MPLAB IDE v8.80 sera prêt à être exécuté. Le cycle de développement est illustré par la figure ci-dessous.



(a)



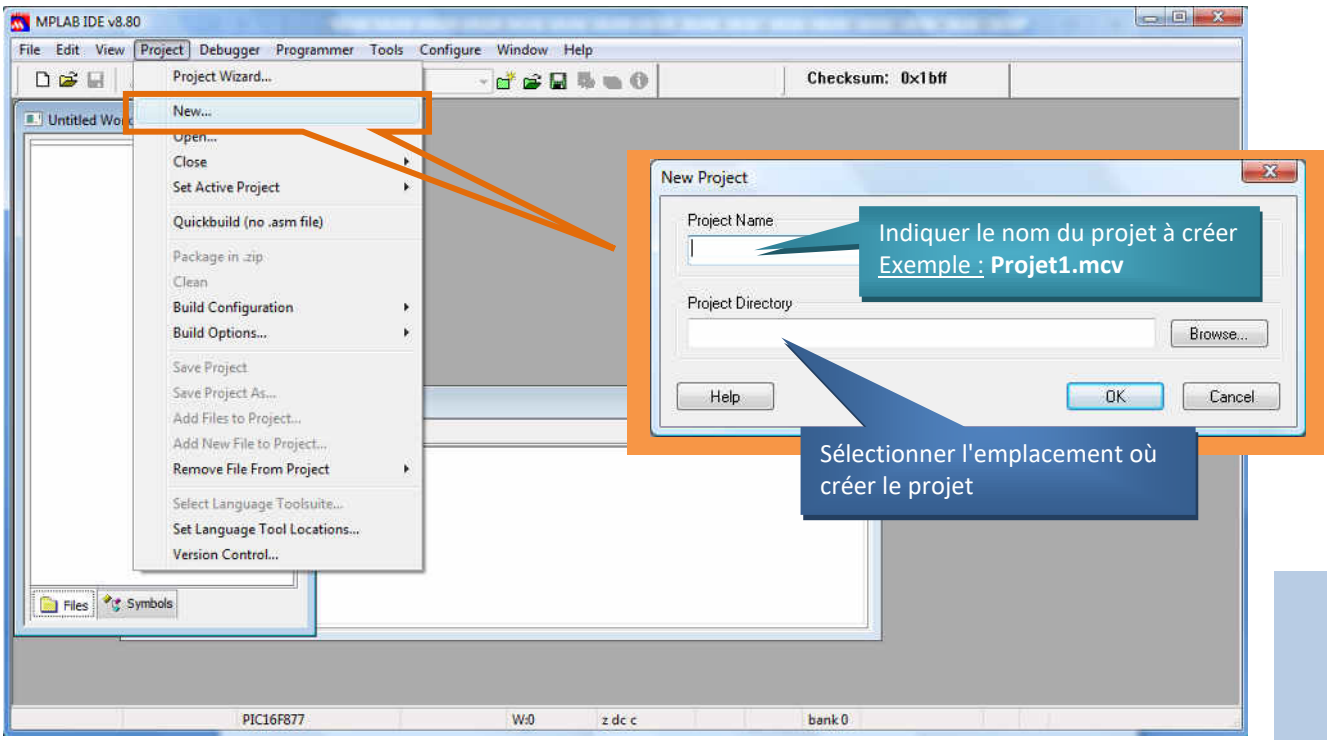
(b)

Figure 5.4 : *Cycle de développement.*

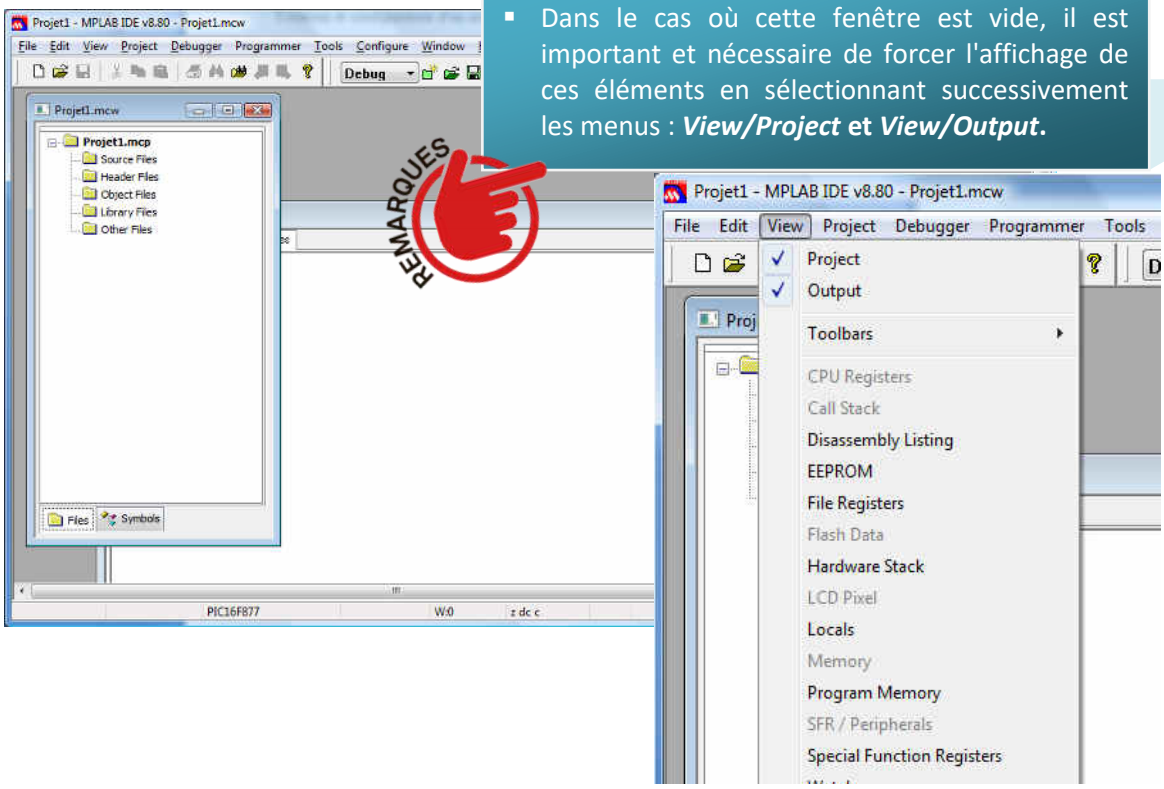
5.2.2.1. Création et configuration d'un nouveau projet

a. Création d'un nouveau projet

Après avoir lancé le programme (outil de développement MPLAB) déjà installé, la première étape est la création d'un nouveau projet. La procédure consiste à sélectionner le menu Project/New de la fenêtre principale du programme MPLAB IDE v8.80.



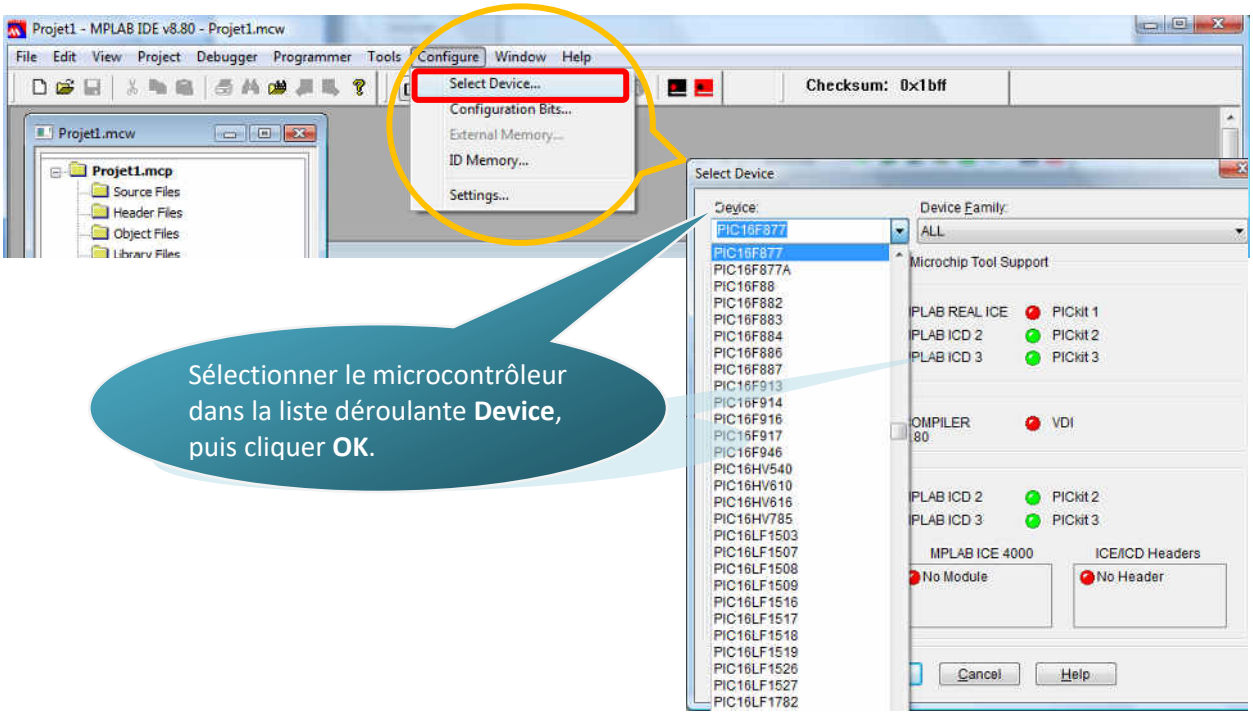
La fenêtre de travail est apparue comme suit :



- Dans le cas où cette fenêtre est vide, il est important et nécessaire de forcer l'affichage de ces éléments en sélectionnant successivement les menus : ***View/Project*** et ***View/Output***.

