UNIVERSITE HASSIBA BENBOUALI DE CHLEF

FACULTE DE TECHNOLOGIE

Département d'Electronique

Polycopie de Cours
Microprocesseur et Microcontrôleur

Réalisée par

Bouziane Nasreddine

Maître de Conférence "B"

Destiné aux étudiants de la 3éme année licence en Automatique

2025-2026

A mes parents.
A tous ceux que j’aime.
Un grand merci à tous mes collègues du Département d’électronique.

PRÉFACE

CE COURS MICROPROCESSEUR ET MICROCONTRÔLEUR A ÉTÉ RÉDIGÉ À L'INTENTION DES ÉTUDIANTS DE TROISIÈME ANNÉE LICENCE SPÉCIALITÉ AUTOMATIQUE DE L'UNIVERSITÉ DE CHLEF, FACULTÉ DE TECHNOLOGIE DÉPARTEMENT ÉLECTRONIQUE. IL EST CONFORME AU PROGRAMME OFFICIEL.

L'OBJECTIF DE CE POLYCOPIE EST DE FOURNIR AUX ÉTUDIANTS DES CONNAISSANCES FONDAMENTALES ET ACTUELLES QUI PERMETTRE D'ACQUÉRIR UNE CONNAISSANCE APPROFONDIE DES MICROPROCESSEURS ET MÊME MICROCONTRÔLEUR

SOMMAIRE

CHAPITRE I ARCHITECTHURE D'UN MICRO PROCESSEUR

I.1 Introduction…… 05
I.2 Architecture externe d'un à microprocesseur…….. 05
I.2.1 Le microprocesseur…….. 06
I.2.2 La mémoire……... 06
I.2.3 Interface I/O…….. 07
I.2.4 Les buses……..……. 08
I.2.4.1 Buse de donné…… 08
I.2.4.2 Buse adresse……...… 08
I.2.4.3 Bus des commandes…….. 08
I.3 Architecture interne d'un microprocesseur……. 08
I.3.3 Les registre…… 09
I.3.3.1 Les registres d'usage général…… 09
I.3.3.2 Les registres d'adresses……. 09
I.3.2 L'unité de commande……... 10
I.3.3 Unité de traitement……... 10
I.3.3.1 ALU (Arithmetic and Logic Unit)…… 10

CHAPITRE II INTRODUCTION AU JEU D'INSTRUCTION ET INTERRUPTION

II.1 Introduction……... 14
II.2 Description schématique du 8086……. 14
II.2.1 Organisation interne du 8086…… 15
II.3 La gestion de la mémoire…… 17
II.4 Jeu d'instruction Intel 8086……. 18
II.4.1 Format d'une instruction…… 18
II.4.2 Les instructions de transfert……... 18
II.4.2.1 Instruction MOV…….. 18
II.4.2.2 Instruction de pile……. 19
II.4.3 Les instructions arithmétiques……... 20
II.4.4 Les instructions logiques……. 23
II.4.5 Les instructions de branchement……... 25
II.5 Adressage……... 28
II.5.1 Adressage registre…….. 28
II.5.2 Adressage immédiat……... 29
II.5.3 Adressage direct…….. 29
II.5.4 Adressage indirect par registre……. 29
II.5.5 Adressage basé…… 29
II.5.6 Adressage indexé…… 30
II.5.7 Adressage basé indexé…… 30
II.6 Les interruptions…….. 30
II.6.1 Affichages des caractères à l'écran……... 30
II.6.1.1 Affichages des caractères à l'écran……. 30
II.6.1.2 Afficher un message à l'écran……... 30

II.6.2 Saisir une frappe au clavier…….. 31
II.6.2.1 Saisie avec écho……..... 31
II.6.3 L'accès aux fichiers…… 32
II.6.3.1 Principe…...…….. 32
II.6.3.2 Création d'un fichier…… 32
II.6.3.3 Ouverture d'un fichier…… 33
II.6.3.4 Fermeture d'un fichier…… 33
II.6.3.5 Lecture dans un fichier……... 33
II.6.3.6 ´Ecriture dans un fichier……. 33
II.6.4 Lire l'heure courante……. 34
II.6.5 Lire la date courante…….. 34

CHAPITRE III LES MEMOIRES

III.1 Introduction……. 36
III.2 Organisation d'une mémoire…… 36
III.3 Caractéristiques d'une mémoire……... 37
III.4 Réfléchissement de mémoire…….. 37
III.5 Différents types de mémoire…….. 39
III.5.1 Mémoires vives (RAM) ……. 39
III.5.2 Les mémoires mortes (ROM)……... 40
III.6 l'interface d'un circuit mémoire au bus système 8086…… 42
III.6.1 mode d'adressage et mapping de mémoire……. 42
III.6.2 Décodage d'adresses 45

CHAPITRE IV LES INTERFACES

IV.1 Définition ……. 48
IV.2 Généralité sur interfaçage……... 48
IV.2.1Adressage des ports d'E/S……... 49
IV.3 Les Interfaces parallèles……. 50
IV.3.1 Le port parallèle…….. 51
IV.3.2 Le coupleur parallèle PPI 8255…….. 53
IV.4 L'interface série……... 62
IV.4.1 Circuit intégré 8250…… 63

CHAPITRE V LE MICROCONTROLEUR

V.1 Introduction……... 67
V.2 Généralités sur le microcontrôleur…….. 67
V. 2.1 Définition d'un PIC…… 67
V.2. 2 Les différentes familles des PICs……...…… 68
V.3 Architecture du microcontrôleur……. 69
V.3.1 Présentation de PIC 16F84…….. 69
V.3.2 Brochage et caractéristiques principales……... 71
V.4 La programmation des microcontrôleurs…….. 73
V.4.1 Le jeu instructions…….. 73
V.4.1.1Instructions sur les registres : octet……. 73
V.4.1.2 Instructions sur le bit d'un registre…….…... 74
V.4.1.3 Instructions générales de contrôle…….. 75
V.4.1.4 Instructions de saut et appel……. 75

V.5 Interruptions…….. 76
V.6 Mise en cas pratique via Mplab……... 78
V.6.1 Organigramme…… 79
V.6.2 Choisir le pic…… 79
V.6.3 Configuration de pic……... 79
V.6.4 configuration des ports……... 80
V.6.5 Programme principal…….. 80
Exercices proposer avec ces solutions……. 82
Références……. 149

CHAPITRE I

I.1 Introduction :

Le premier microprocesseur a été fabriqué par INTEL en 1971. C'était un 4 bits baptisé 4004 destiné à équiper des calculatrices de bureau. En 1972 INTEL produit le premier microprocesseur 8 bits baptisé 8008 par référence au précédent. Ce microprocesseur était destiné à répondre à un contrat de fabrication d'un terminal. En réalité le 8008 s'est avéré trop lent pour satisfaire au cahier des charge du terminal et INTEL a décidé de tenter le lancement de ce produit sur le marché grand public. L'énorme succès de cette initiative fut à l'origine de la fabrication massive des microprocesseurs. A la suite du succès du 8008, INTEL produisit dès 1974 le 8080 qui constituera le premier élément de la future famille de microprocesseurs de ce fabriquant. En 1974, MOTOROLA, autre fondeur de silicium, décide de lancer le 6800 qui constituera lui aussi le début d'une grande famille. Les années 70 voient alors apparaître de petites entreprises de fabrication de microprocesseurs souvent constituées par des transfuges des deux grandes compagnies. On peut notamment citer MOSTEK avec son 6502 très inspiré du 6800 et ZILOG avec son Z80 qui constitue une amélioration technique du 8080 (augmentation du nombre de registres, simplification de l'alimentation...). Très vite les 8 bits commencent à atteindre la limite de leurs performances et les constructeurs se penchent sur de nouvelles solutions. Les principaux reproches faits aux 8 bits sont :

- Peu d'espace mémoire accessible (64K octets)
- Peu de types d'informations manipulés (1 ou 2 octets)
- Pas adaptés aux architectures multiprocesseurs
- Peu de modes d'adressage Pour toutes ces raisons les constructeurs commencent à étudier une nouvelle génération de microprocesseurs.

De sorte que l'on verra apparaître des microprocesseurs 8 bits plus puissants permettant de manipuler plus facilement des informations sur 16 bits et offrant de nouvelles possibilités d'adressage comme le 6809 de MOTOROLA ou le 8085 d'INTEL.

I.2 Architecture externe d'un à microprocesseur (Newman)

Dans un système à microprocesseur (Architecture de Von Neumann), on retrouve au minimum : (1) Un microprocesseur, (2) Une mémoire morte (ROM) + Une mémoire vive (RAM), (3) Une interface entrées/sorties

Tous ces organes sont reliés entre eux avec 3 bus : Bus de données bidirectionnel, Bus d'adresse, Bus de commande

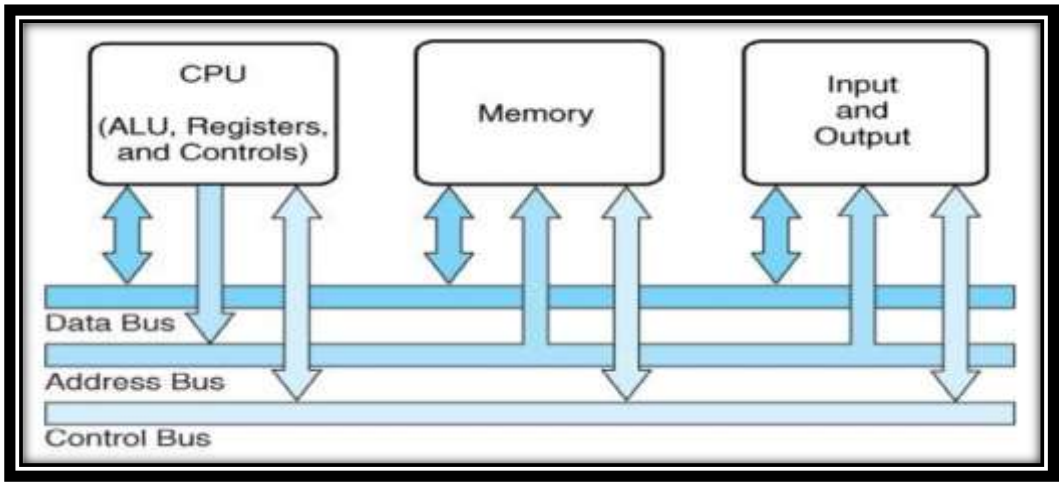


Figure (I.1) : Système à microprocesseur

I.2.1 Le microprocesseur

Le microprocesseur est chargé d'effectuer les fonctions suivantes :

☞ Fournir les signaux de synchronisation et de commande à tous les éléments du système,

☞ Prendre en charge les instructions et les données dans la mémoire,

☞ Transfert des données entre la mémoire et les dispositifs d'E/S et vice versa,

☞ Décoder les instructions,

☞ Effectuer les opérations arithmétiques et logiques commandées par les instructions,

☞ Réagir aux signaux de commande produits par les entrées/sorties comme le signal "Reset" et les interruptions.

I.2.2 La mémoire

La mémoire a pour rôle de conserver des groupes de chiffres binaires (mots) qui sont soit des instructions formant un programme, soit des données dont le programme a besoin.

Elle se décompose souvent en :

➢ une mémoire morte (ROM = Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.

1) ROM : Read Only Memory. Mémoire à lecture seule, sans écriture. Son contenu est programmé une fois pour toutes par le constructeur. Avantage : faible coût.

Inconvénient : nécessite une production en très grande quantité.

2) PROM: Programmable Read Only Memory. ROM programmable une seule fois par l'utilisateur.

3) EPROM : Erasable PROM, appelée aussi UVPROM. ROM programmable électriquement avec un programmateur et effaçable par exposition à un rayonnement ultraviolet.

3) EEPROM : Electrically Erasable PROM. ROM programmable et effaçable électriquement.

Les EEPROM contiennent des données qui peuvent être modifiées.

➢ une mémoire vive (RAM = Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données de dans, ces données

sont perdues à la mise hors tension.

1) SRAM: Static Random Access Memory. Mémoire statique à accès aléatoire, à base de bascules à semi-conducteurs à deux états (bascules RS).

2) DRAM : Dynamic RAM. Basée sur la charge de condensateurs : condensateur chargé =1, condensateur déchargé = 0.

Remarque :

Les disques durs, disquettes, CDROM, etc… sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

On peut donc schématiser un circuit mémoire par la figure suivante où l'on peut distinguer :

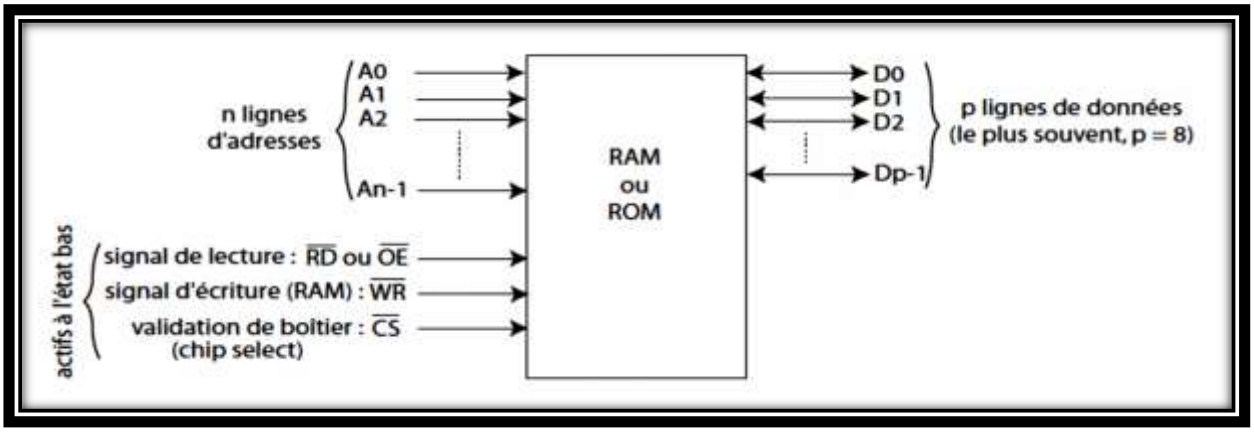


Figure (I.2) : Schéma fonctionnel d'une mémoire

Les entrées d'adresses, les entrées / sorties de données, les entrées de commandes :

Entrée de sélection de lecture ou d'écriture. (R/ W), une entrée de sélection du circuit. (CS).