b. Définition du microcontrôleur cible

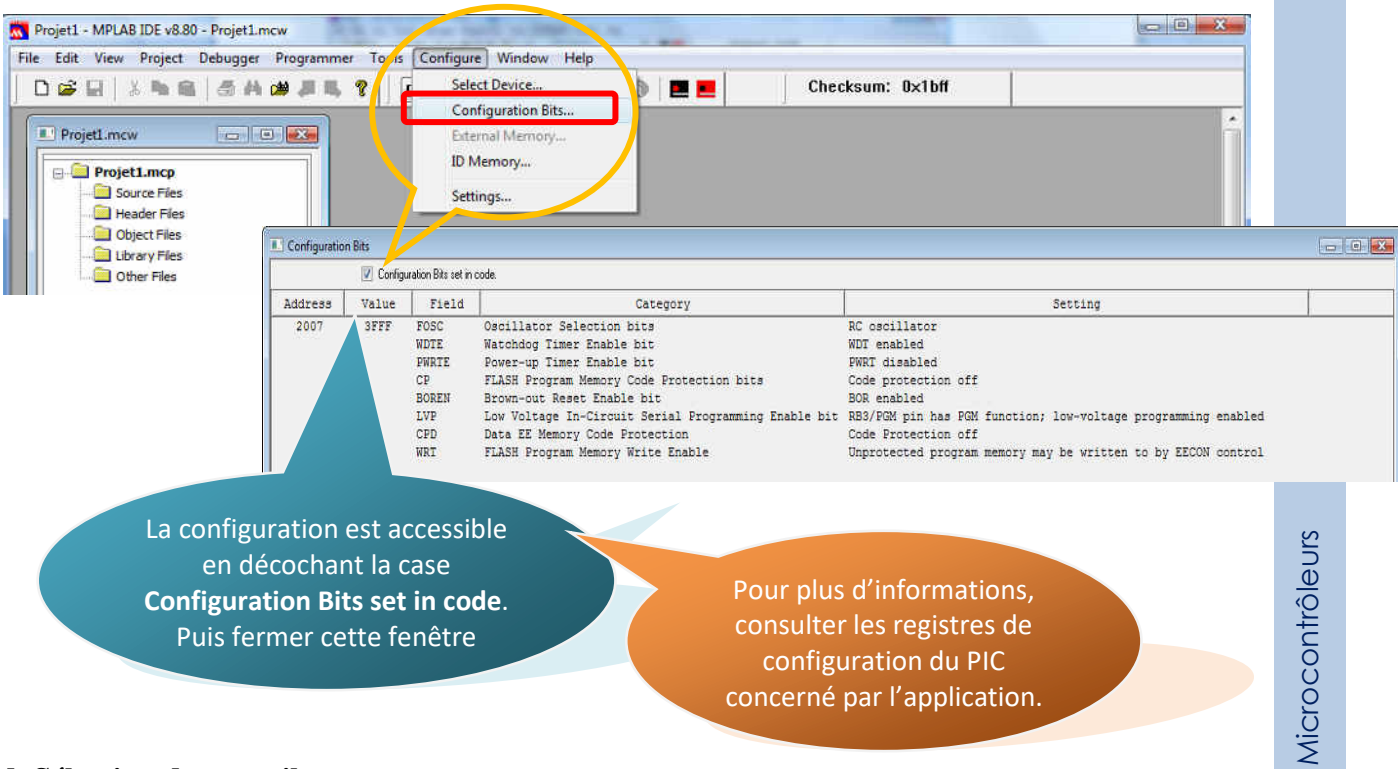
Sélectionner le menu **Configure/Select Device.**



c. Définition des bits de configuration

Cette opération est nécessaire, voir obligatoire, lors d'une application de programmation d'un composant réel.

Sélectionner le menu **Configure/Configuration Bits**.

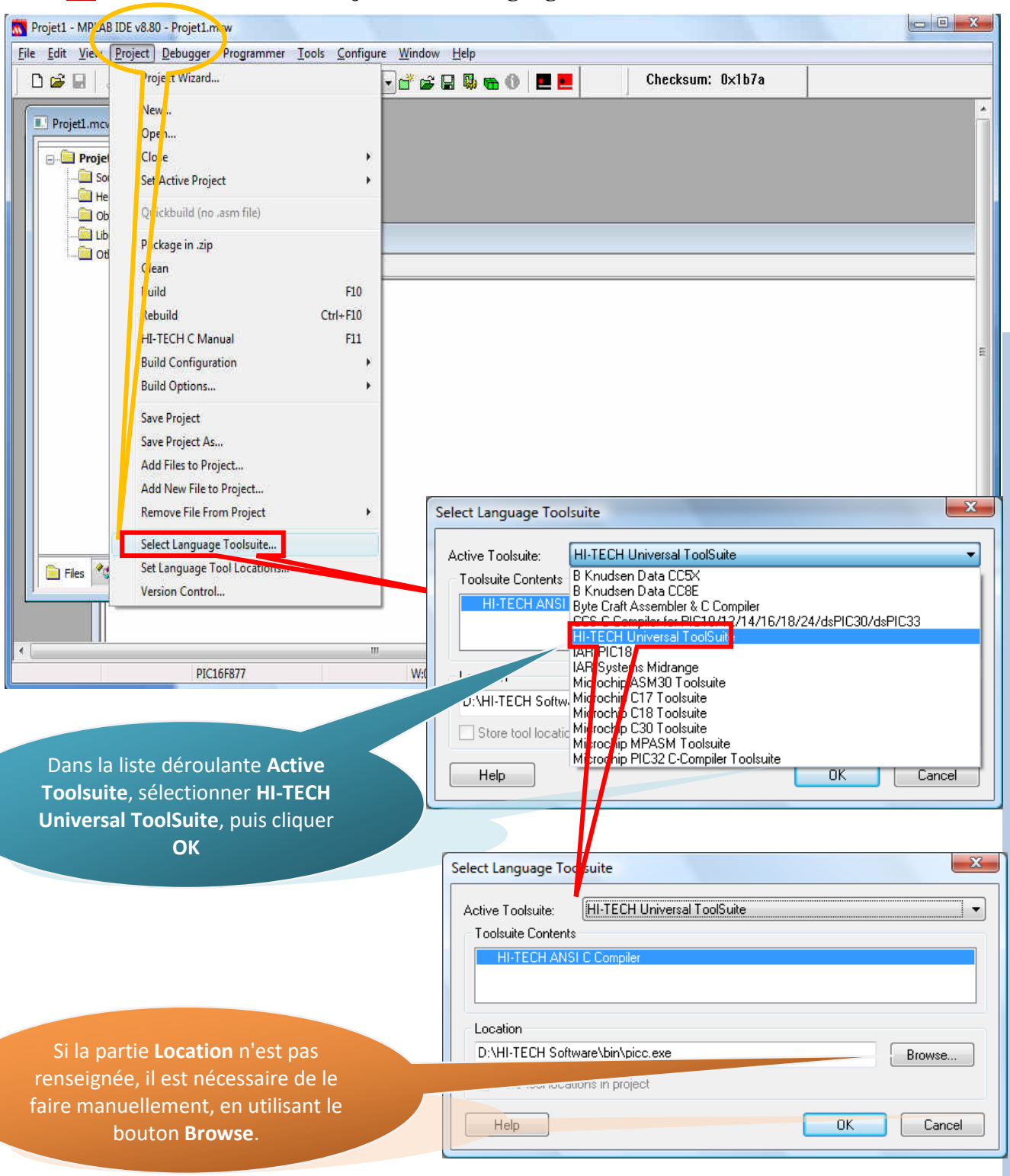


d. Sélection du compilateur

Plusieurs compilateurs sont disponibles lors de l'installation de MPLAB IDE v8.80 et ceci et suivant la famille de microcontrôleur PIC concerné par l'application en cours. A titre

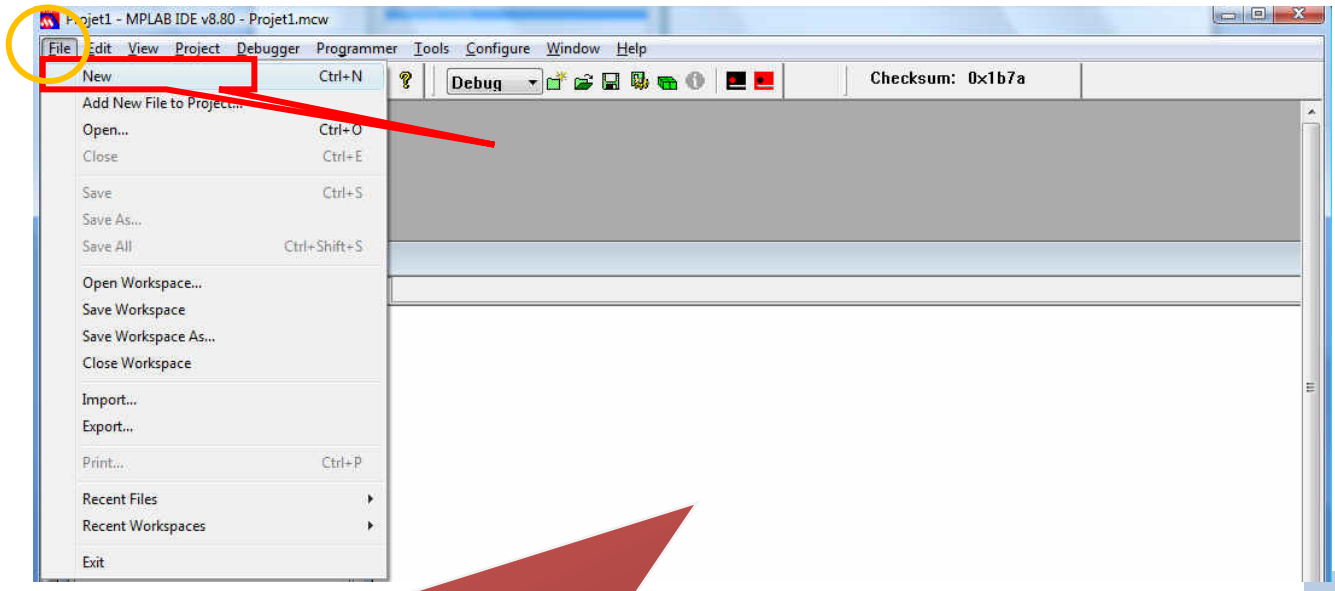
d'exemple, et pour le cas des PIC 16Fxxx, c'est un compilateur **HI-TECH est utilisé** (Version limitée, qui est utilisé.

- Sélectionner le menu **Project/Select Language Toolsuite**.



e. Création d'un fichier C

- Sélectionner le menu **File/New**



REMARQUES

- Une fenêtre d'édition apparaît, permettant de taper le code du fichier C.
- Après avoir créé le fichier C, on le sauvegarde dans le répertoire de travail.
- Par défaut, lorsqu'on sauvegarde un nouveau projet, l'extension qui lui est donnée est .c.
- Lorsqu'on travaille en assembleur, il faut spécifier l'extension .asm.

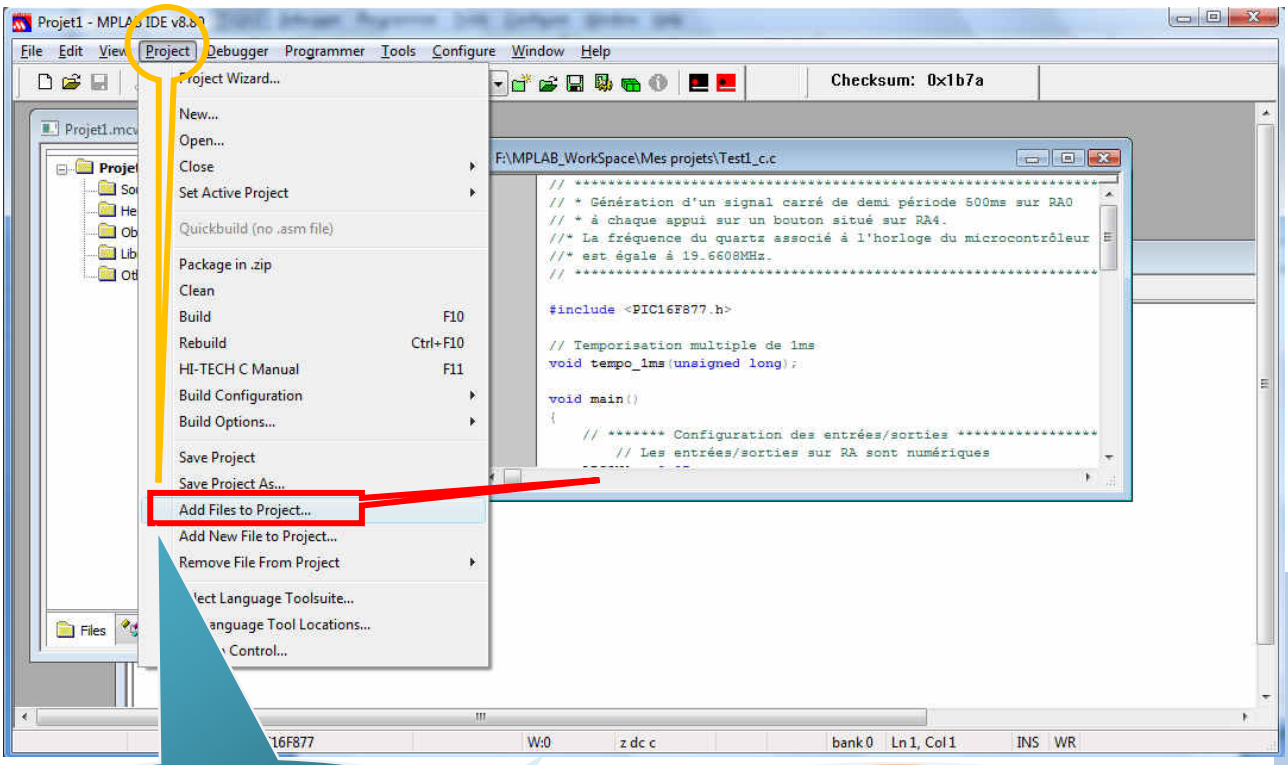
f. Association d'un fichier au projet

Info

- Dans un environnement de développement tel que MPLAB, on parle d'un projet et non un fichier (compilation, simulation).
- Un projet est constitué :
  - D'un ou plusieurs fichiers (en Assembleur et/ou en langage C) ;
  - Du choix du microcontrôleur cible (cœur de l'application);
  - De la définition des bits de configuration selon le mode d'utilisation du PIC;
  - Des informations diverses, telles que
    - Les options de compilation,
    - Les chemins des dossiers de travail,
    - De l'emplacement des outils d'assemblage,
    - De compilation,
    - D'édition de liens.

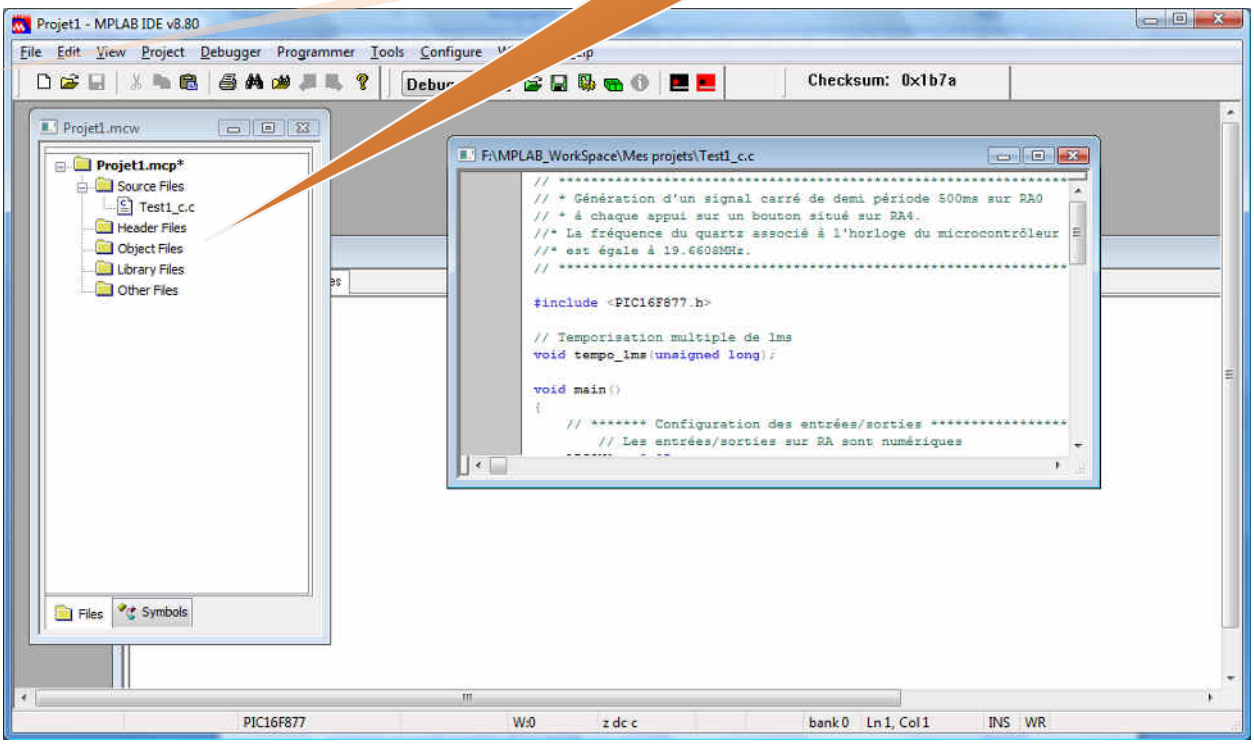
- Il est donc indispensable d'associer les fichiers créés au projet auquel ils sont associés.

L'association de fichiers à un projet est effectuée en sélectionnant le menu **Project/Add Files to Project.**



On sélectionne ensuite le ou les fichiers que l'on souhaite associer au projet (en langage C, ou en assembleur).

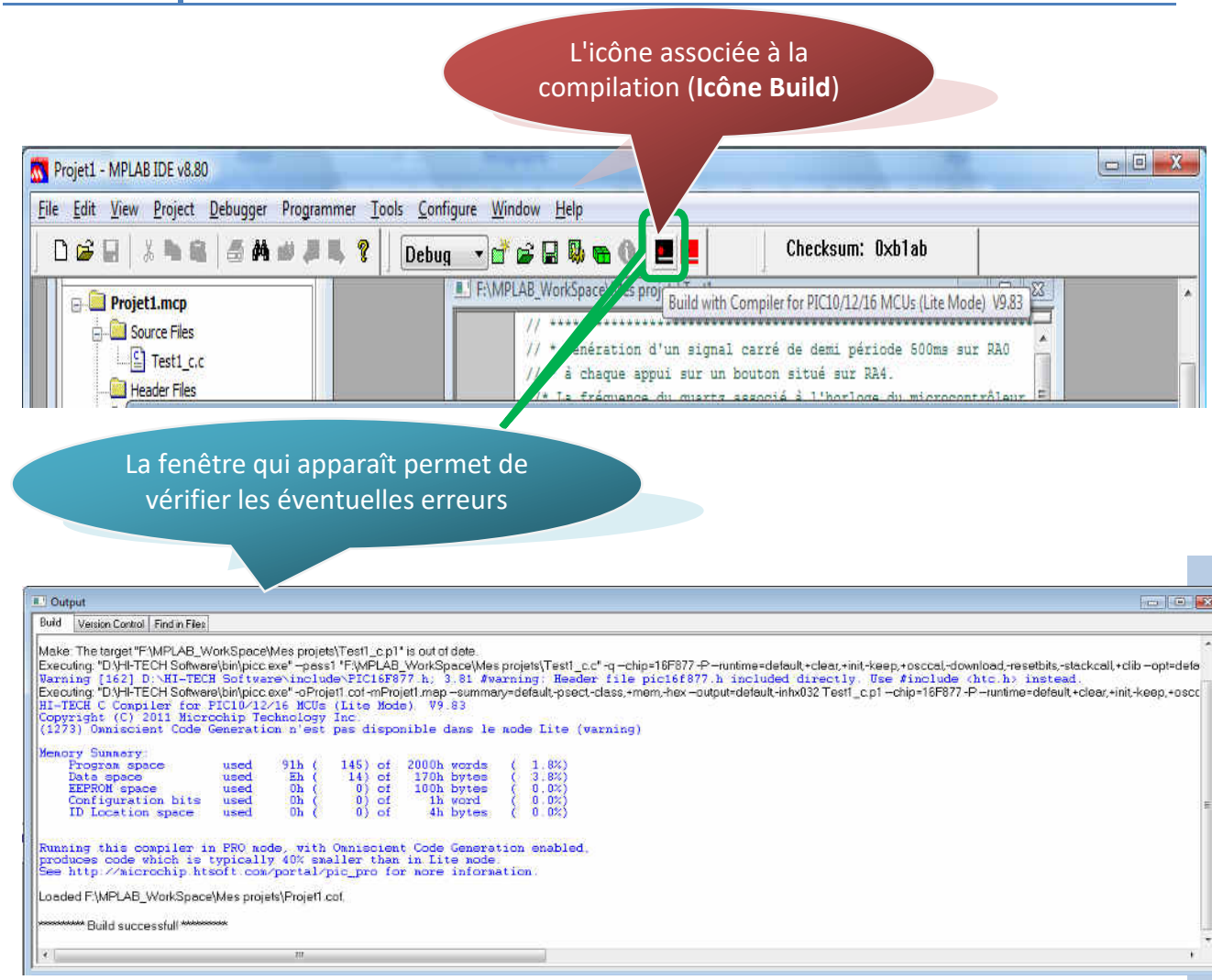
On retrouve alors le ou les fichiers choisis dans la fenêtre de projet



g. Compilation du projet

Sélectionner le menu **Projet/Build** afin de compiler le projet.

L'icône associée à la compilation (Icône **Build**) est la suivante :



5.2.2.2. Utilisation du simulateur

- Dans un environnement de développement tel que MPLAB, on parle d'un projet et non un fichier (compilation, simulation).
- Un projet est constitué :
  - D'un ou plusieurs fichiers (en Assembleur et/ou en langage C) ;
  - Du choix du microcontrôleur cible (cœur de l'application);
  - De la définition des bits de configuration selon le mode d'utilisation du PIC;
  - Des informations diverses, telles que
    - Les options de compilation,
    - Les chemins des dossiers de travail,
    - De l'emplacement des outils d'assemblage,
    - De compilation,
    - D'édition de liens.