Le nombre de lignes d'adresses dépend de la capacité de la mémoire : n lignes d'adresses permettent d'adresser 2n cases mémoire : 8 bits d'adresses permettent d'adresser 256 octets, 16 bits d'adresses permettent d'adresser 65536 octets (= 64 Ko), ...

Exemple : mémoire RAM 6264, capacité = 8K×8 bits : 13 broches d'adresses A0 à A12

2^{13} = 8192 = 8 Ko.

I.2.3 Interface I/O

C'est un circuit intégré permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante, bouton poussoir, processus industriel, actionneurs…

Les transferts E/S correspondent à des instructions de lecture et d'écriture. Le plus souvent le circuit d'interface comporte un registre appelé « port ».

- Ecriture : Le microprocesseur envoie l'ordre d'écriture et les données sont mémorisées dans le registre de l'interface.
- Lecture : le circuit se contente de présenter les informations du périphérique sur le bus de données.

I.2.4 Les buses :

I.2.4.1 Buse de donné :

Il permet de véhiculer des données du microprocesseur vers un composant ou d'un composant vers le microprocesseur. Il est donc bidirectionnel. Le nombre de fils de ce bus varie suivant : Les microprocesseurs (8 / 16 / 32 / 64 bits). Dans la littérature, les différents fils de ce bus sont appelés D0, D1, ..., Dp-1, si le bus a "p" fils.

I.2.4.2 Buse adresse :

La mémoire est composée de nombreuses cases mémoires. Chaque case est repérée par une adresse. Lorsque le microprocesseur veut, par exemple, lire une case, il doit indiquer à quelle adresse elle se trouve. Il met cette adresse sur le bus des adresses. La case mémoire reconnaît alors son adresse et met sur le bus de données son contenu.

Exemple : Bus d'adresse 16 bits - données sur 8 bits.

I.2.4.3 Bus des commandes :

Le bus des commandes est constitué d'un ensemble de fils de "commandes", permettant la synchronisation et bien sûr la commande des boîtiers mémoires et entrées/sorties par le microprocesseur. Dans le cas précédent, la cellule mémoire doit savoir à quel instant elle doit mettre son contenu sur le bus de données. Pour cela, le microprocesseur possède une broche appelée Read (RD) qu'il met à 0 (0v) lorsque la cellule doit agir. De même, lors d'une écriture du microprocesseur vers la cellule, il met sa broche Write (WR) à 0 (0V). Les signaux RD et WR sont des signaux de synchronisation, de contrôle et de commande. Ils sont reliés aux autres composants par un bus : le bus des commandes. Celui-ci comporte d'autres signaux de commandes

I.3 Architecture interne d'un microprocesseur

Le microprocesseur est construit autour de deux éléments principaux :

- Unité de traitement
- Unité de commande

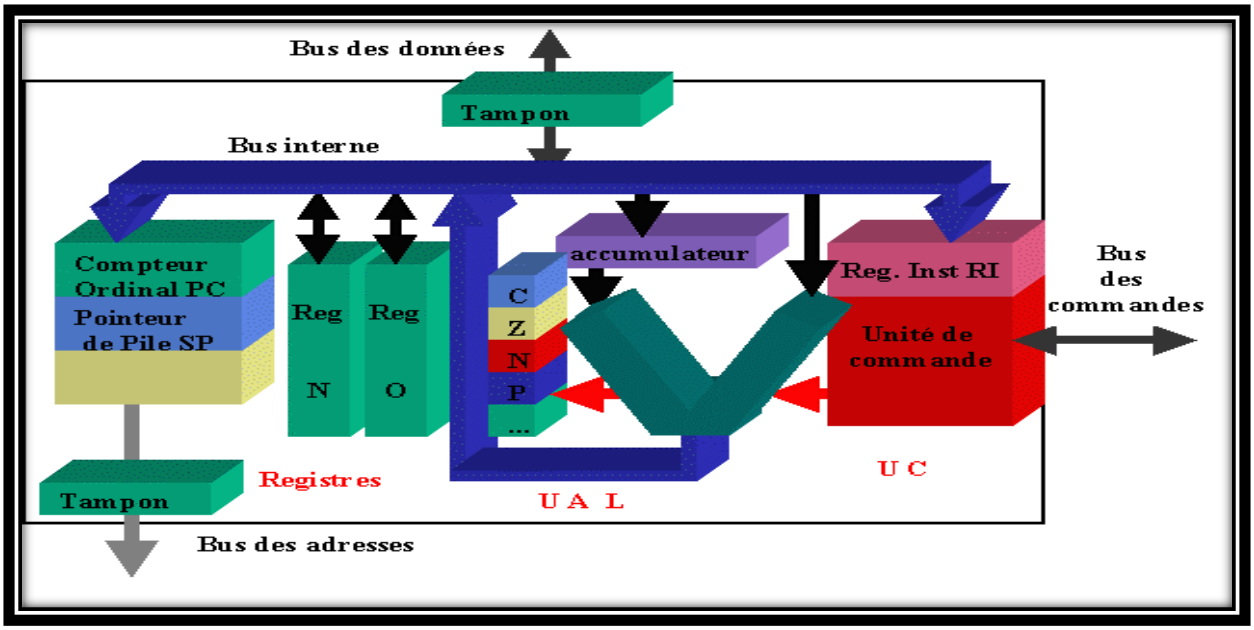


Figure (I.3) : Architecture interne d'un microprocesseur (Intel µp 8086)

I.3.1 Les registre :

Il y'a deux type de registres : les registres d'usage général, et les registres d'adresses (pointeurs).

I.3.3.1 Les registres d'usage général:

Ce sont des mémoires rapides, à l'intérieur du microprocesseur, qui permettent à l'UAL de manipuler des données à vitesse élevée. Ils sont connectés au bus de données interne au microprocesseur. L'adresse d'un registre est associée à son nom (on donne généralement comme nom une lettre) A, B,C...

Exemple : MOV C, B : transfert du contenu du registre B dans le registre C.

I.3.3.2 Les registres d'adresses (pointeurs) :

Ce sont des registres connectés sur le bus d'adresses. On peut citer comme registre :

✓ Le compteur ordinal (pointeur de programme PC) : Il contient l'adresse de l'instruction à rechercher en mémoire. L'unité de commande incrémente le compteur ordinal (PC) du nombre d'octets sur lequel l'instruction, en cours d'exécution, est codée. Le compteur ordinal contiendra alors l'adresse de l'instruction suivante.

Prenons l'exemple d'un microprocesseur 8086 INTEL

Exemple : (PC)=10000H ; il pointe la mémoire qui contient l'instruction MOV CX, BX qui est codée sur deux octets (89 D9H) ; l'unité de commande incrémentera de deux le contenu du PC : (PC) = 10002H (la mémoire sera supposée être organisée en octets).;

✓ Le pointeur de pile (stack pointer SP) : Il a pour rôle d'indiquer au microprocesseur la prochaine case disponible dans la pile. ;

✓ Les registres d'index (index source SI et index destination DI).

I.3.2 L'unité de commande.

C'est l'ensemble séquenceur/décodeur d'instruction. Elle est composée par :

☞ Séquenceur :C'est le maître d'orchestre, il cadence le microprocesseur. C'est lui qui pilote le bus de commande et les blocs internes

☞ Décodeur d'instruction : Il reconnaît l'instruction et la transforme en signaux internes grâce à un microcode contenu dans une ROM interne.

☞ Le compteur de programme : Constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de l'instruction à exécuter.

☞ Stack Pointer : Il a pour rôle d'indiquer au microprocesseur la prochaine case disponible dans la pile.

Elle permet de "séquencer" le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction, le décodage, l'exécution et la préparation de l'instruction suivante.

L'unité de commande élabore tous les signaux de synchronisation internes ou externes (bus des commandes) au microprocesseur.

I.3.3 Unité de traitement

Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions.

Les blocs de l'unité de traitement :

I.3.3.1 ALU (Arithmetic and Logic Unit)

C'est l'unité qui permet d'effectuer des calculs arithmétiques simples (additions, soustractions, décalages, etc.) et des opérations logiques (ET, OU, etc.).

Deux registres sont associés à l'UAL : l'accumulateur et le registre d'état.

a) L'accumulateur (nommé : A).

C'est une des deux entrées de l'UAL. Il est impliqué dans presque toutes les opérations réalisées par l'UAL. Certains constructeurs ont des microprocesseurs à deux accumulateurs (Motorola : 6800).

Exemple: A étant l'accumulateur et B un registre, on peut avoir : A+B (ADD A,B : addition du contenu du registre A avec celui du registre B, le résultat étant mis dans A)

b) Le registre d'état (Flags : F)

A chaque opération, le microprocesseur positionne un certain nombre de bascules d'état. Ces bascules sont appelées aussi indicateurs d'état ou drapeaux (status, flags). Par exemple, si une

soustraction donne un résultat nul, l'indicateur de zéro (Z) sera mis à 1. Ces bascules sont regroupées dans le registre d'état

On peut citer comme indicateurs:

- retenue (carry : C)
- retenue intermédiaire (Auxiliary-Carry : AC)
- signe (Sign : S)
- débordement (Overflow : O)
- zéro (Z)
- parité (Parity : P)

- ✓ **Retenue : (carry : C).**

Exemple: addition de nombres binaire sur 8 bits

11111100 FCH

+ 10000010 + 82H

carry : 1 = 01111110 carry : 1 = 7EH

Sur cet exemple, nous pouvons remarquer que le résultat sur 8 bits est faux (7EH). Toutefois Le bit de retenu expulsé n'est pas définitivement perdu. Ce bit C (carry) est stocké dans le registre d'état est peut être testé par l'utilisateur. De la même manière lors d'une opération de décalage ou de rotation ce bit peut être positionné en cas de débordement. Par exemple, soit le décalage à gauche sur d'un bit de cet octet : 10010110, après le décalage nous avons 00101100. En ce qui concerne le carry, il va être positionné à 1, puisque le bit expulsé est égal à 1.

- ✓ **Retenue intermédiaire : (Auxiliary Carry : AC).**

Sur les opérations arithmétiques, ce bit signale une retenue entre groupes de 4 bits (Half-byte: demi-octet) d'une quantité de 8 bits.

- ✓ **Signe : (S)**

Ce bit est mise à 1 lorsque le résultat de l'opération est négatif (MSB: bit de plus fort poids du résultat: à 1).

- ✓ **Débordement : (overflow : O)**

Cet indicateur est mis 1, lorsqu'il y a un dépassement de capacité pour les opérations arithmétiques en complément à 2. Sur 8 bits, on peut coder de -128 (1000 0000) à +127 (0111 1111).

104 0110 1000 - 18 1110 1110

+ 26 + 0001 1010 - 118 1000 1010

=130 = 1000 0010 (-126) - 136 0111 1000 (120) avec C=1

Dans cet exemple et dans les deux cas de figure (opération à droite ou opération à gauche), le bit O est positionné à 1. L'indicateur de débordement est une fonction logique (OU exclusif) de la retenue (C) et du signe (S).

- ✓ **Le bit Zéro : (Zéro : Z)**

Ce bit est mis à 1 lorsque le résultat de l'opération est nul.

- ✓ **Le bit de parité : (P)**

Ce bit est mis à 1 lorsque le nombre de 1 de l'accumulateur est pair.

Remarque : La plupart des instructions modifient le registre d'état.

Exemple :

ADD A, B positionne les drapeaux : O, S, Z, A, P, C

OR B, C (B ou C -> B) positionne S, Z, P tandis que

MOV A, B (Move, Transférer le contenu de B dans A) n'en positionne aucun.

CHAPITRE II

II.1 Introduction :

La mise en œuvre de système de microprocesseur s'appuie sur deux modes de réalisation distincte, le matériel et le logiciel. 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.