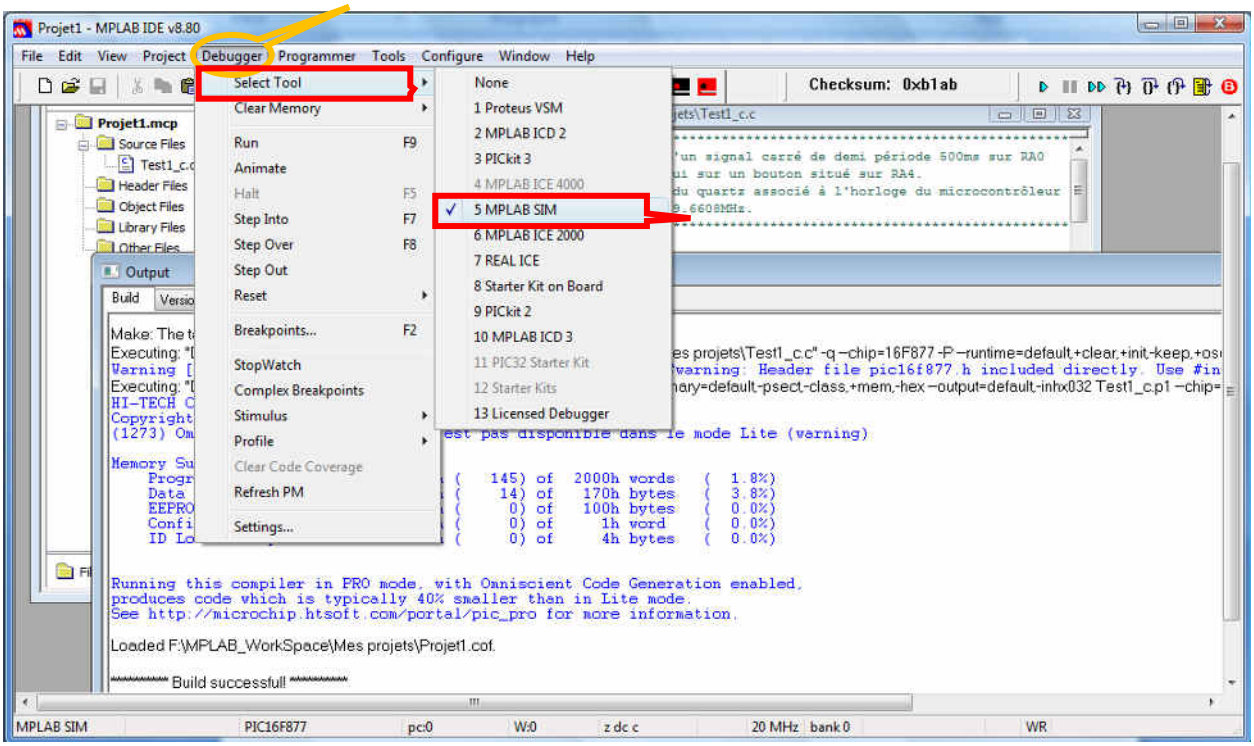
- Il est donc indispensable d'associer les fichiers créés au projet auquel ils sont associés.

Afin d'analyser le bon déroulement d'un programme, MPLAB offre un certain nombre d'outils permettant permettent d'assurer cette tâche, à savoir :

- Fenêtre de visualisation de variables et de registres choisis.
- Fenêtre de visualisation de l'ensemble des registres.
- Fenêtre de visualisation de la mémoire.
- Suivi de l'évolution de signaux à l'aide d'un analyseur logique.

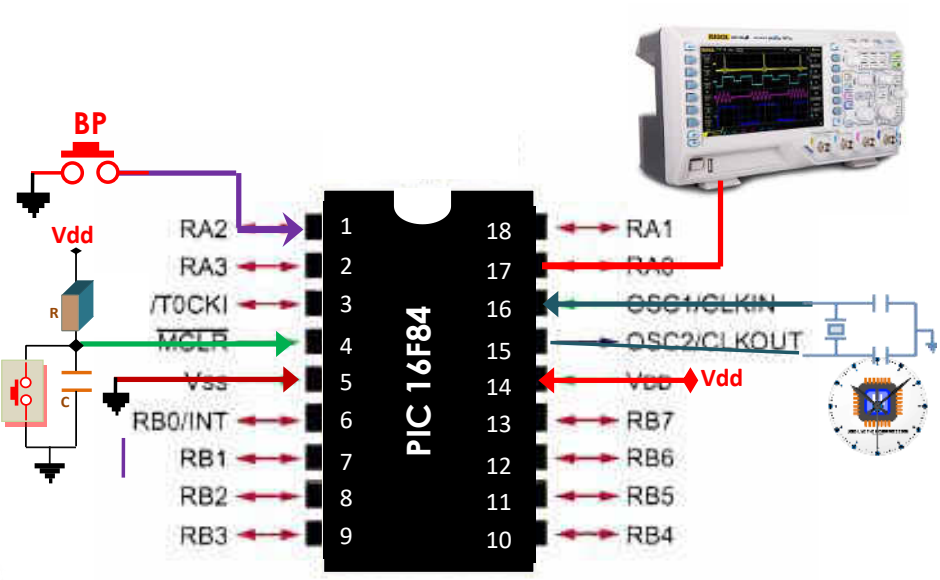
a. Choix du simulateur

- Pour effectuer l'opération de débogage, sélectionner le menu **Debugger/Select Tool** et choisir l'outil **MPLABSIM**.



b. Présentation d'un exemple d'application

- Le projet est constitué d'un fichier en langage C permettant de générer un signal carré de fréquence 50Hz sur la broche **RA0** d'un PIC.
- La génération de ce signal sera autorisée par un bouton poussoir associé à l'entrée **RA2** du PIC.



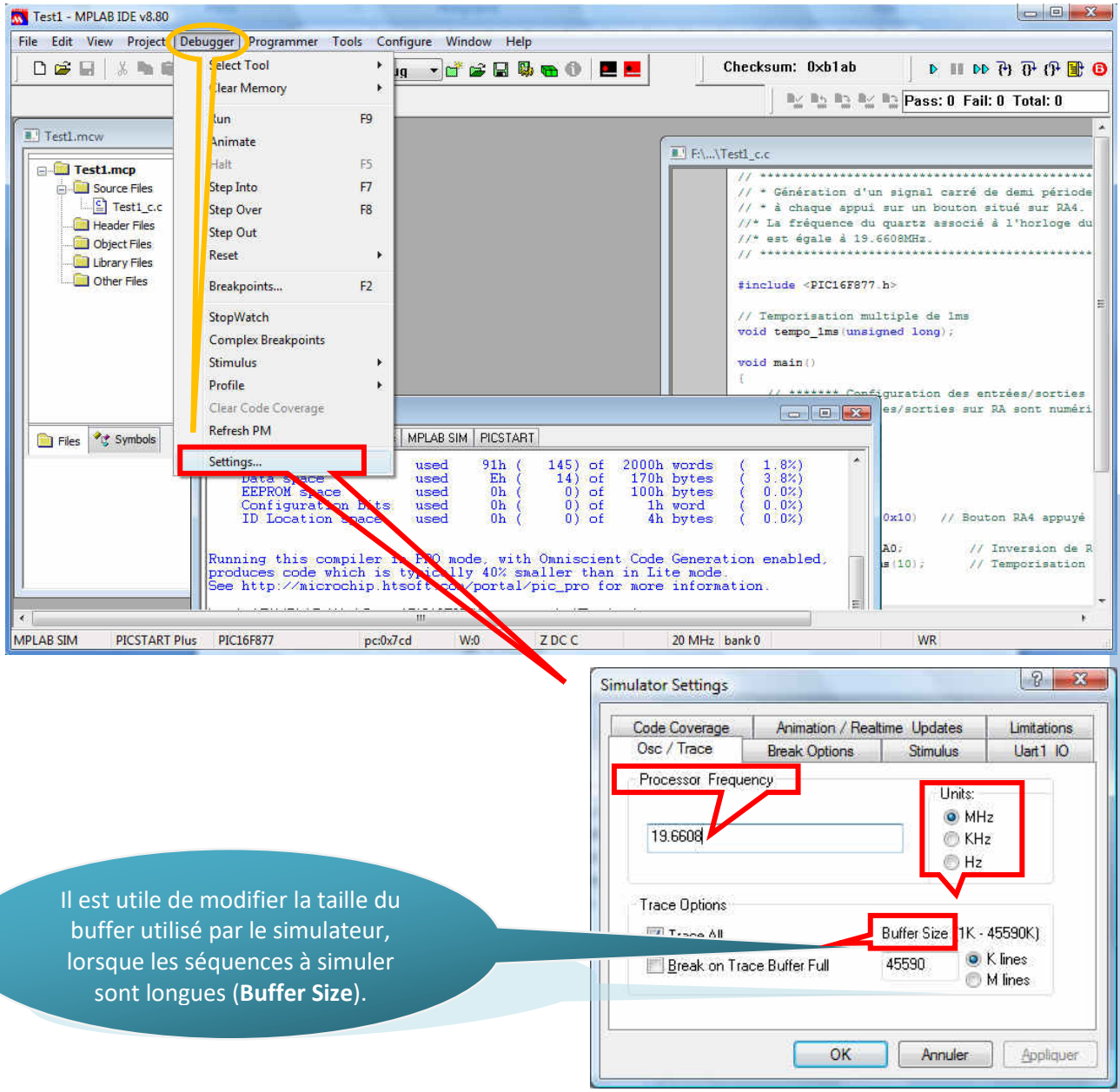

```
//*****************************************************************************
// Génération d'un signal carré de période 20ms sur la broche RA0 à chaque
appui sur un bouton poussoir connecté sur la broche RA3.
// La fréquence du quartz associé à l'horloge du microcontrôleur est égale
à 19.6608 MHz.
//*****************************************************************************

LIST P=16f84 ;                  // Directive qui définit le processeur utilisé
f =inhx8m ;
#include <PIC16F84.h> ;         // Fichier de définition des constantes

//*****************************************************************************
// Temporisation multiple de 1ms Bits de configuration
//*****************************************************************************
void tempo_1ms(unsigned long);
//*****************************************************************************
/*****************************************************************************
// Routine  de temoprisation
/*****************************************************************************
void tempo_1ms(unsigned long k)
{
        unsigned long i, j;
                for (i=0; i<k; i++)
                for (j=0; j<147; j++);
}
//*****************************************************************************
// Configuration des entrées/sorties
//*****************************************************************************

/*****************************************************************************
// Programme principal
/*****************************************************************************
```

```
void main()
        {
                // Configuration des entrées/sorties
                // Les entrées/sorties sur RA sont numériques
        ADCON1 = 0x07;
                // RA0 en sortie
                // RA3 en entrée
        TRISA = 0xFE;
        while(1)
                {
                if (PORTA & 0x10)               // Bouton poussoir sur RA3 appuyé
                        {
                        RA0 = !RA0;             // Inversion de RA0
                        tempo_1ms(10);          // Temporisation de 10ms
                        }
                }
        }
/*****************************************************************************
```

- Une fois le projet créé, le fichier précédent associé au projet et le projet compilé avec succès.
- On peut réaliser la simulation du fonctionnement et analyser la conformité avec la tâche demandée.

c. Configuration du simulateur

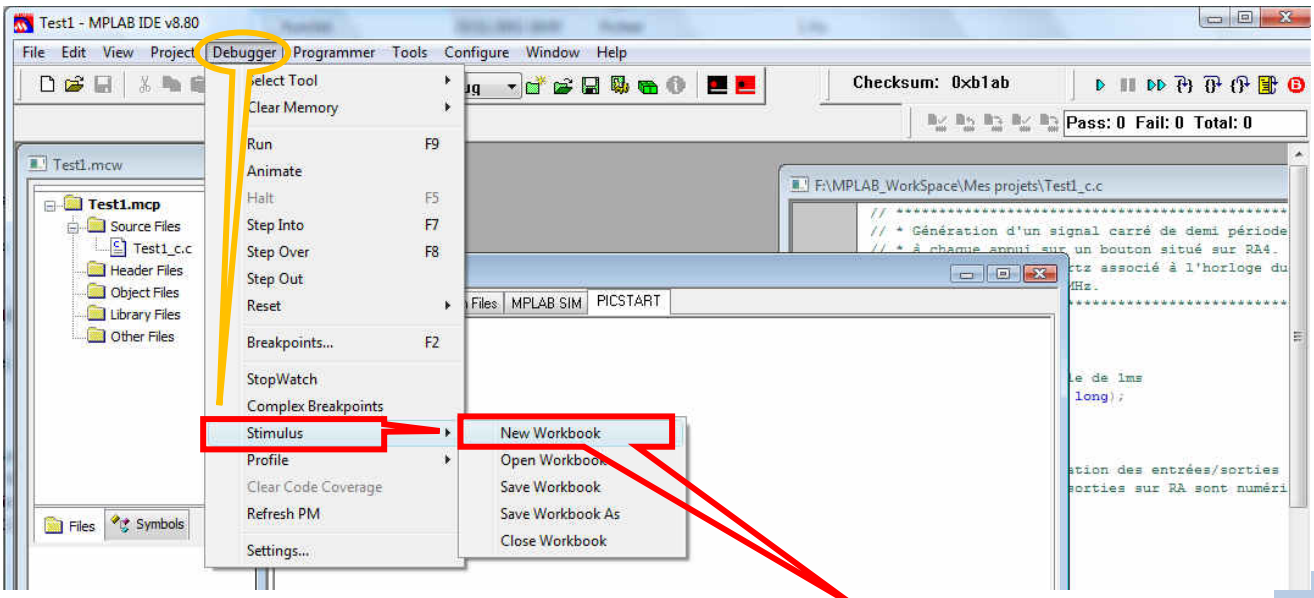
- La fréquence du quartz associé au microcontrôleur ou l'horloge externe (si c'est le cas), est la principale information à fournir pour visualiser la forme des signaux lors de la simulation.
- Sélectionner le menu **Debugger/Settings**.

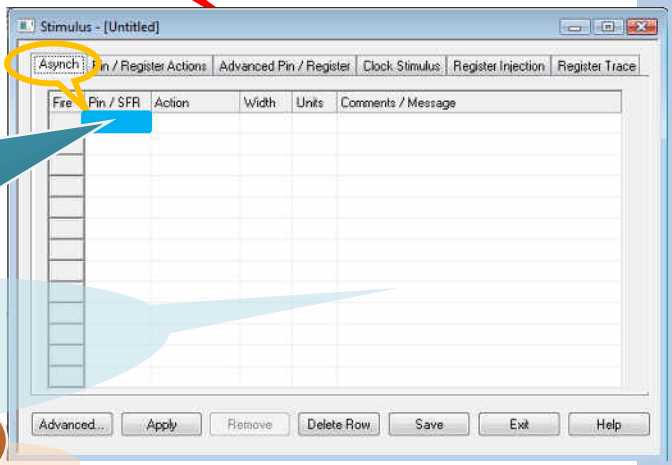


Il est utile de modifier la taille du buffer utilisé par le simulateur, lorsque les séquences à simuler sont longues (**Buffer Size**).

d. Création d'un stimulus

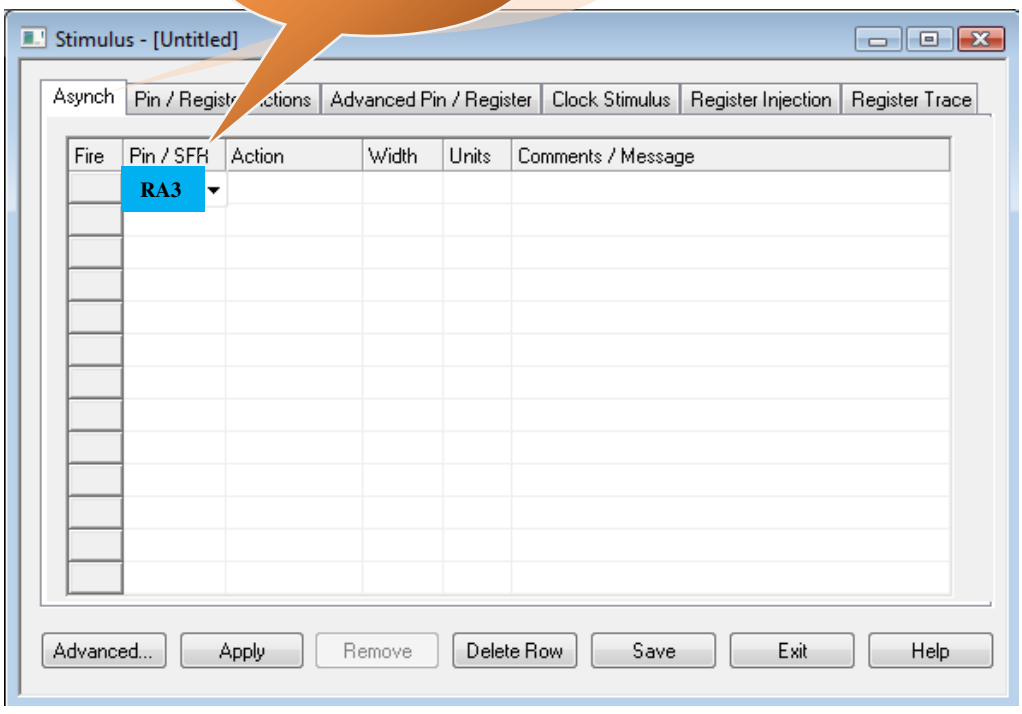
- Dans le fonctionnement du programme précédent, il faut pouvoir agir sur **RA3** pour produire ou non le signal carré sur **RA0**.
- Sélectionner le menu **Debugger/Stimulus/New Workbook**.



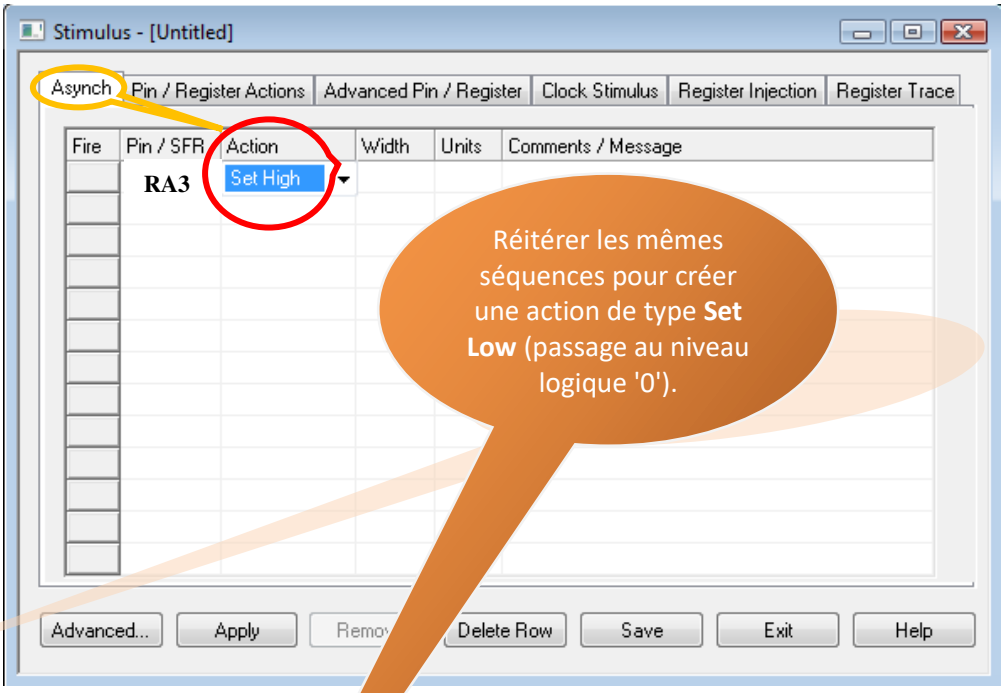


- En restant dans l'onglet **Asynch** de cette fenêtre, cliquer sur la case en surbrillance.
- Une liste déroulante comportant tous les signaux à disposition apparait.