Dans ce chapitre ont utilisant le microprocesseur 8086

II.2 Description schématique du 8086

Le microprocesseur Intel 8086 est un microprocesseur 16 bits, apparu en 1978. C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...). Il se présente sous la forme d'un boitier DIP (Dual In-line Package) `a 40 broches :

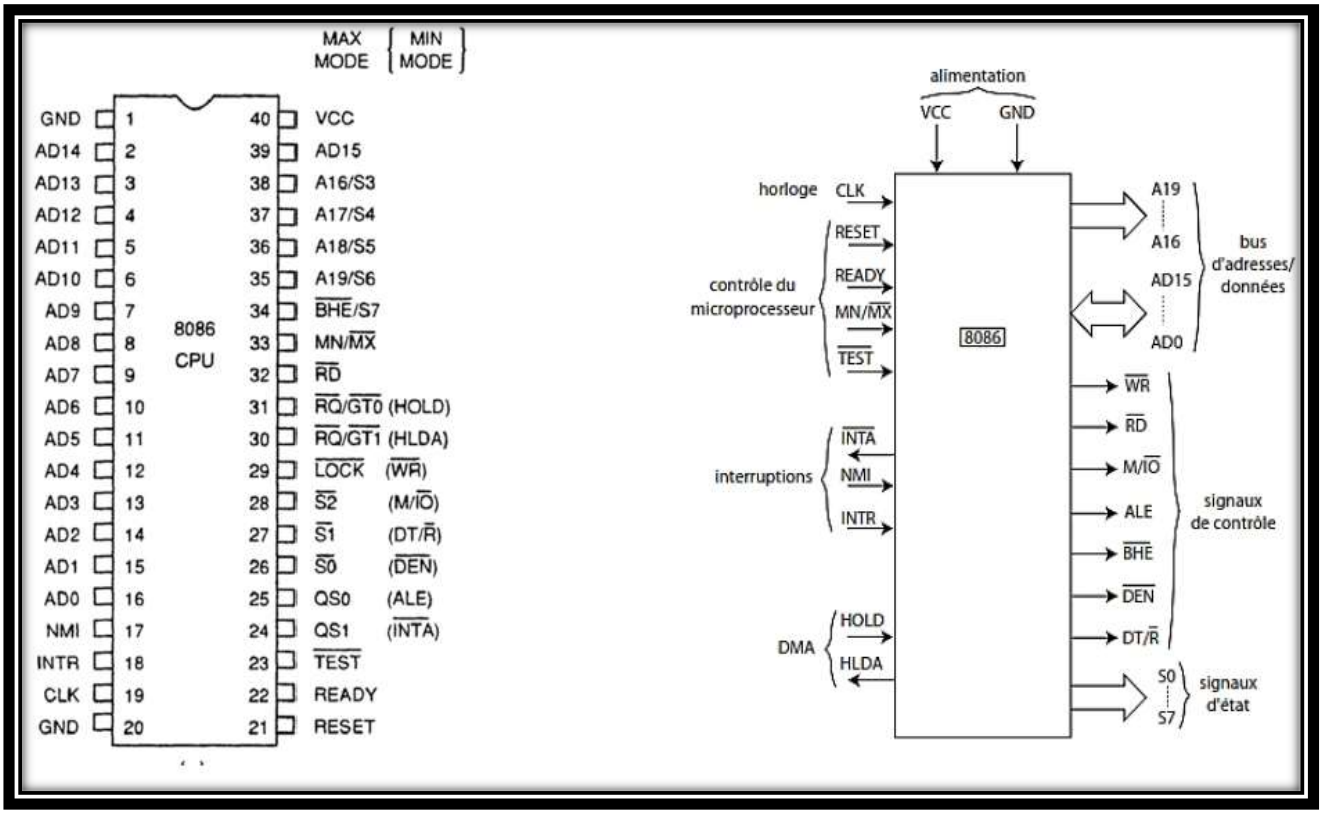


Figure (II.1) : Brochage et Schéma fonctionnel du 8086

Brochage et Fonctions des Pins du 8086 :

CLK : entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un générateur d'horloge : le 8284.

AD0 à AD15 : Ces lignes représentent 16 bits de bus d'adresse multiplexés avec 16 bits de lignes de données. Pendant T1, elles représentent des lignes d'adresse A15-A0.

Pendant T2, T3, T4, elles représentent des lignes de données D0-D15.

A16/S3 à A19/S6 : Ces lignes adresse sont multiplexées avec les lignes d´état. Pendant T1, elles représentent des lignes d'adresse A19-A16. Pendant T2, T3, T4, elles représentent des signaux d'état S6-S3.

S0 à S7 : signaux d'état indiquant le type d'opération en cours sur le bus.

ALE (Address Latch Enable): Ce signal est une impulsion active pendant T1, elle indique que l'information qui circule dans bus A/D est une adresse. Elle est fournie par le CPU pour verrouiller les lignes d'adresse au cours des cycles T2, T3, T4.

RD : Read, signal de lecture d'une donnée.

WR : Write, signal d´écriture d'une donnée.

M/IO : Memory/Input-Output, indique si le CPU adresse la mémoire (M/IO = 1) ou les entrées/sorties (M/IO = 0).

DEN : Data Enable, indique que des données sont en train de circuler sur le bus A/D (équivalente de ALE pour les données).

DT/R : Data Transmit/Receive, indique le sens de transfert des données :

• DT/R = 1 : données émises par le microprocesseur (écriture) ;

• DT/R = 0 : données reçues par le microprocesseur (lecture).

II.2.1 Organisation interne du 8086

Le 8086 est constitué de deux unités internes distinctes:

• l'unité d'exécution (EU : Execution Unit) ;

• l'unité d'interface de bus (BIU : Bus Interface Unit).

Le rôle de la BIU est de récupérer et stocker les informations à traiter, et d'établir les transmissions avec les bus du système. L'EU exécute les instructions qui lui sont transmises par la BIU.

Pendant que l'EU du 8086 exécute les informations qui lui sont transmises, l'instruction suivante est chargée dans la BIU. Les instructions qui suivront sont placées dans une file d'attente. Lorsque l'EU a fini de traiter une instruction la BIU lui transmet instantanément l'instruction suivante, et charge la troisième instruction en vue de la transmettre à l'EU. De cette façon, l'EU est continuellement en activité.

Le microprocesseur 8086 contient 14 registres répartis en 4 groupes :

1) Registres de travails : 4 registres sur 16 bits AX, BX, CX et DX.

Chacun de ces registres est de 16-bits de large, mais Ils peuvent être également considérés comme 8 registres sur 8 bits.

Registre AX : (Accumulateur) : registre de travail principal.

Registre BX : (registre de base) : Il est utilisé pour l'adressage.

Registre CX : (Le compteur) : le registre CX a été fait pour servir comme compteur lors des instructions de boucle.

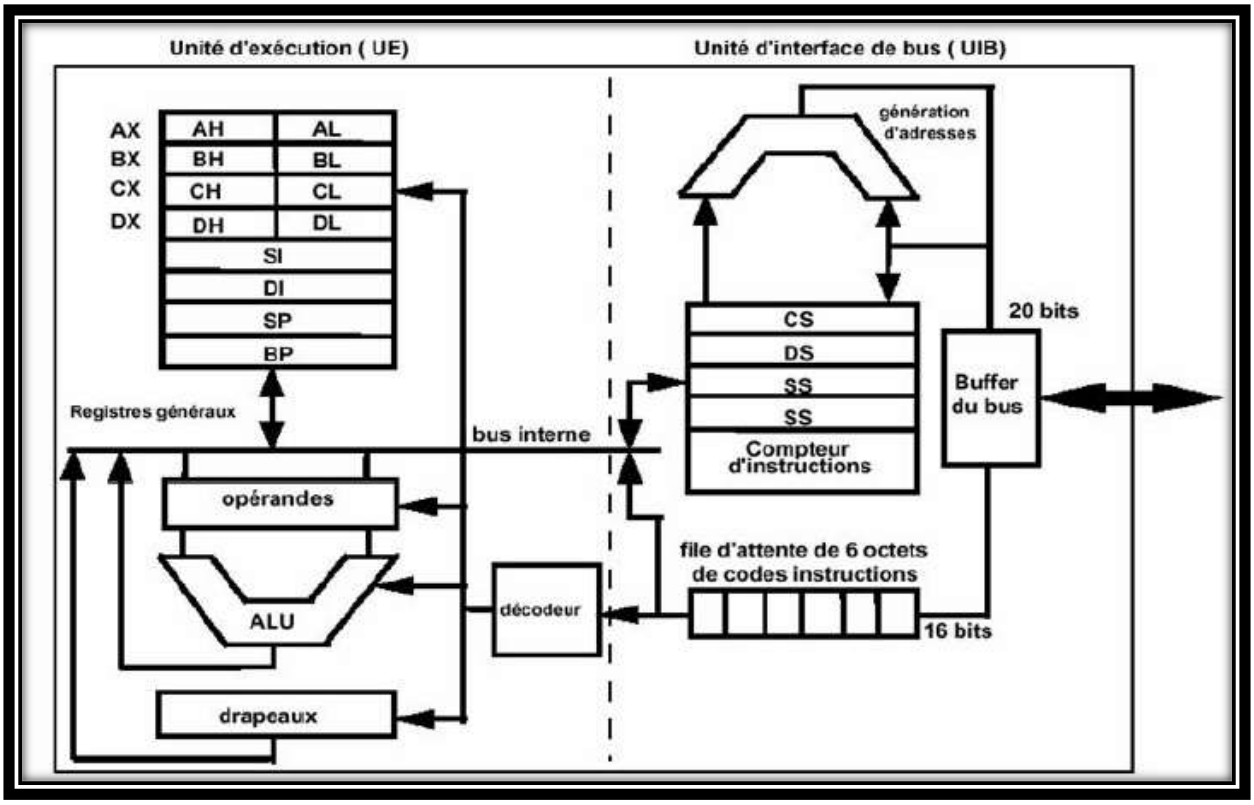


Figure (II.2) : architecture du microprocesseur 8086

Registre DX : On utilise le registre DX pour les opérations de multiplication et de division et pour contenir le numéro d'un port d'entrée/sortie pour adresser les interfaces d'E/S.

2) Registres de segments : 4 registres sur 16 bits.

Ils sont employés par la CPU pour déterminer les adresses de segment de mémoire.

Le registre CS (code segment) : Il pointe sur le segment qui contient les codes des instructions du programme exécutable.

Le registre DS (Data segment) : Le registre segment de données pointe sur la zone mémoire de données.

Le registre ES (Extra segment) : segment auxiliaire pour données.

Le segment SS (Stack segment) : Le registre SS pointe sur la pile.

Remarque : la pile est une zone mémoire ou on peut sauvegarder le contenu des registres, les adresses ou les données pour les récupérer après l'exécution d'un sous-programme ou l'exécution d'un programme d'interruption.

3) Registres de pointeurs et d'index : 4 registres sur 16 bits.

Ce sont des registres de 16 bits ils sont utilisés comme pointeurs de mémoire.

L'index SI : (source indexe) : Il est associé au registre DS, il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source.

L'index DI : (Destination indexe) : Il est associé aux registres (DS ou ES), il sert aussi pour les instructions de chaîne de caractères, il pointe alors sur la destination.

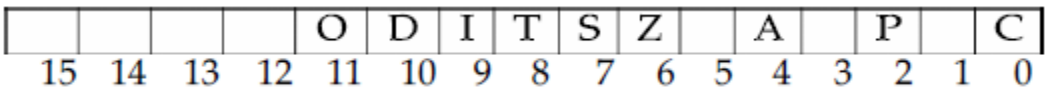
Les pointeurs SP et BP : (Stack pointer et base pointer) :

Ils sont associés au registre SS, Ils pointent sur la zone pile. Pour le registre BP a un rôle proche de celui de BX, le registre SP Pointe sur la tête de la pile.

Le pointeur IP : Associé au registre CS, il a seulement une fonction qui est d'indiquer la prochaine instruction à exécuter par le CPU.

4) le registre FLAG : La plupart des instructions arithmétiques et logiques affectent le registre d'état du processeur (ou registre de FLAG). C'est un registre de 16 bits. Six des drapeaux sont des indicateurs d'état reflétant les propriétés du résultat de la dernière instruction arithmétique ou logique. Trois des indicateurs peuvent être activés ou remis à zéro par le programmeur : (TF, IF, DF).

Flags :



CF : indicateur de retenue (carry) ;

PF : indicateur de parité;

AF : indicateur de retenue auxiliaire ;

ZF : indicateur de zéro ;

SF : indicateur de signe ;

TF : indicateur d'exécution pas à pas (trap) ;

IF : indicateur d'autorisation d'interruption ;

DF : indicateur de décrémentation ;

OF : indicateur de dépassement (overflow).

II.3 La gestion de la mémoire

Le processeur 8086 a un bus d'adresses du bit 20, Ceci permet au processeur d'adresser 220 ou 1,048,576 emplacements mémoire différents (1 Mo).

Puisque les registres du 8086 ne font que 16 bits et avec 16 bits en peut adresser que 64 ko alors INTEL a proposé de fractionner la mémoire totale adressable de 1 Mo en bloc de 64 ko

appelés segments. On utilise alors deux registres pour adresser une case mémoire (registre de segment combiné au registre d'offset), on appelle donc une **adresse logique :** le couple : **segment : offset.** On appelle **adresse physique :** L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits.

Adresse physique = segment x 16+ offset

Exemple : CS : IP = 3F51 : 0021 donc Adresse physique = 3F510 + 0021 = 3F530h.

II.4 Jeu d'instruction Intel 8086 :

II.4.1 Format d'une instruction

Le programme est une suite d'instruction. Une instruction est un mot binaire décrivant l'action que doit exécuter le microprocesseur, par exemple copier une case mémoire dans une autre, ajouter le contenu de deux cases mémoires etc.

Ces instructions sont des mots binaires qu'on écrit généralement en hexa, mais pour une plus grande facilité on les désignera par leurs « mnémoniques ». La mnémonique est une abréviation en caractère latin.

Une instruction est formée par un ou plusieurs octets. Le premier champ représente l'opération à effectuer et qui est appelée l'**opcode** .

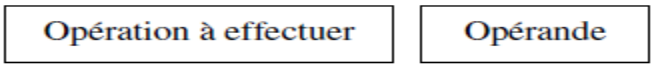


Figure (II.3) : Format d'une instruction

II.4.2 Les instructions de transfert

Elles permettent de déplacer des données d'une source vers une destination :

• registre vers mémoire ;

• registre vers registre ;

• mémoire vers registre.

Remarque : le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire (pour ce faire, il faut passer par un registre intermédiaire).

II.4.2.1 Instruction MOV

Le format général d'une instruction MOV est : **MOV Destination, Source**

(MOV Op1, Op2).

L'instruction MOV copie le contenu de l'opérande source dans l'opérande de destination.

La source peut être : Registre, mémoire, valeur immédiate (latérale).

La destination peut être : Registre, mémoire.

Toutes les combinaisons autorisées des deux opérandes sont énumérées ci-dessous:

MOV AX, BX copier un registre dans un autre.

MOV M, AX copier le contenu d'une case mémoire dans un registre.

MOV AX, M copier un registre dans une case mémoire.

MOV AX, 4 copier une constante dans un registre.

MOV taille M, 4 copier une constante dans une case mémoire. (taille = BYTE ou WORD)

Les actions suivantes ne **sont pas autorisées** avec l'instruction MOV :

✓ Interdit d'utiliser le registre (CS) comme destination.

✓ Interdit de lire ou écrire dans le registre (IP).

✓ Interdit de copier la valeur d'un registre de segment vers un autre registre de segment, il faut passer par une case mémoire ou par un registre interne de 16 bits.

✓ Interdit de copier une valeur immédiate vers un registre de segment, il faut passer par un registre interne ou une case mémoire de 16 bits.

✓ Interdit de transférer le contenu d'une case mémoire vers une autre, il faut passer par un registre interne de même taille.

II.4.2.2 Instruction de pile

Données à partir de la pile mémoire selon le protocole LIFO (dernier entré, premier sorti).

Le processeur 8086 prend en charge 4 instructions de pile:

PUSH

POP

PUSF (Push Drapeau enregistrer)

POPF (Pop Drapeau enregistrer)

a/ PUSH

Le format général d'une instruction de PUSH est : **PUSH opérande**

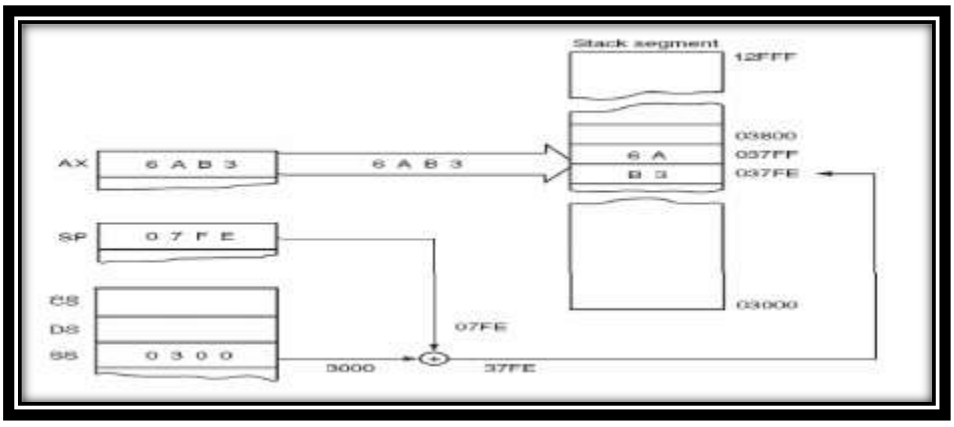


Figure (II.4) : Fonctionnement de l'instruction PUSH

- L'instruction PUSH stocke le contenu de l'opérande Op (16 bits) dans le sommet de Pile
- Décrémente SP de 2

Les opérandes admis de l'instruction PUSH sont :

PUSH AX registre

PUSH word [@] une case mémoire de 16 bits.

b/ POP

Le format général d'une instruction POP est : **POP opérande**.

- L'instruction POP Dépile une valeur de 16 bits depuis le sommet de la Pile (STACK SEGMENT) et la stocke dans les opérandes (16 bits).

- Les opérandes admis de l'instruction POP sont:

POP AX registre

POP word [@] une case mémoire de 16 bits.

Les instructions de pile sont des instructions importantes qui stockent et récupérent des

- Incrémente SP de 2

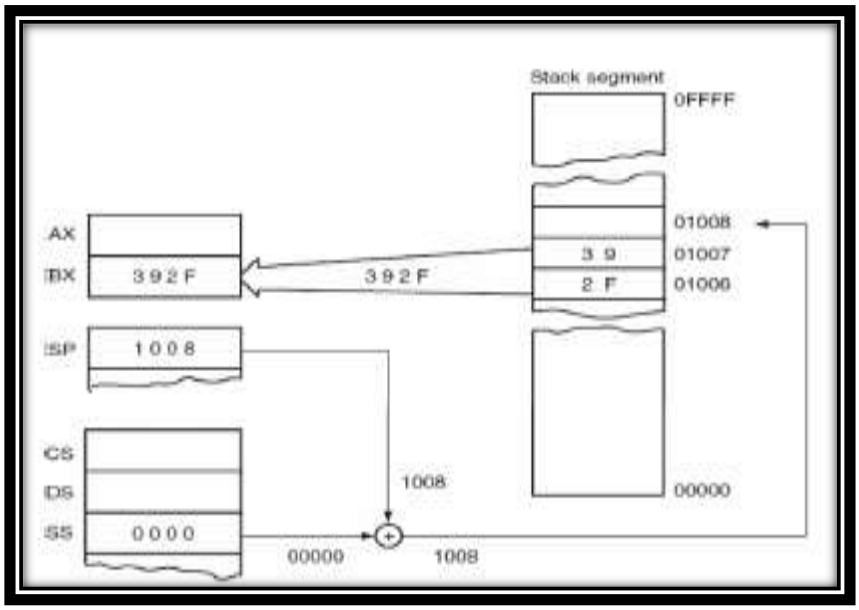


Figure (II.5) : Fonctionnement de l'instruction POP

II.4.3 Les instructions arithmétiques

Avec le 8086 on a les instructions arithmétiques de base : l'addition, la soustraction, la multiplication et la division. Qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles. Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signés ou non signés.

1) Instruction ADD

La forme générale de l'instruction ADD est : **ADD destination, source.**

L'instruction ADD Additionne l'opérande source et l'opérande destination avec résultat dans l'opérande destination, Destination destination + source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Toutes les combinaisons autorisées des deux opérandes sont énumérées ci-dessous:

Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

Exemple : ADD AX, 4 ADD AX, BX ADD [@], BX ADD AX, [SI]

2) Instruction ADC

L'instruction ADC Additionne l'opérande source, l'opérande destination et le curry avec résultat dans l'opérande destination :

ADC Destination, Source ; Destination destination + source + CF.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF

3) Instruction INC

Incrémentation d'un opérande.

INC Destination ; Destination destination+1.

La destination peut être : registre, mémoire.

Les FLAGs affectés : OF ; AF ; SF ; ZF ; PF

CF reste inchangé

Pour incrémenter une case mémoire, il faut préciser la taille :

INC byte [ ], INC word [ ]

4) Instruction SUB

La forme générale de l'instruction SUB (Soustraction) est : **SUB destination, source.**

L'instruction SUB Soustrait l'opérande source et l'opérande destination et place le résultat dans l'opérande de destination. Destination destination – source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

5) Instruction SBB

L'instruction SBB fait la soustraction entre deux opérandes avec carry.

SBB Destination, Source ; Destination destination – source – CF.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

6) Instruction DEC

Décrémentation d'un opérande : **DEC Destination** ; Destination :destination – 1.

La destination peut être : registre, mémoire.

Les FLAGs affectés : OF ; AF ; SF ; ZF ; PF.

CF reste inchangé.

7) Instruction NEG

L'instruction NEG donne le complément à 2 de l'opérande Op : remplace Op par son négatif : NEG Destination ; Destination Cà2 (destination).

La destination peut être : registre, mémoire.

Les FLAGs affectés : CF ; OF; AF; SF ; ZF; PF.

8) Instruction CMP

L'instruction CMP fait la comparaison entre deux opérandes et positionne les drapeaux en fonction du résultat. L'opérande Op1 n'est pas modifié.

CMP Op1, Op2 ; Résultat Op1 – Op2.

Op1 peut être : registre, mémoire.

Op2 peut être : registre, mémoire, valeur immédiate.

Les FLAGs affectés : CF ; OF ; AF ; SF ; ZF ; PF.

10) Instruction MUL

Cette instruction exécute la multiplication entre l'accumulateur et un opérande (non signé) : **MUL Opérande**.

Si l'opérande est une valeur sur 8 bits : AX AL * Opérande.

Si l'opérande est une valeur sur 16 bits : DX : AX AX * Opérande.

L'opérande ne peut pas être une donnée, c'est soit un registre soit une position mémoire, dans ce dernier cas, il faut préciser la taille (byte ou word)

Les FLAGs affectés : CF et OF si la partie haute du résultat est non nulle. La partie haute est AH pour la multiplication 8 bits et DX pour la multiplication 16 bits

Exemple :

MUL BL ; AL x BL → AX

MUL CX ; AX x CX → DX:AX

11) Instruction DIV

Cette instruction exécute la Division entre l'accumulateur et un opérande (non signé) : **DIV Opérande.**

Si l'opérande sur 8 bits : AL AX / Opérande ; AH Reste.

Si l'opérande sur 16 bits : AX (DX : AX) / Opérande ; DX Reste.

L'opérande doit être soit un registre soit une mémoire. Dans le dernier cas il faut préciser la taille de l'opérande, exemple : DIV byte [adresse] ou DIV word [adresse].

Les FLAGs : CF ; OF ; ZF ; SF ; PF ; et AF sont inconnus (X).

12) Instructions IMUL et IDIV

La multiplication et la division de l'accumulateur avec un opérande (signé).

Les mêmes syntaxes que MUL et DIV.

III.4.4 Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont : ET ; OU ; OU exclusif ; décalages et rotations.

1) Instruction AND

ET logique entre deux opérandes bit à bit : **AND Destination, Source ;**

Destination destination (ET) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAGs affectés : ZF ; SF ; PF

2) Instruction OR

OU logique entre deux opérandes bit à bit : **OR Destination, Source ;**

Destination destination (OU) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAGs affectés : ZF ; SF ; PF

3) Instruction XOR

OU Exclusif entre deux opérandes bit à bit : **XOR Destination, Source ;**

Destination destination (ou exclusif) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAGs affectés : ZF ; SF ; PF

Remarque : Astuces sur les instructions logiques (OR, XOR, AND) :

1. Forçage à 1 : c'est le OR avec un « 1 ».
2. Masquage à 0 : c'est le AND avec un « 0 ».
3. Inverser un bit : c'est le XOR avec un « 1 ».

4) Instruction TEST

L'instruction TEST effectue l'opération ET.

La différence est que l'instruction AND change l'opérande de destination, alors que l'instruction TEST positionne uniquement les FLAG.

TEST Op1, Op2 ; Résultat Op1 (ET) Op2.

Op1 peut être : registre, mémoire.

Op2 peut être : registre, mémoire, immédiate.

Les FLAGs affectés : ZF ; SF ; PF

5) Instruction NOT

Complément à 1 de l'opérande : NOT **Destination.**

La destination peut être : registre, mémoire.

6) Instructions de Décalages

Ces instructions déplacent d'un certain nombre de positions les bits d'un mot vers la gauche ou vers la droite.

Dans les décalages, les bits qui sont déplacés sont remplacés par des zéros. Il y a les décalages logiques (opérations non signées) et les décalages arithmétiques (opérations signées).

Le 8086 prend en charge quatre instructions de décalage différentes :

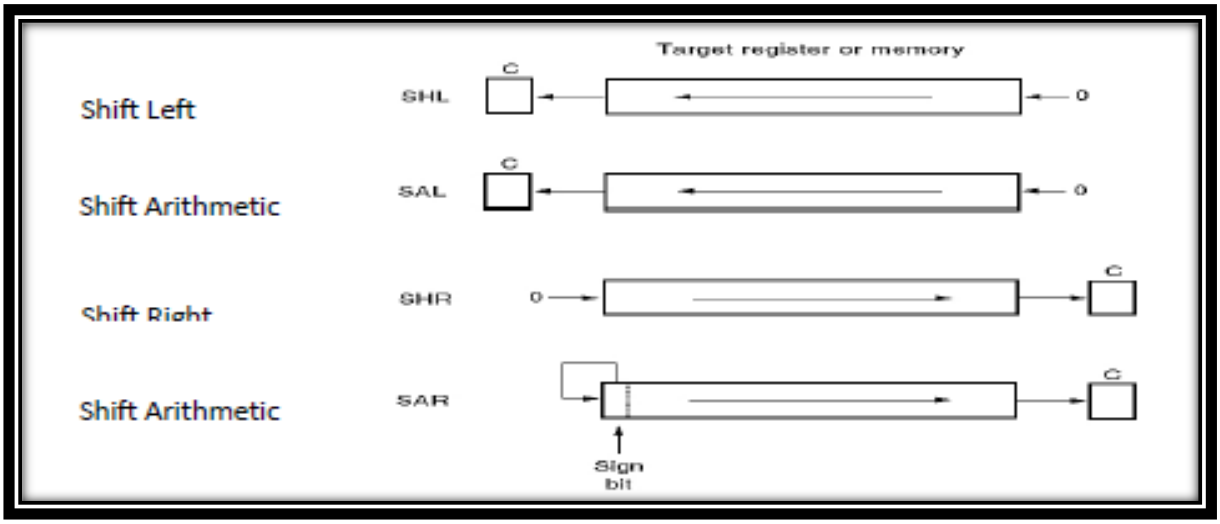


Figure (II.6) : Les instructions de décalage

Remarque : Les décalages arithmétiques permettent de conserver le signe. Ils sont utilisés pour effectuer des opérations arithmétiques comme des multiplications et des divisions par 2.