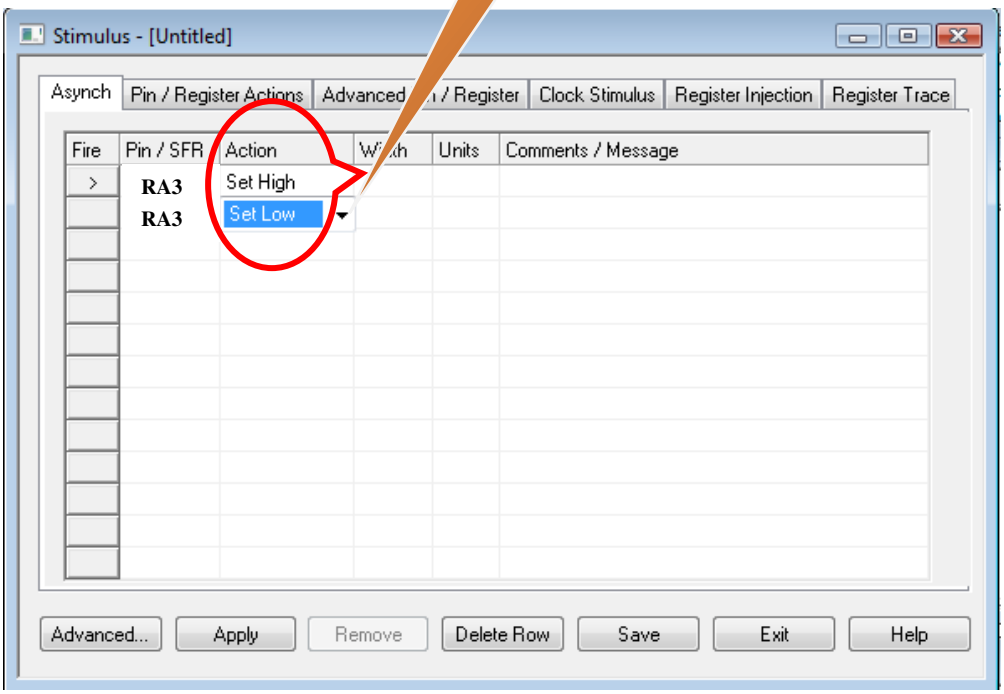
Sélectionner la broche **RA3**



- Dans la mesure où **RA3** est susceptible de posséder un niveau logique '**0**' ou '**1**', on prépare, dans la fenêtre précédente, ces deux possibilités.
- Sur la même ligne que **RA3**, dans la fenêtre précédente, et dans la colonne **Action**, sélectionner **Set High** (passage au niveau logique '**1**').



Réitérer les mêmes séquences pour créer une action de type **Set Low** (passage au niveau logique '0').



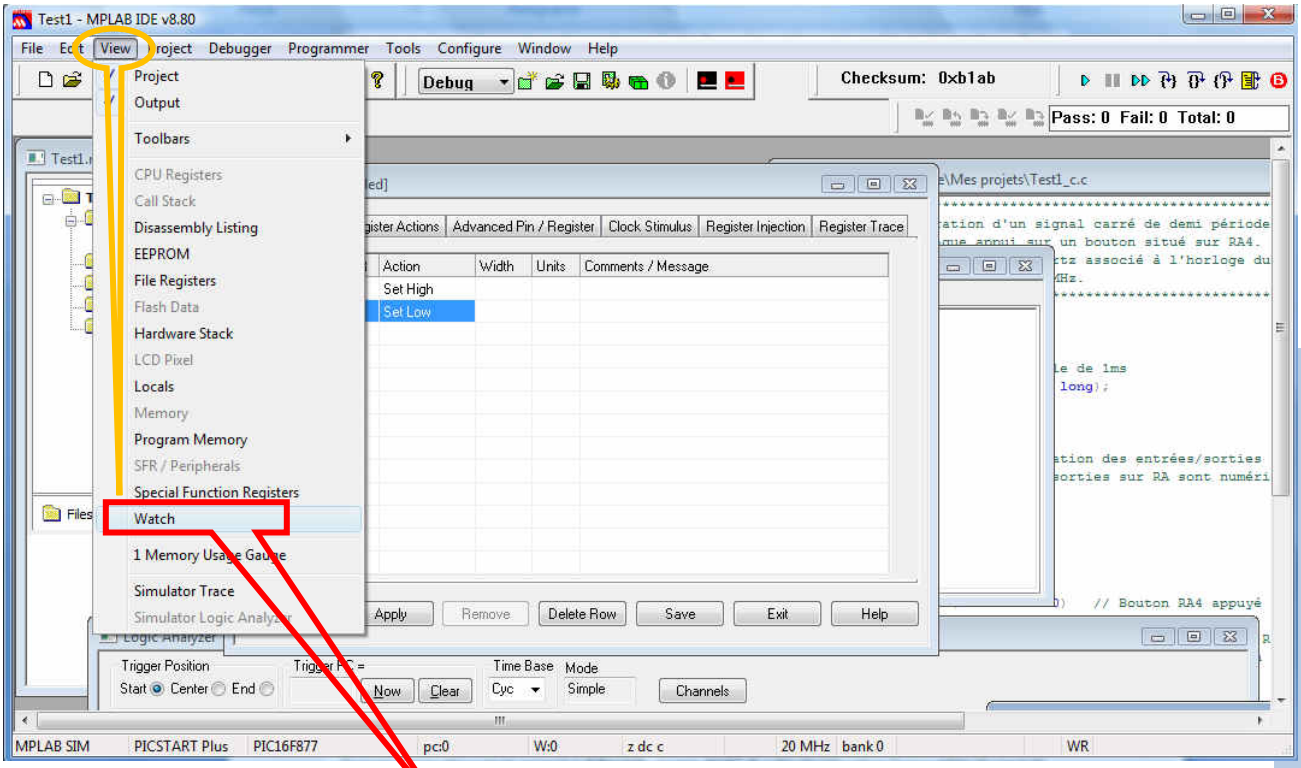
à noter!

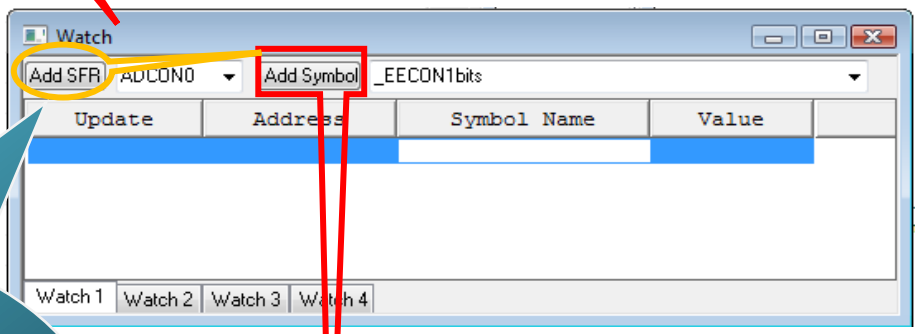
- Le forçage au niveau logique '**1**' ou '**0**', durant la simulation, se fera en cliquant dans la colonne **Fire**, respectivement sur la ligne correspondant à l'action **Set High** ou **Set Low**.

Le stimulus créé dans cette partie peut être sauvegardé, en vue d'être réutilisé dans un autre travail nécessitant des conditions analogues.

e. Visualisation des informations utiles

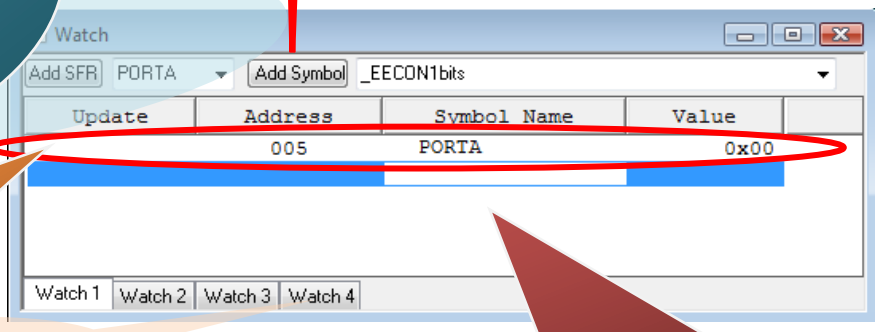
Pour visualiser les informations utiles, il suffit de sélectionner le menu **View/Watch**.





Dans le menu déroulant associé à **Add SFR** (Special Function Registers), choisir **PORTA** (afin de visualiser l'ensemble du port A du microcontrôleur).

Cliquer sur le bouton **Add SFR** pour valider ce choix.

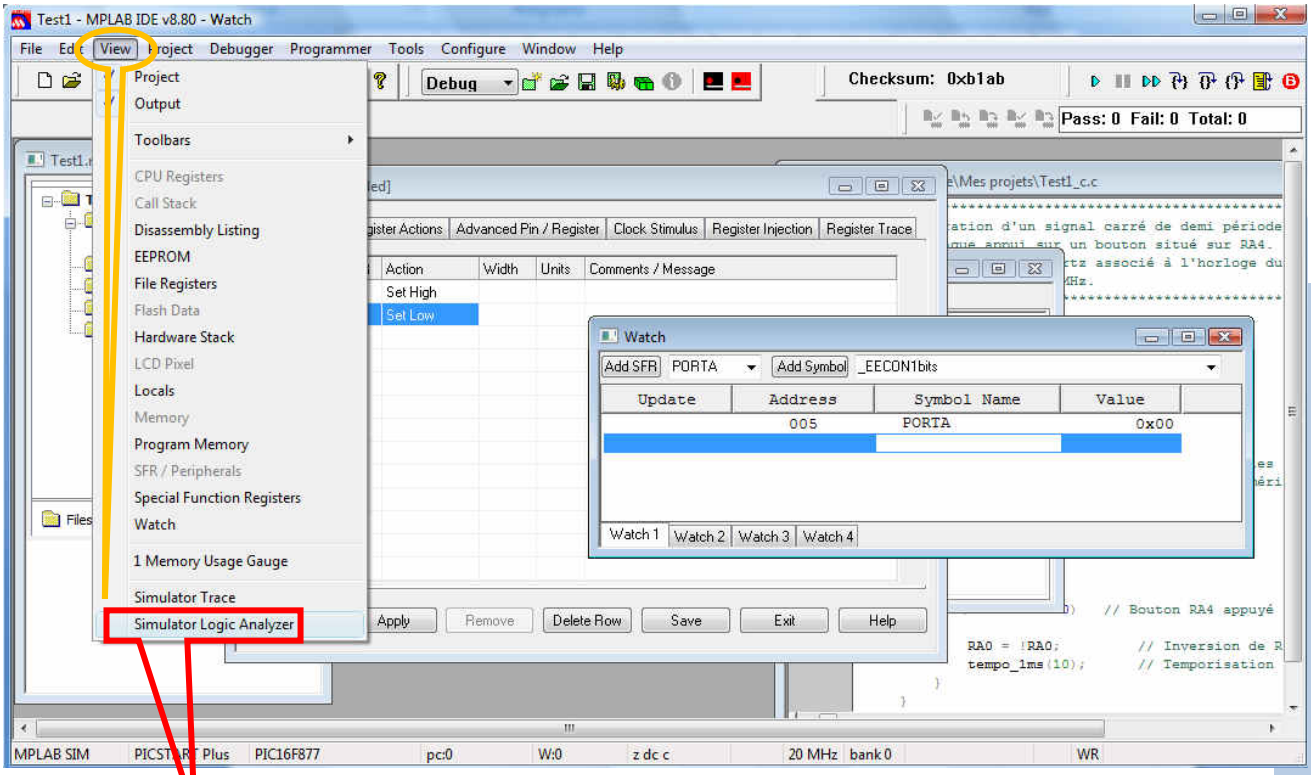


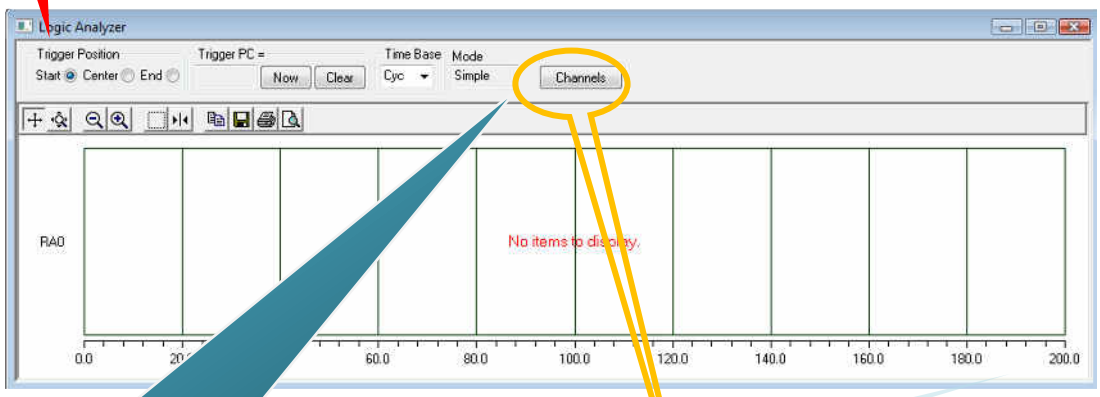
Ceci permettra de visualiser l'évolution de la valeur du **PORT A** pendant le déroulement de la simulation.