La forme générale des instructions de décalage est : **XXX destination, N** Où: XXX est le mnémonique de décalage (i.e., SHL, SHR, SAL, SAR), La destination peut être : registre, mémoire et l'opérande *N* peut être soit une constante (immédiat) soit le registre CL :

Exemples: SHL AX, 1

SHL AX, CL

7) Instructions de Rotations

Dans les rotations, les bits déplacés dans un sens sont réinjectés de l'autre côté du mot. Les quatre opérations de rotation disponibles apparaissent dans la figure suivante :

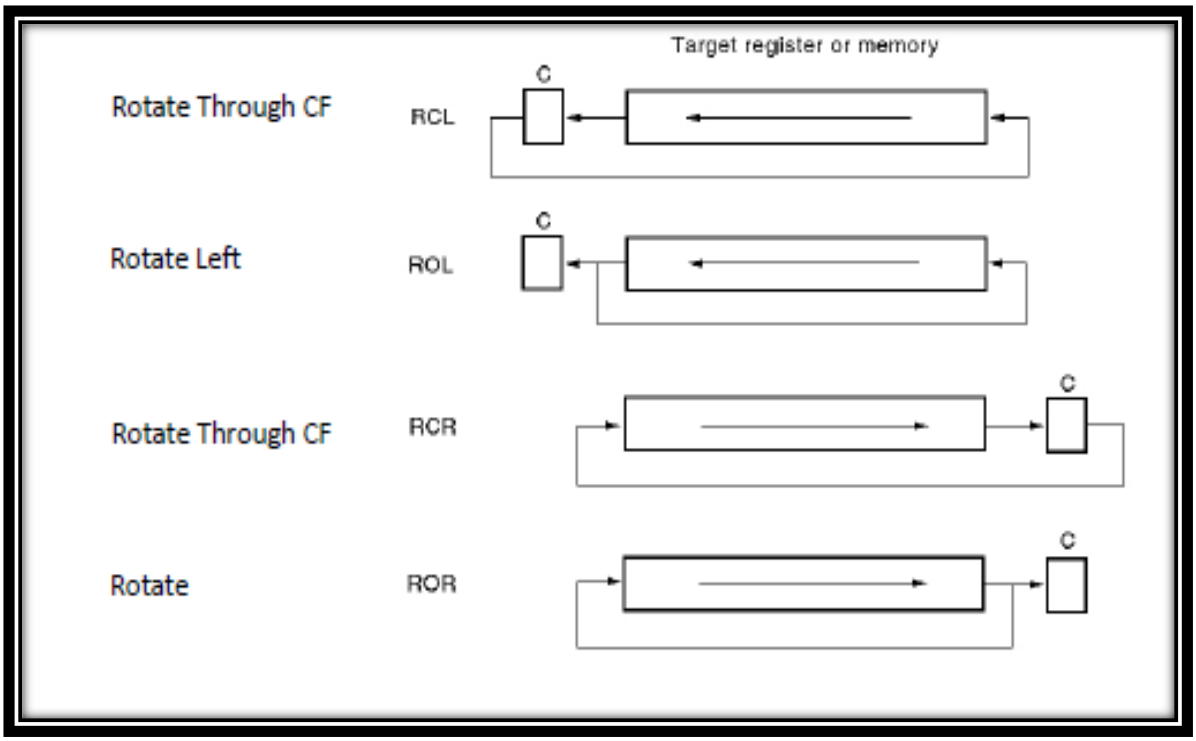


Figure (II.7) : instructions de rotation du registre ou de la mémoire indiquant la direction et le fonctionnement de chaque rotation**.**

La forme générale des instructions de rotation est : **XXX destination, N**

Où: XXX est la mnémonique de rotation (i.e., RCL, RCR, ROL, ROR),

Exemples: ROL AX, 1

ROL AX, CL

II.4.5 Les instructions de branchement

Les instructions de branchement (ou saut) permettent de faire un saut dans le programme, et exécuter une instruction beaucoup plus éloignée, ou au contraire revenir en arrière. Il existe 3 types de saut :

• saut inconditionnel ;

• sauts conditionnels ;

• appel de sous-programmes.

a) Instruction de saut inconditionnel

JMP label ; Provoque un saut sans condition à la ligne portant l'étiquette label.

Où label est un identificateur d'adresse de programme.

b) Instructions de sauts conditionnels

Instructions de sauts conditionnels sont conditionnées par l'état des indicateurs (drapeaux) qui sont eux même positionnés par les instructions précédentes.

Jxxx label où: xxx décrit la condition du saut

Tableau 1: Liste des instructions de sauts conditionnels du 8086.

	Instruction	Description	Tested Condition
1	JA	Jump if above	CF = 0 and ZF = 0
2	JNBE	Jump if neither bellow nor equal	CF = 0 and ZF = 0
3	JBE	Jump if bellow or equal	CF = 1 or ZF = 1
4	JNA	Jump if not above	CF = 1 or ZF = 1
5	JAE	Jump if above or equal	CF = 0
6	JNB	Jump if not bellow	CF = 0
7	JB	Jump if bellow	CF = 1
8	JNAE	Jump if neither above nor equal	CF = 1
9	JG	Jump if greater than	ZF = 0 and SF = OF
10	JNLE	Jump if not less nor equal	ZF = 0 and SF = OF
11	JLE	Jump if less or equal	ZF = 1 or SF ≠ OF
12	JNG	Jump if not greater	ZF = 1 or SF ≠ OF
13	JGE	Jump if greater than or equal	SF = OF
14	JNL	Jump if not less than	SF = OF
15	JL	Jump if less than	SF ≠ OF
16	JNGE	Jump if neither greater nor equal	SF ≠ OF
17	JO	Jump if overflow	OF = 1
18	JNO	Jump if no overflow	OF = 0
19	JZ	Jump if zero	ZF = 1
20	JE	Jump if equal	ZF = 1
21	JNZ	Jump if not zero	ZF = 0
22	JNE	Jump if not equal	ZF = 0
23	JC	Jump on carry	CF = 1
24	JNC	Jump on not carry	CF = 0
25	JS	Jump if sign (negative)	SF = 1
26	JNS	Jump if no sign (not negative)	SF = 0
27	JP	Jump if parity	PF = 1
28	JPE	Jump if even parity	PF = 1
29	JNP	Jump if no parity	PF = 0
30	JPO	Jump if parity odd	PF = 0
31	JCXZ	Jump if CX is zero	CX = 0

Remarque :

- Les termes supérieur et inférieur (above and below) pour les nombres non signés.
- Les termes plus grand et plus petit (greater than and less than) pour les nombres signés.

3) Instruction CALL

Appel d'une procédure (sous-programme) : CALL **nom_ sous-programme**

Le CPU fait plusieurs choses quand une instruction CALL est exécutée:

(1) Il ajuste le contenu du registre IP comme s'il va exécuter l'instruction suivante.

(2) Il stocke le contenu du registre IP dans la pile de programme.

(3) On ajuste le contenu du registre d'IP de nouveau pour pointer sur la première instruction de la procédure appelée.

L'unité centrale continue à exécuter les instructions jusqu'à ce qu'elle rencontre une instruction RET.

4) Instruction RET

Un RET (Return) instruction de retour de sous-programme : RET

Quand le CPU rencontre une instruction RET, il effectue les actions suivantes :

(1) Il récupère le contenu du registre IP stocké dans la pile de programme par l'instruction CALL.

(2) Il met la valeur restaurée dans le registre IP et continue l'exécution.

Le CPU revient à l'état où il était lorsque l'instruction CALL a été rencontrée.

5) Instructions des interruptions

INT n: Appel à l'interruption logicielle n° n

IRET : Cette instruction termine un sous-programme de traitement d'une interruption.

6) Instruction de boucles LOOP

L'instruction LOOP décrémente le contenu de CX et provoque un saut relatif court si ce dernier n'est pas nul.

La forme générale d'une instruction de boucle est : **LOOP label**

Où label est un identificateur d'adresse de programme.

L'instruction LOOP est équivalente au deux instructions :

DEC CX

JNZ label

La diférence entre les deux instructions ci-dessus et de l'instruction LOOP est que cette dernière, n'affecte pas les états des flags.

Le 8086 à 4 variations de l'instruction LOOP:

LOOPZ (Loop While Zero) ou **LOOPE** (Loop While Equal) : si le flag ZF=1 et si CX ≠ 0.

LOOPNZ (Loop While Not Zero) ou **LOOPNE** (Loop While Not Equal) : si le flag ZF=0 et si CX ≠ 0.

Ces variations permettent le déroulement du programme pour deux conditions (le registre CX et le drapeau zéro (ZF) en même temps).

II.4.6 Les instructions de manipulation de chaines

Dans le CPU 8086 on a 5 instructions de manipulation de chaînes :

MOVS, LODS, STOS sont des instructions de transfert, elles peuvent être répétées à l'aide du préfixe **REP**

CMPS et SCAS sont des instructions de comparaisons, elles peuvent être répétées à l'aide du préfixe **REPZ**

Remarque : Chacune des instructions existe en deux versions, entre deux octets, ou entre 2 words, le préfixe REP pour les instructions de transfert MOVS LODS STOS, Ce préfixe utilise le registre CX comme un compteur de répétition.

Les préfixes REPE et REPZ : Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=1 et CX≠0.

Les préfixes REPNE et REPNZ : Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=0 et CX≠0.

Ces instructions sont utilisées sans opérandes. les registres utilisés sont : pour la source DS : SI, et pour la destination ES:DI. A chaque exécution d'une instruction de traitement de chaîne, les registres d'indexes sont incrémentés ou décrémentés selon le flag DF de registre d'état.

* DF = 0 : SI et DI sont incrémentés, (l'instruction CLD mit (DF = 0))

* DF = 1 : SI et DI sont décrémentés. (l' instruction STD mit (DF = 1))

SI et DI sont incrémentés de 1 ou de 2 selon que l'opération s'effectue sur un octet (byte) ou sur un mot de 16 bits (word).

MOVSB : Copie Le contenu d'un octet de mémoire DS: [SI] dans un octet de mémoire ES: [DI], puis auto inc/decrémente les registres SI et DI.

MOVSW : Copie Le contenu du mot de mémoire DS: [SI] dans le mot de mémoire ES: [DI], puis auto inc/decrémente de 2 les registres SI et DI

LODSB : Copie l'octet source DS: [SI] dans AL puis inc/décr le registre SI.

LODSW : Copie le mot source DS: [SI] dans AX puis inc/décr de 2 le registre SI.

STOSB : Copie AL dans l'octet destination ES: [DI] et inc/décr le registre DI.

STOSW : Copie AX dans le mot destination ES: [DI] et inc/décr de 2 le registre DI.

CMPSB : Compare l'octet source ES : [DI] avec l'octet destination DS: [SI], positionne les indicateurs puis inc/décrémente les registres SI et DI.

CMPSW : Compare le mot source ES : [DI] avec mot destination DS: [SI], positionne les indicateurs puis inc/décrémente de 2 les registres SI et DI

SCASB : Compare AL avec l'octet destination ES : [DI], positionne les indicateurs puis inc/décr le registre DI.

SCASW : Compare AX avec le mot destination ES : [DI], positionne les indicateurs puis inc/décr le registre DI de 2.

II.5 Adressage

II.5.1 Adressage registre :

Ce mode utilise les registres internes de µp, dans ce mode if faut respecté la taille de différents registres.

Exemple :

Dans le plus part du cours, on va utiliser l'instruction MOV (instruction de transfert) :

Syntaxe :

MOV Destination, Source ; Destination Source.

MOV BX, AX ; Transfert du contenu de AX vers BX.

MOV BX, AL ; Déclaration d'erreur par ce que BX est registre 16 bits et ; AL est registre 8 bits.

MOV BL, BH ; Transfert du contenu de BH vers BL.

II.5.2 Adressage immédiat :

Est un mode de manipulation des valeurs latéraux avec un

registre interne ou une case mémoire.

Exemple :

MOV Alpha, 40h ; Alpha est une case mémoire de 8 bits.

Exemple :

MOV Alpha, 40h ; Alpha est une case mémoire de 8 bits.

MOV AL, 102h ; déclaration d'une erreur, AL registre de 8 bits, mais 102h est ; une valeur immédiate sur 16 bits.

MOV AX, 40h ; cette instruction est juste ; Par ce que, on peut écrire 40h =0040h valeur sur 16 bits.

II.5.3 Adressage direct :

Comme l'indique son nom, c'est la réalisation des opérations entre les registres internes de µp et la mémoire.

Exemple :

MOV AL, Bêta ; transférer le contenu de la case mémoire Bêta vers AL.

MOV (2000h), BL ; transférer l contenu de BL vers la case mémoire ; 2000h par rapport au DATA segment

II.5.4 Adressage indirect par registre :

C'est le mode d'accès entre la mémoire et les registres internes de µp par un intermédiaire, ces sont les registres pointeurs : (SI, DI, BX).

Exemple :

MOV AX, [BX] ; transfert de contenu de la case mémoire 16 bits pointer par BX ; par rapport au DATA segment vers AX.

MOV [SI], DL ; transfert de contenu de DL vers la case mémoire 8 bits pointer ; par SI par rapport au DATA segment.

II.5.5 Adressage basé :

Tous simplement, il utilise la registres de base comme pointeur entre la mémoire et les registres internes de µp ; les registres de base sont (BX, BP).

Syntaxe :

MOV [BX/BP + déplacement 8 bits ou 16 bits], source.

MOV destination, [BX/BP + déplacement 8 bits ou 16 bits].

Exemple :

MOV [BX+12], AL ; transfert de contenu de AL vers la case mémoire pointer par ; (BX+12) par rapport au DATA segment.

MOV SI, [BP+400] ; transfert de contenu de la case mémoire 16 bits pointer par ; (BP+400) par rapport au STACK segment vers SI.

II.5.6 Adressage indexé :

Le même mode que le basé, mais il utilise les registres d'indexes comme pointeur ; les registres d'indexes sont (SI, DI).

Syntaxe :

MOV [SI/DI + déplacement 8 bits ou 16 bits], source.

II.5.7 Adressage basé indexé:

C'est un mode combine entre de modes d'adressage comme l'indique son nom.

Exemple : MOV AX, [BP+SI-8] : AX est chargé par la mémoire d'adresse SS : BP+SI-8

MOV AX, [BX+DI+4] : AX est chargé par la mémoire d'adresse DS : BX+DI+4

II.6 Les interruptions :

Le mécanisme d'interruptions sera décrit en détail dans ce partie.

Sachons simplement ici qu'une interruption est une espéce de sous-programme pré-existant dans la machine. L'appel se fait via l'instruction int 21h. Le registre ah contient un numéro qui référence la fonctionnalité que l'on veut utiliser (9 pour afficher une chaine de caractères, 1 pour saisir la frappe d'un caractère au clavier, ...)

II.6.1 Affichages des caractères à l'écran

II.6.1.1 Affichages des caractères à l'écran

```
mov dl, 'A'
mov ah, 2
int 21h
```

Affiche le caractère A à l'écran.

II.6.1.2 Afficher un message à l'écran

```
.data
msg db 'hello world', 13, 10, '$'
.code
mov ax, @data
mov ds, ax
```

```
mov ah, 9
lea dx, msg
int 21h
mov ah, 4ch
int 21h
end
```

Affiche le message hello world `a l'écran. La chaine de caractères doit impérativement se terminer par un caractère '$'. Les caractères 13 et 10 sont les codes ASCII des 2 caractères constituant un retour-chariot.

II.6.2 Saisir une frappe au clavier

II.6.2.1 Saisie avec écho

Saisie d'un caractére

```
mov ah, 1
int 21h
```

Renvoie dans le registre al le code du caractére lu au clavier.

Saisie d'une chaine de caractères

Saisie d'une chaine de caractères

```
.data
msg db 80 dup (?)
.code
mov ax, @data
mov ds, ax
mov bx, 0
mov cx, 80
lea dx, msg
mov ah, 3fh
int 21h
```

Saisit une chaine de caractères au clavier d'au plus cx caractères et la stocke dans la variable msg qui est déclarée comme un tableau de 80 octets (soit 80 caractères). Le nombre de caractères lus est placé dans le registre ax. Notons que le retour-chariot est stocké sous forme de deux caractères successifs : le premier de code 13, le second de code 10.

II.6.2.2 Saisie sans écho

```
mov ah, 8
int 21h
```

Lit un caractère au clavier et le renvoie dans le registre al. Ce caractère n'est pas affiché à l'écran.

II.6.3 L'accès aux fichiers

Nous présentons ici les interruptions `a utiliser pour effectuer les traitements de base sur les fichiers. On se reportera à 6.3 pour un exemple complet de traitement de fichiers.

II.6.3.1 Principe :

En assembleur, les fichiers sont traités d'une maniéré assez similaire à un langage évolué comme Pascal. Pour pouvoir accéder aux données d'un fichier, que ce soit en lecture ou en écriture, il faut préalablement l'ouvrir. `A l'ouverture, le programme récupère une poignée. Ensuite, toutes les opérations sur ce fichier seront effectuées en spécifiant cette poignée. `A un instant donné, plusieurs fichiers peuvent avoir et ouverts par un programme. Chacun a une poignée différente laquelle permet au programme de les distinguer.

II.6.3.2 Création d'un fichier

Avant l'appel

☞ cx contient la valeur 0 ;

☞ ds: dx pointent sur le nom de Fichier . C'est une chaine de caractères terminée par un caractère ASCII de code 0.

L'appel a le format suivant :

```
mov ah, 3ch
int 21h
```

Après l'appel

Le bit C du registre flags vaut 1 si une erreur est survenue.

II.6.3.3 Ouverture d'un fichier

Avant l'appel

– al contient le mode d'accès :

0 pour lecture, 1 pour l'écriture,

2 pour les deux ;

– ds : dx pointent sur le nom de fichier. C'est une chaine de caractères terminée par un caractère ASCII de code 0.

L'appel a le format suivant :

```
mov ah, 3dh
int 21h
```

Après l'appel

Le bit C du registre flags vaut 1 si une erreur est survenue. Sinon, le registre ax contient la poignée sur le fichier ouvert

II.6.3.4 Fermeture d'un fichier

Avant l'appel

– bx contient la poignée du fichier à Fermer

L'appel a le format suivant :

mov ah, 3eh

int 21h

Après l'appel

Le bit C du registre flags vaut 1 si une erreur est survenue. Sinon, le fichier est fermé. Il ne peut plus être accédé via sa poignée. Si on veut l'accéder à nouveau, il faudra le ré-ouvrir au préalable.

II.6.3.5 Lecture dans un fichier

Avant l'appel :

– bx contient la poignée du fichier ;

– cx contient le nombre de caractères à lire ;

– ds:dx contient l'adresse du tampon ou il faut stocker les caractères lus.

On peut utiliser cet appel pour lire une chaine de caractères tapés au clavier. Pour cela, il faut que bx contienne la valeur 0 en entrée. La lecture s'effectuera sur au plus cx caractères et au plus jusqu'à la frappe d'un retour-chariot.

L'appel a le format suivant :

mov ah, 3fh

int 21h

Après l'appel

Le bit C du registre flags vaut 1 si une erreur est survenue. Sinon, le registre ax contient le nombre de caractères lus. En principe, c'est le nombre de caractères qui devait être lus à moins que la fin de fichier ne soit atteinte.

II.6.3.6 ´Ecriture dans un fichier

Avant l'appel

– bx contient la poignée du fichier ;

– cx contient le nombre de caractères à écrire ;

– ds:dx pointent sur le tampon contenant les caractères à écrire.

L'appel a alors le format suivant :

mov ah, 40h

int 21h

Après l'appel

Le bit C du registre flags vaut 1 si une erreur est survenue. Sinon, le registre ax contient le nombre de caractères écrits.

II.6.4 Lire l'heure courante

La fonction 2ch de l'interruption 21h lit l'heure courante, telle qu'elle est stockée dans l'ordinateur (s'il n'est pas à l'heure, cela ne donnera pas la bonne heure).

mov ah, 2ch

int 21h

Au retour de l'appel, les registres contiennent les informations suivantes :

ch heures

cl minutes

dh secondes

dl centiémes de secondes

II.6.5 Lire la date courante

La fonction 2ah de l'interruption 21h lit la date courante, telle qu'elle est stock´ee dans l'ordinateur

(comme pour l'heure, si la date n'est pas bonne, cela ne donnera pas la bonne date).

mov ah, 2ah

int 21h

Au retour de l'appel, les registres contiennent les informations suivantes :

al jour de la semaine codé (0 : dimanche, 1 : lundi, ...)

cx année

dh mois

dl jour

.

CHAPITRE III

III.1 Introduction :

Une mémoire est un circuit à semi-conducteur permettant d'enregistrer, de conserver et de restituer des informations (instructions et variables). C'est cette capacité de mémorisation qui explique la polyvalence des systèmes numériques et leur adaptabilité à de nombreuses situations. Les informations peuvent être écrites ou lues. Il y a écriture lorsqu'on enregistre des informations en mémoire, lecture lorsqu'on récupère des informations précédemment enregistrées.

III.2 Organisation d'une mémoire :

Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs. Chaque tiroir représente alors une case mémoire qui peut contenir un seul élément : des **données**. Le nombre de cases mémoires pouvant être très élevé, il est alors nécessaire de pouvoir les identifier par un numéro. Ce numéro est appelé **adresse**. Chaque donnée devient alors accessible grâce à son adresse.

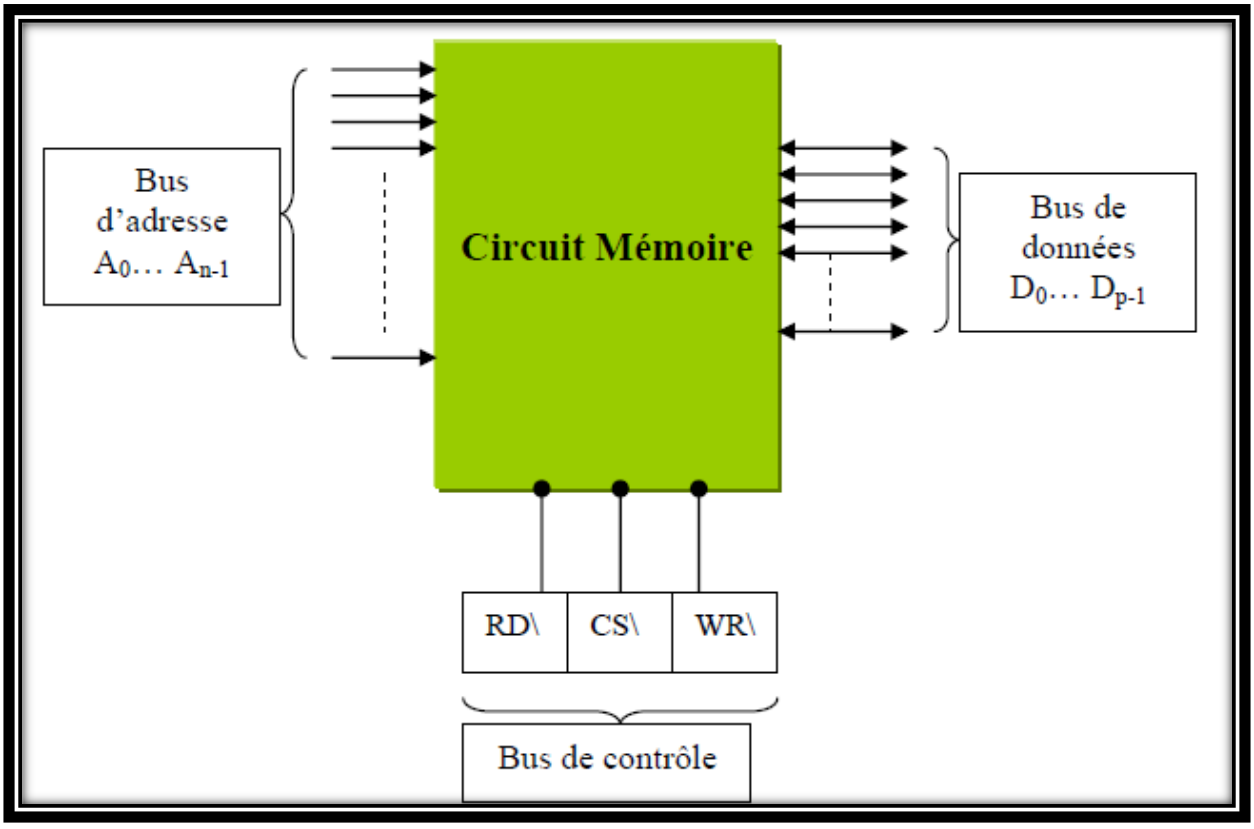


Figure (III.1) : organisation global de la mémoire

RD : signal de lecture case mémoire.
WR : signal d'écriture case mémoire.
CS : signal de validation de circuit mémoire ou boîtier.

Avec une adresse de *n* bits il est possible de référencer au plus 2^n cases mémoire. Chaque case est remplie par un mot de données (sa longueur *m* est toujours une puissance de 2). Le nombre

de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire. En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.

On peut donc schématiser un circuit mémoire par la figure (III.1) où l'on peut distinguer :

III.3 Caractéristiques d'une mémoire :

Capacité	C'est le nombre total de bits. Elle s'exprime aussi souvent en octet.
Format de données	C'est le nombre de bits que l'on peut mémoriser par case mémoire. On dit aussi que c'est la largeur du mot mémorisable.
Temps d'accès	C'est le temps écoulé entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
Temps de cycle	Il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.
Débit	C'est le nombre maximum d'informations lues ou écrites par seconde.
Volatilité	Elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

III.4 Réfléchissement de mémoire (Chronogrammes de lecture/écriture en mémoire)

Une caractéristique importante des mémoires est leur temps d'accès : c'est le temps qui s'écoule entre l'instant ou l'adresse de la case mémoire est présentée sur le bus d'adresses et celui ou la mémoire place la donnée demandée sur le bus de données. Ce temps varie entre 50 ns (mémoires rapides) et 300 ns (mémoires lentes).

Chronogramme de lecture en mémoire :

Remarque : si le temps d'accès d'une mémoire est supérieur à une période d'horloge (mémoire lente), le microprocesseur peut accorder `a la mémoire un temps supplémentaire (une ou plusieurs périodes d'horloge), à la demande de celle-ci. Ce temps supplémentaire est appelé **temps d'attente** (wait time : *TW*) :

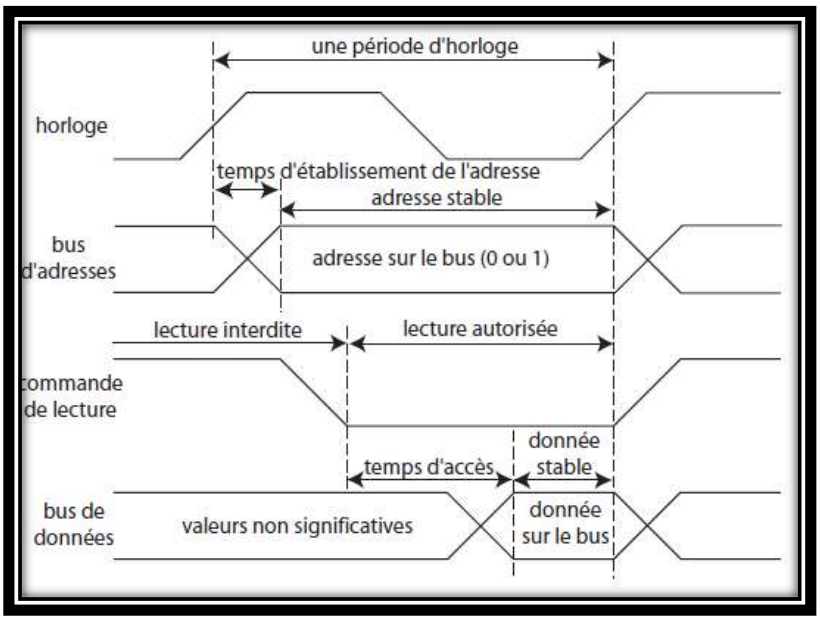


Figure (III.2) : Chronogramme de lecture en mémoire

Remarque : si le temps d'accès d'une mémoire est supérieur à une période d'horloge (mémoire lente), le microprocesseur peut accorder `a la mémoire un temps supplémentaire (une ou plusieurs périodes d'horloge), à la demande de celle-ci. Ce temps supplémentaire est appelé **temps d'attente** (wait time : *TW*) :

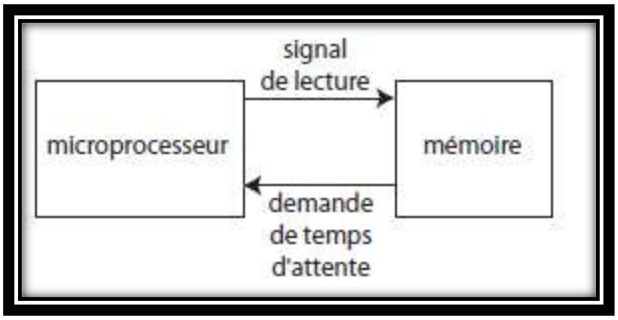


Figure (III.3) : temps de l'attente de la mémoire

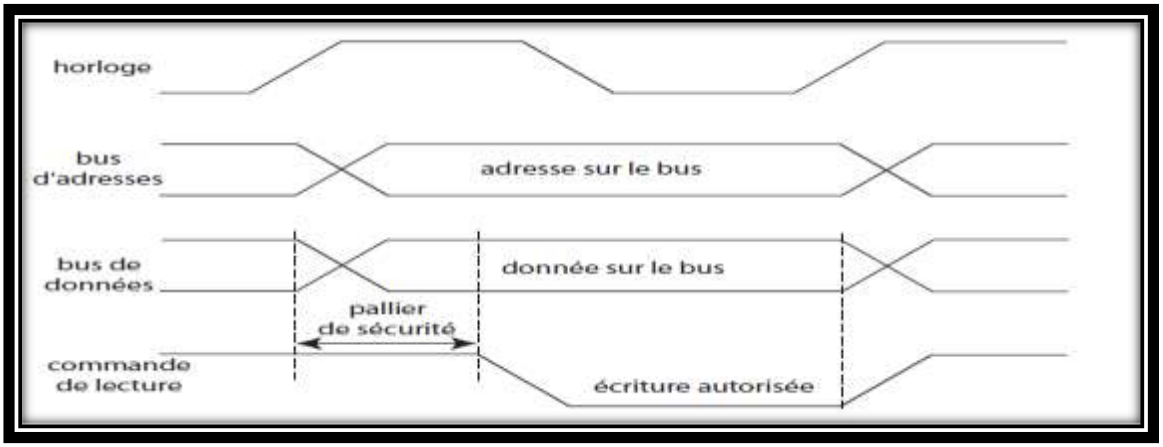


Figure (III.4) : Chronogramme d' écriture en mémoire

III.5 Différents types de mémoire :

III.5.1 Mémoires vives (RAM) :

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure d'alimentation. Certaines d'entre elles, ayant une faible consommation, peuvent être rendues non volatiles par l'adjonction d'une batterie. Il existe deux grandes familles de mémoires RAM (RANDOM ACCESS MEMORY : mémoire à accès aléatoire) :

- Les RAM statiques.
- Les RAM dynamique.