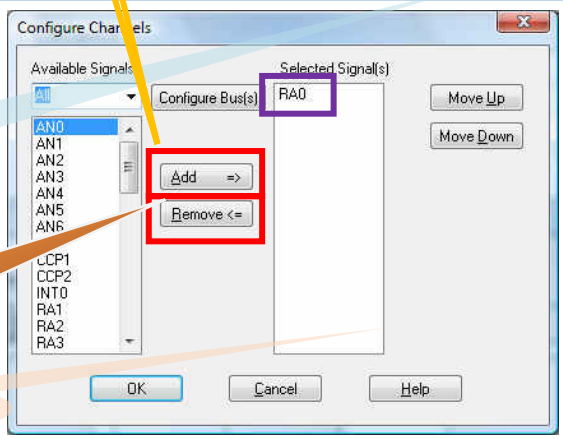
Il est possible de visualiser des variables autres que les registres en utilisant la partie **Add Symbol** de la même fenêtre.

f. Utilisation de l'analyseur logique

- Dans la mesure où le programme doit permettre de générer un signal carré de fréquence 50Hz sur la sortie **RA0** du microcontrôleur, pour une des valeurs de l'entrée **RA3**, il peut être intéressant d'observer le signal fourni par **RA0** et de mesurer ses caractéristiques.
- Pour cela, on utilise **l'analyseur logique**.
- Celui-ci est accessible par le menu **View/Simulator Logic Analyzer**.







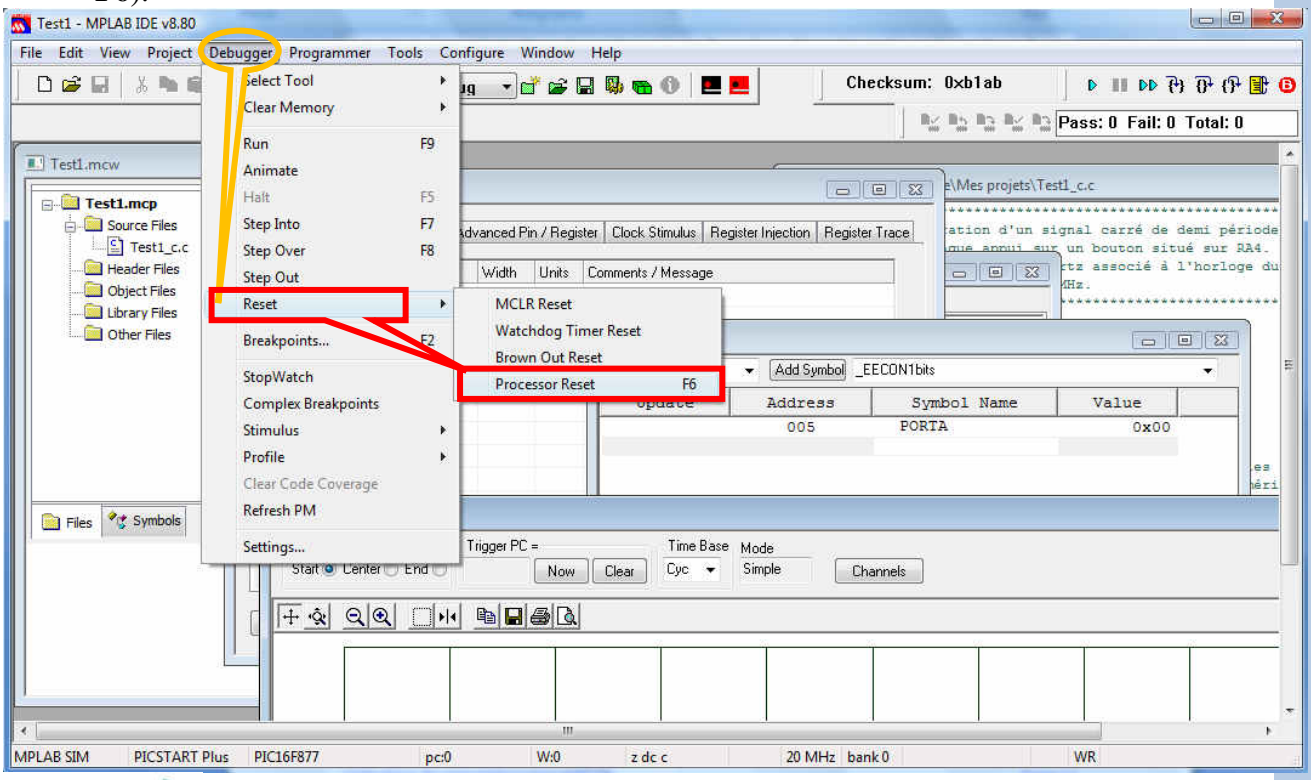
Le choix du ou des signaux à analyser se fait en cliquant sur le bouton **Channels**.

Ajouter ou supprimer des signaux.

- Le signal de sortie du PIC à visualiser dans notre cas, est le signal **RA0** choisi.

g. Lancement de la simulation

- Avant de lancer la simulation du comportement du microcontrôleur, il est impératif d'effectuer un **RESET** à l'aide du menu **Debugger/Reset/Processor Reset** (raccourci **F6**).



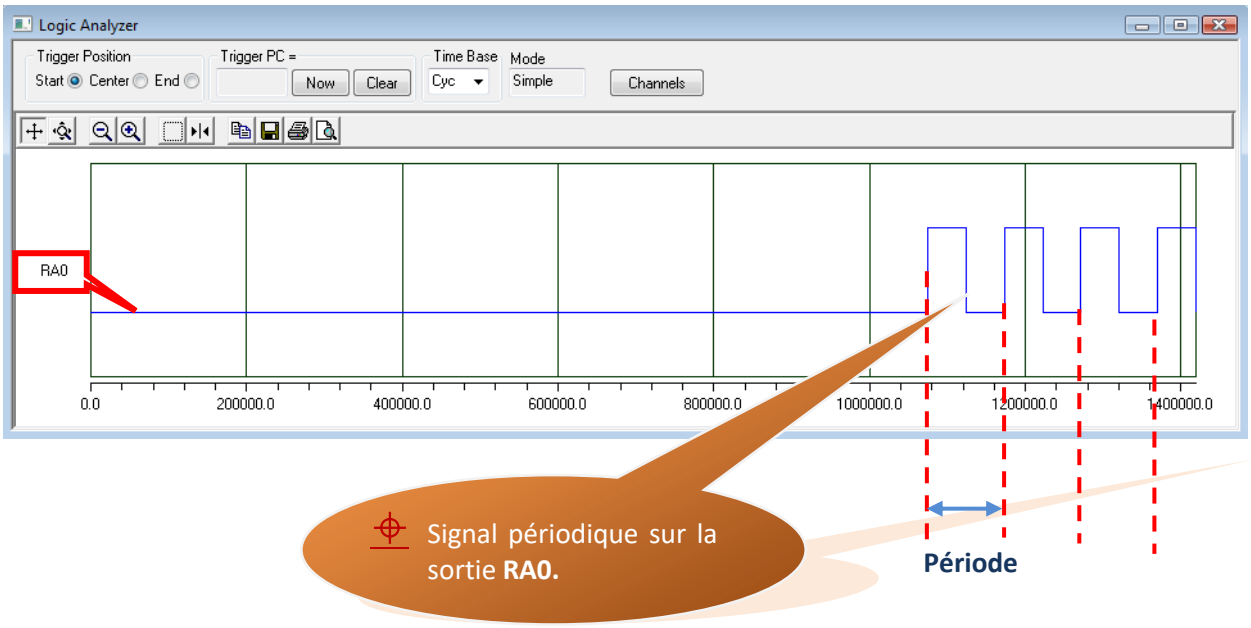
- On peut faire défiler le programme en mode **pas à pas** (menu **Debugger/Step Into** ou **Debugger/Step Over**, (raccourcis **F7** et **F8**, respectivement), ou placer des points d'arrêt judicieusement situés.
- Pour placer un point d'arrêt, il suffit de se placer dans la fenêtre contenant le programme et de **double clique** en face de la ligne concernée.