☞ **Les RAM statiques** : le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule, chaque bascule contient entre 4 et 6 transistors.

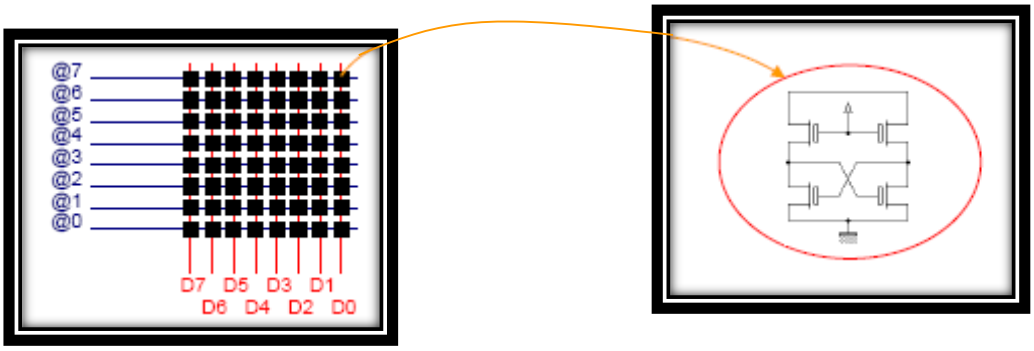


Figure (III.5) : Schéma synoptique d'un RAM statique

☞ **Les RAM dynamique :** dans cette mémoire, l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur.

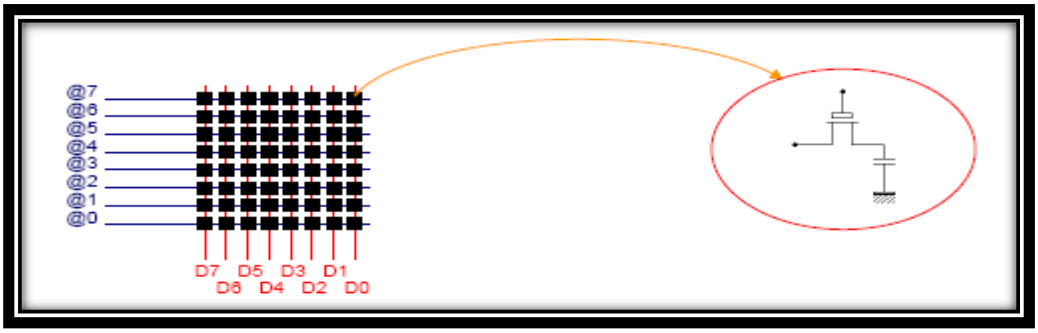


Figure (III.6) : Schéma synoptique d'un RAM dynamique

➢ **Avantages :**

Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite.

➢ **Inconvénients :**

La présence de courants de fuite dans le condensateur contribue à sa décharge.

Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur). Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger. Ce rafraîchissement indispensable a plusieurs conséquences :

- Il complique la gestion des mémoires dynamiques car il faut tenir compte des actions de rafraîchissement qui sont prioritaires.
- La durée de ces actions augmente le temps d'accès aux informations.

D'autre part, la lecture de l'information est destructive. En effet, elle se fait par décharge de la capacité du point mémoire lorsque celle-ci est chargée. Donc toute lecture doit être suivie d'une réécriture.

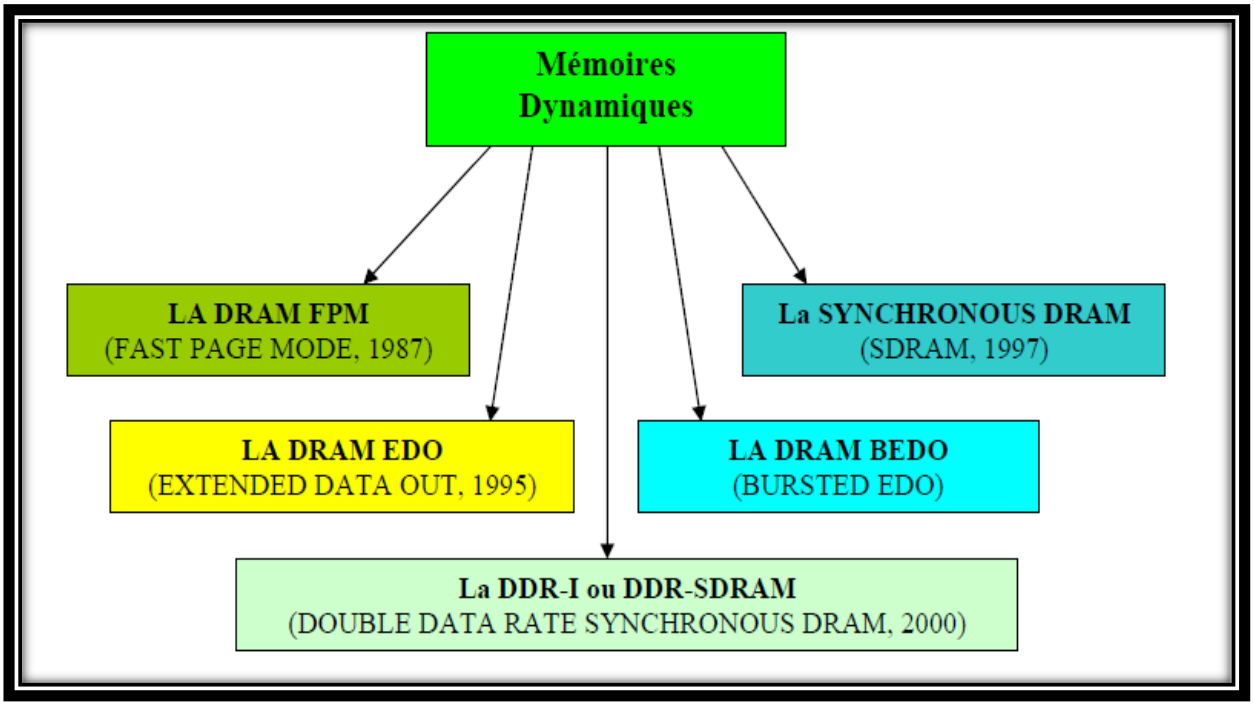


Figure (III.7) : Déférents types des mémoires dynamique

III.5.2 Les mémoires mortes (ROM) :

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule (ROM : READ ONLY MEMORY). Ces mémoires sont non volatiles.

Ces mémoires, contrairement aux RAM, ne peuvent être que lue. L'inscription en mémoire des données restent possible mais est appelée programmation. Suivant le type de ROM, la méthode de programmation changera. Il existe donc plusieurs types de ROM :

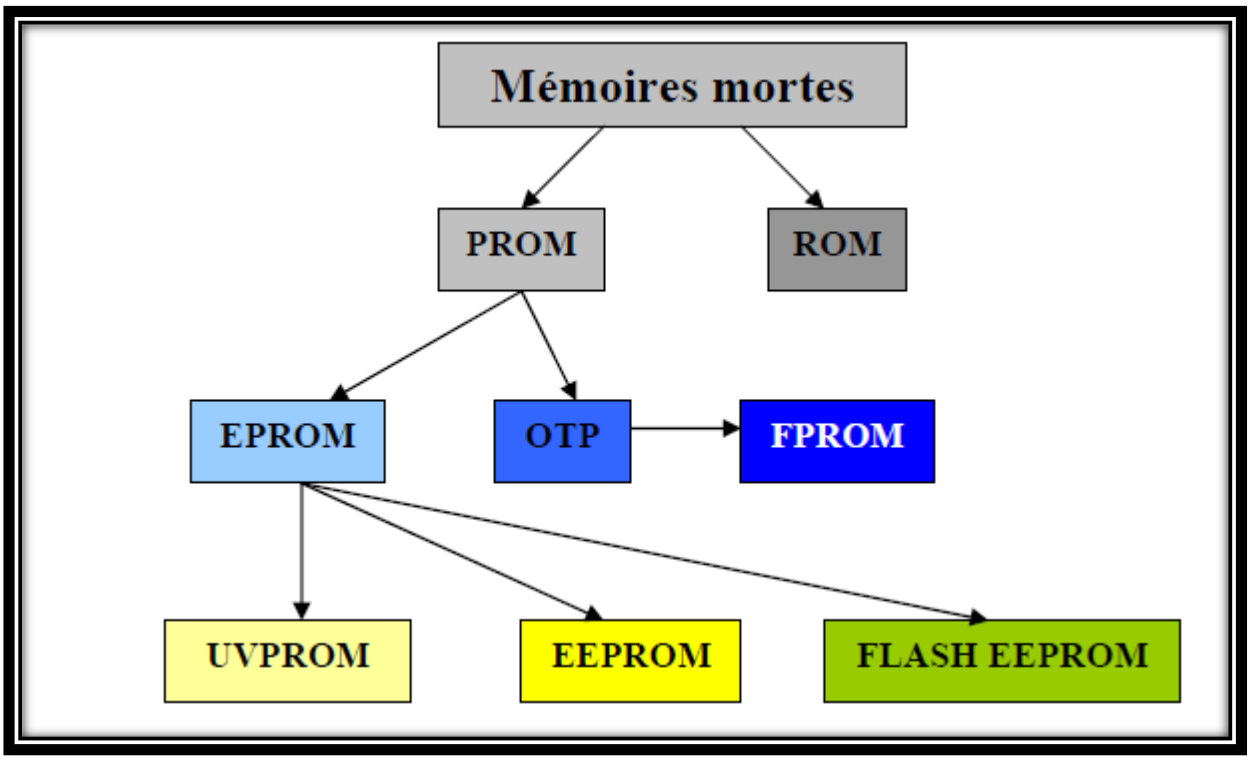


Figure (III.8) : Déférents types des mémoires ROM

☞ **Critères de choix d'une mémoire :**

Les principaux critères à retenir sont :

✓ Capacité.

✓ Vitesse.

✓ Consommation.

✓ Coût.

a) Les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.

b) **La mémoire cache** est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.

c) **La mémoire principale** est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.

d) **La mémoire d'appui** sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.

e) **La mémoire de masse** est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

III.6 l'interface d'un circuit mémoire au bus système 8086.

II.6.1 mode d'adressage et mapping de mémoire :

Après cette longue définition des différents types de mémoires, on passe au cœur du cours, c'est l'interface d'un circuit mémoire au bus système 8086.

Avant de commencer, il faut d'abord citer les différentes lignes utilisées pour interfacer un boîtier mémoire :

- ✓ Les lignes d'adresses A0 … A19.
- ✓ Les lignes de données D0 … 15.
- ✓ Le signal BHE\.
- ✓ Le signal M/IO\.
- ✓ Le signal RD\.
- ✓ Le signal WR\.

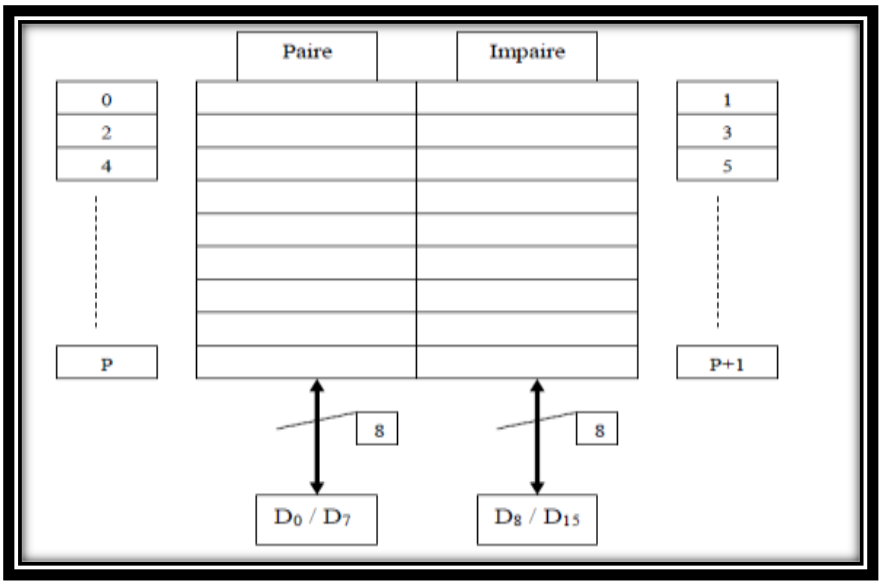


Figure (III.9) : Décomposition de mémoire (paire , impaire)

A partir de la taille de bus d'adresse de 20 bits, on distingue que l'espace mémoire adressable par le µp 8086 est de capacité 1 Mo = 2^{20} * 8. Cet espace est divisé en deux parties, partie paire de 512Ko et partie impaire de 512Ko. La partie paire est connectée au bus de données D0/D7, et la partie impaire est connectée au bus D8/D15.