F:\MPLAB_WorkSpace\Mes projets\Test1_c.c

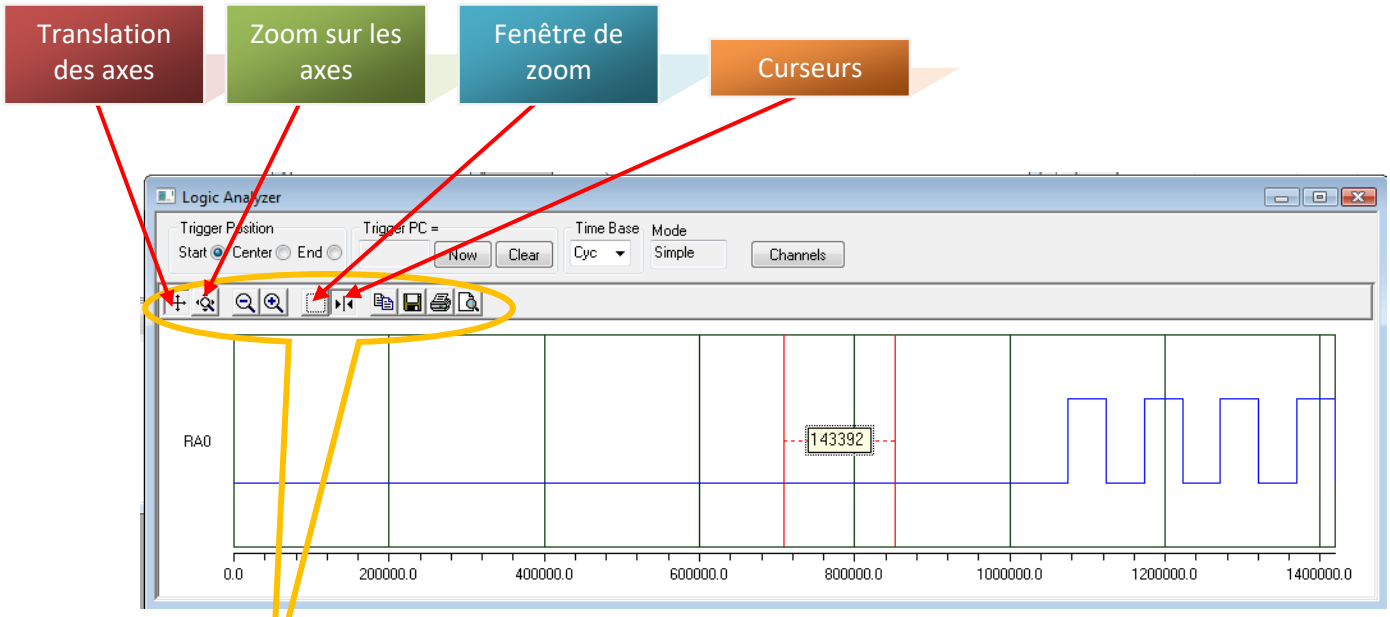
```
void main()
{
    // ******* Configuration des entrées/sorties ****************
        // Les entrées/sorties sur RA sont numériques
    ADCON1 = 0x07;
    // RA0 en sortie
    // RA4 en entrée
    TRISA = 0xFE;
    while(1)
    {
        if (PORTA & 0x10)    // Bouton RA4 appuyé
        {
            RA0 = !RA0;          // Inversion de RA0
            tempo_1ms(10);       // Temporisation de 10ms
        }
    }
}

// Temporisation multiple de 1ms
void tempo_1ms(unsigned long k)
{
    unsigned long i, j;
    for (i=0; i<k; i++)
        for (j=0; j<147; j++);

}
```

h. Exploitation des relevés de l'analyseur logique

- Dans l'exemple précédent, après avoir simulé le programme en modifiant la valeur du stimulus **RA3** créé préalablement, on obtient le relevé suivant, dans **l'analyseur logique** :



- Afin de valider complètement le fonctionnement, il est nécessaire de mesure la période du signal obtenu.
- Pour ce faire, on ajoute des curseurs sur le relevé.

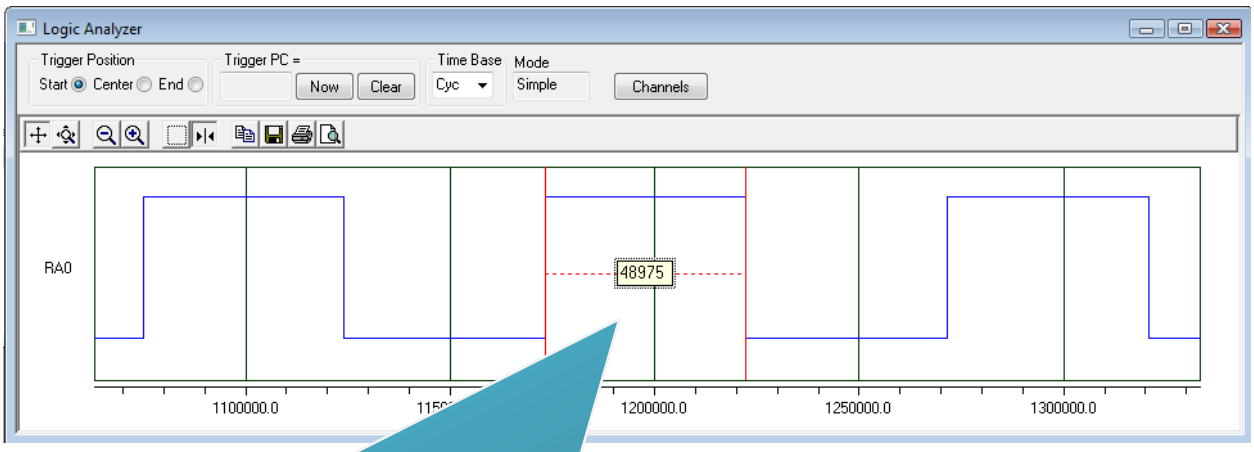


Raccourcis d'ajustement du signal visualisé après la simulation du comportement.

Info

- Les informations fournies par les curseurs s'expriment en nombre de cycles machine.
- Lorsque le quartz associé au microcontrôleur possède une fréquence **Fosc**, la fréquence machine est égale à **Fosc/4**.
- Par conséquent, la durée d'un cycle machine est égale à **4/Fosc**.

Après avoir effectué un zoom et avoir positionné correctement les curseurs, on obtient le résultat sur la fenêtre ci-dessous.



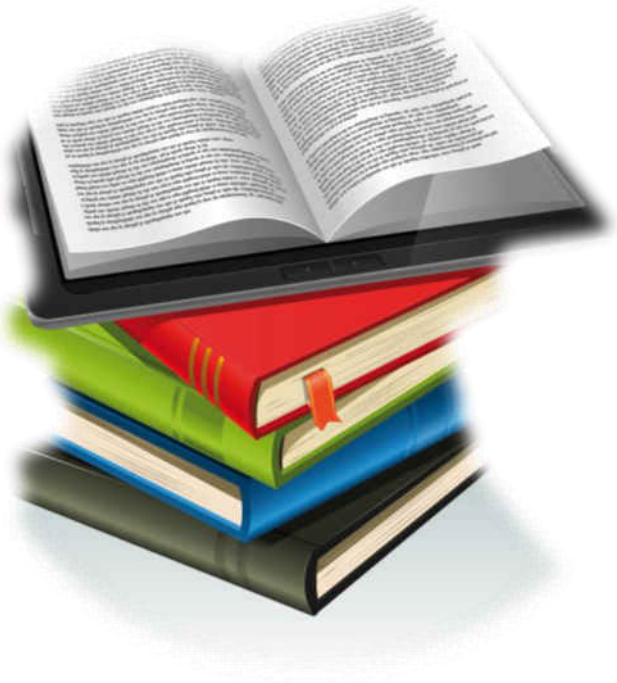
- La demi-période du signal dure donc **48975** cycles machine.
- La fréquence du quartz associé au microcontrôleur est **Fosc = 19.6608MHz.**
- **Tcycle = 4/19.6608MHz = 203.45ns.**
- La demi-période du signal est égale à :

  48975 * 203.45ns = 9.964ms.
- La période du signal est égale à :

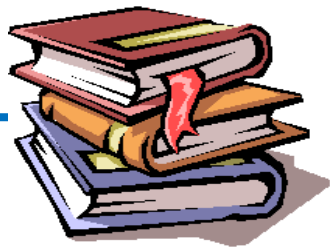
  T = 19.928ms ⇒ f = 50.18Hz

→ ce qui est conforme à l'énoncé du problème (cahier des charges).

Références Biblio-Webographiques



Références Biblio-Web graphiques



Références Bibliographiques

[1]. P. Mayeux, *Apprendre la programmation des PIC Mid Range par l'expérimentation et la simulation*, 4eme Edition, ETSF, Dunod, Paris, 2010.

[2]. J. Sanchez, M. P. Canton, *Microcontroller Programming the Microchip PIC*, CRC Press, 2006.

[3]. P. Mayeux, *Apprendre la programmation des PIC High-Performance par l'expérimentation et la simulation*, ETSF, Paris, 2010.

[4]. C. Tavernier, *Application des microcontrôleurs PIC: des PIC 10 aux PIC 18*, Dunod, 2011.

[5]. C. Tavernier, *Microcontrôleurs PIC 10, 12, 16, Description et mise en œuvre*, Dunod, 2007.

[6]. C. Tavernier, *Programmation en C des PICs*, Dunod, 2009.

[7]. B. Beghyn, *Microcontroleurs PIC*, Hermes Science Publications, 2003.

[8]. D. Ibrahim, *Advanced PIC Microcontroller*, Elsevier, 2008.

[9]. G. J. Lipovski, *Introduction to Microcontrollers*, Academic Press, California, 1999.

[10]. T. Wilmshurst, *Designing Embedded Systems with PIC Microcontrollers: Principles and applications*, Elsevier, 2007.

[11]. A. Warwick, *Programmation en C des Microcontrôleurs Embraqués*, Elektor 2009.

[12]. Microchip, *Datasheet P16F87X*, Microchip Technology Inc. 2001.

Références Webographiques

(Dernière consultation : 04/05/2019)

[13]. http://www.microchip.com/

[14]. http://www.technologuepro.com/cours-genie-electrique/cours-32-microcontroleurs/

[15]. https://waytolearnx.com/2018/11/difference-entre-microprocesseur-et-microcontroleur.html

[16]. http://hebergement.u-psud.fr/villemejane/eiti/index.php/2017/09/29/microcontroleurs/

[17]. https://fr.wikibooks.org/wiki/Comment_d%C3%A9marrer_avec_un_PIC16F84

[18]. http://fr.slideshare.net/souissi2013/cours-robotique-complet

[19]. https://fr.scribd.com/doc/20069453/Cours-Systemes-Micro-programmes-Parti1

[20]. https://waytolearnx.com/2018/11/difference-entre-microprocesseur-et-microcontroleur.html

[21]. https://www.mikroe.com/ebooks/pic-microcontrollers-programming-in-c/timer-tmr0

[22]. http://hervepage.ch/documents/pic/bigonoff.pdf

[23]. http://www.elektronique.fr/logiciels/mplab.php

[24]. http://ww1.microchip.com/downloads/en/devicedoc/51519a.pdf

[25]. https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

[26]. https://www.lycee-ferry-versailles.fr:8081

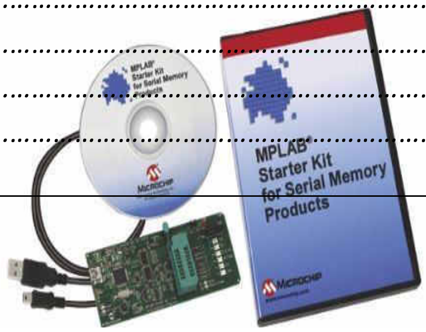
[27]. http://general.developpez.com/edi/

[28]. http://yves.heilig.pagesperso-orange.fr/ElecRob/page1.htm#PIC16F84

Feuille des remarques – Cours -

ENSEIGNANT : ______________________ **LIEU :** ________

Remarques	Page
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………
…………………………………………	………



Feuille des remarques – TD -

ENSEIGNANT : ______________________ **LIEU :** ________

Remarques	Page
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………
……	………