Pour bien comprendre le fonctionnement de la ligne BHE\, on exécute les codes sources suivants :

MOV AL, (0002h) : voir la figure (III.10.a)

MOV AL, (0003h) : voir la figure (III.10.b)

MOV AX, (0004h) : voir la figure (III.10.c)

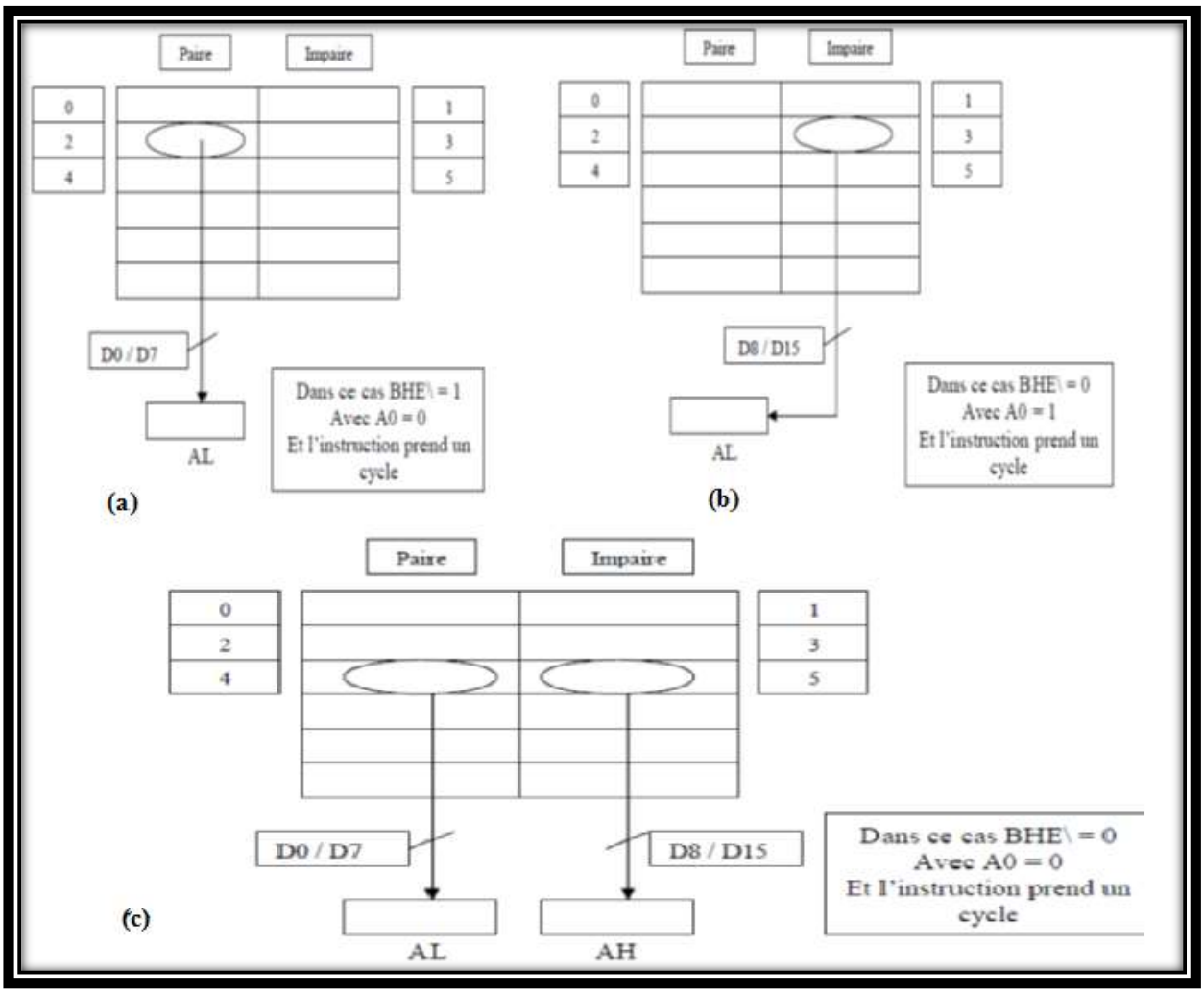


Figure (III.10) : Gestion de le mémoire (paire , impaire)

Il reste l'accès 16 bits adresse impaire, dans ce cas, on prend l'exemple :

MOV AX, (0001h)

Par analogie AL sera chargé par le contenu de l'adresse (0001h) et AH par le contenu de l'adresse (0002h), mais on remarque que ADR (0001h) est une adresse impaire on prend un cycle pour charger AL, et un autre cycle pour charger AH.

AL : (0001h) : A0 = 1 et BHE\ = 0. (1 cycle).

AH : (0002h) :A0 = 0 et BHE\ = 1. (1 cycle).

On tous deux cycles.

Exemple :

On passe maintenant après cette description de la ligne BHE\ à la liaison d'un bloc mémoire de taille 1Ko à l'adresse physique (00000h) à l'aide les boîtiers de RAM de taille 256*8.

Avant d'interfacer ce bloc, on va suivre les étapes suivantes :

1- Calculez le nombre de boîtiers utilisés pour réaliser le bloc mémoire demandé.

$$\text{Nombre de boitiers} = \frac{1.1024.8}{256.8} = 4$$

2- Calculez le nombre de lignes d'adresse du bloc demandé et de boîtier utilisé.
Nombre de lignes d'adresse demandé =

$$\frac{Ln(1.1024)}{Ln(2)} = 10 lignes$$

Nombre de lignes d'adresse utilisé =

$$\frac{Ln(256)}{Ln(2)} = 8 lignes$$

3- Fêtes le mapping suivant :

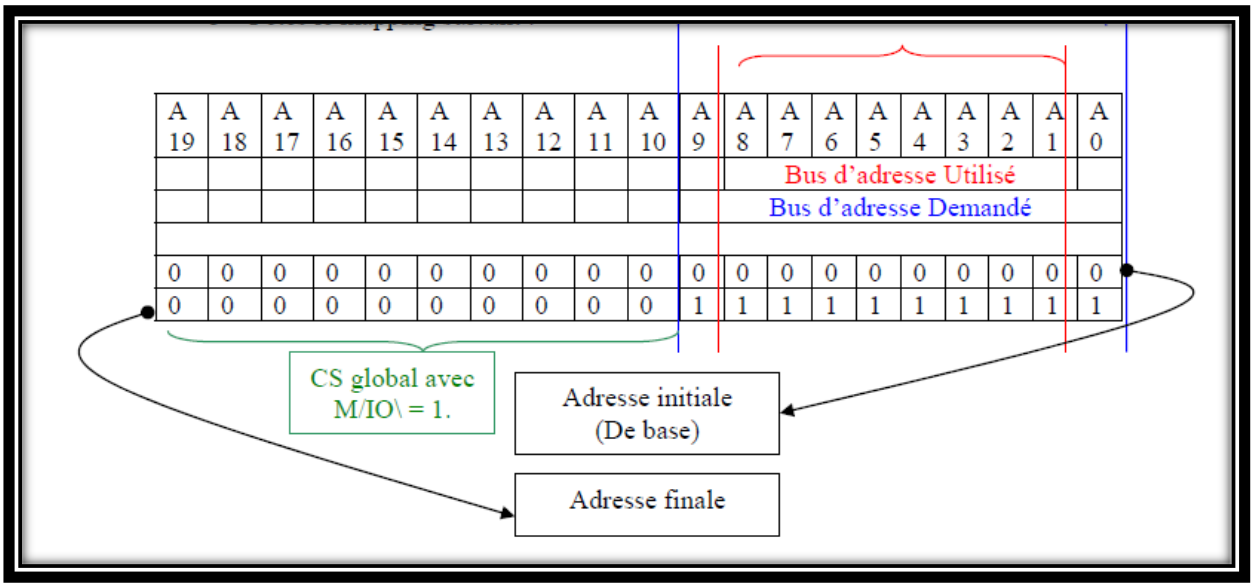


Figure (III.11) : Gestion des buses d'adresse

Comment calculer l'adresse finale : tous simplement par la relation suivante :

Adresse finale = adresse initiale + (valeur max sur le nombre de lignes d'adresse de bloc demandé).

Dans notre exemple :

Adresse finale = 00000h + (valeur max sur 10 lignes = 3FFh).

Adresse finale = 003FFh.

On remarque que les lignes d'adresse A10 … A19 et M/IO\ = 1 prend la même valeur pour n'importe quelle adresse de l'espace demandé de 1 Ko dans l'adresse de base spécifiée c'est le CS (CHIP SELECT) global de cette zone. Et en plus la ligne d'adresse A0 est non utilisée dans les lignes d'adresse utilisée par ce que cette dernière est utilisée avec la ligne BHE\ pour la

séparation paire et impaire du bloc demandé de 1 Ko. Il reste la ligne A9, cette ligne on utilise pour la séparation de l'espace mémoire demandée LOW (A9 = 0), et de l'espace mémoire demandée HIGH (A9 = 1).

On distingue dans ce cas que l'espace demandé est divisé en quatre cadrans sont :

- Cadran LOW paire.
- Cadran LOW impaire.
- Cadran HIGH paire.
- Cadran HIGH impaire.

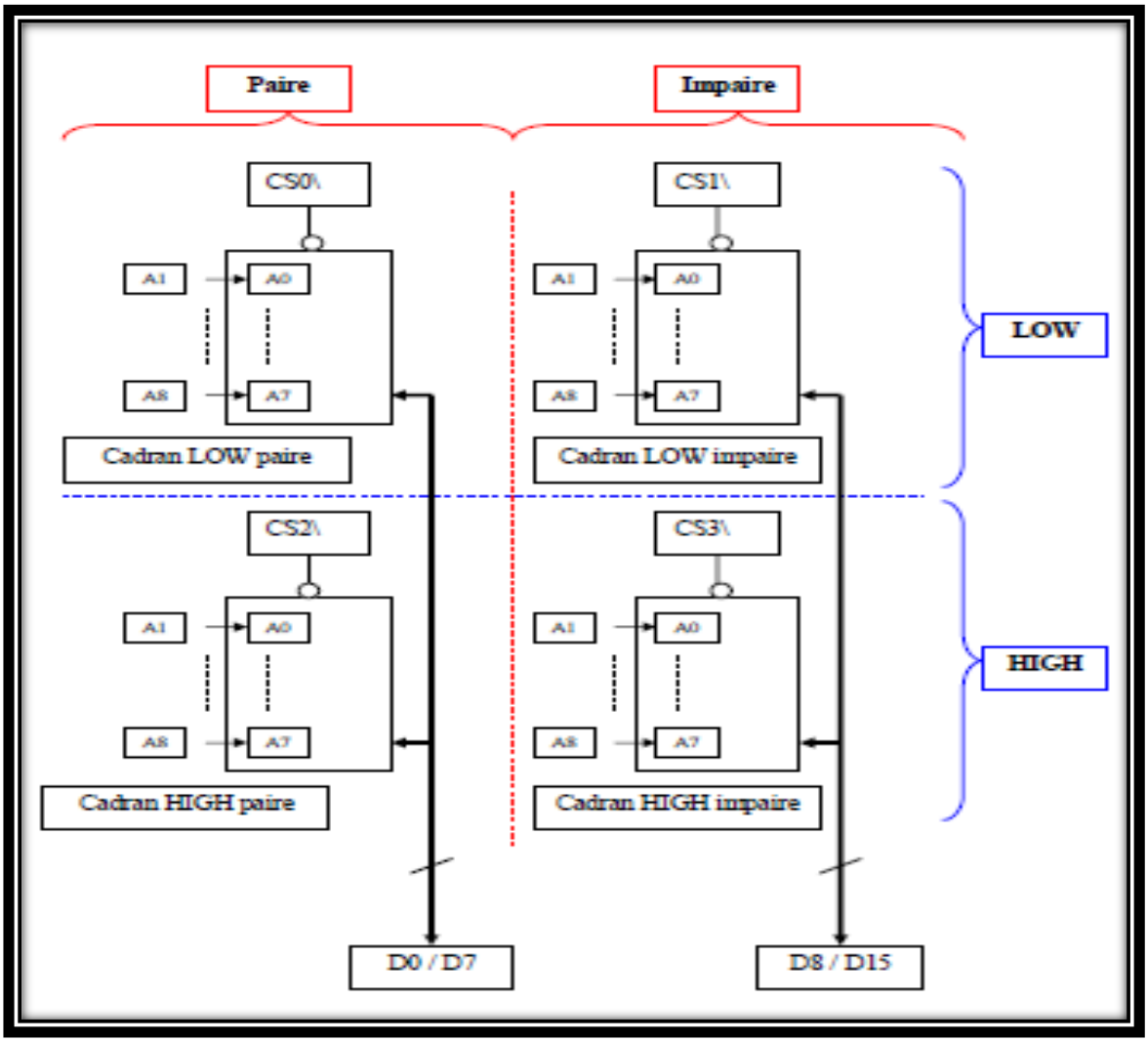


Figure (III.12) : Réalisation bloc mémoire à partir 4 mémoires

II.6.2 Décodage d'adresses

Les trois bits A13, A14 et A15 utilisés précédemment fournissent en fait 8 combinaisons, de 000 `a 111, d'ou la possibilité de connecter jusqu''a 8 boitiers mémoire de 8 Ko sur le bus. La mémoire totale implantée devient donc de 8 × 8 Ko = 64 Ko : valeur maximale possible avec

16 bits d'adresses. Pour cela, il faut utiliser un **circuit de décodage d'adresses**, dans ce cas : un **décodeur 3 vers 8**.

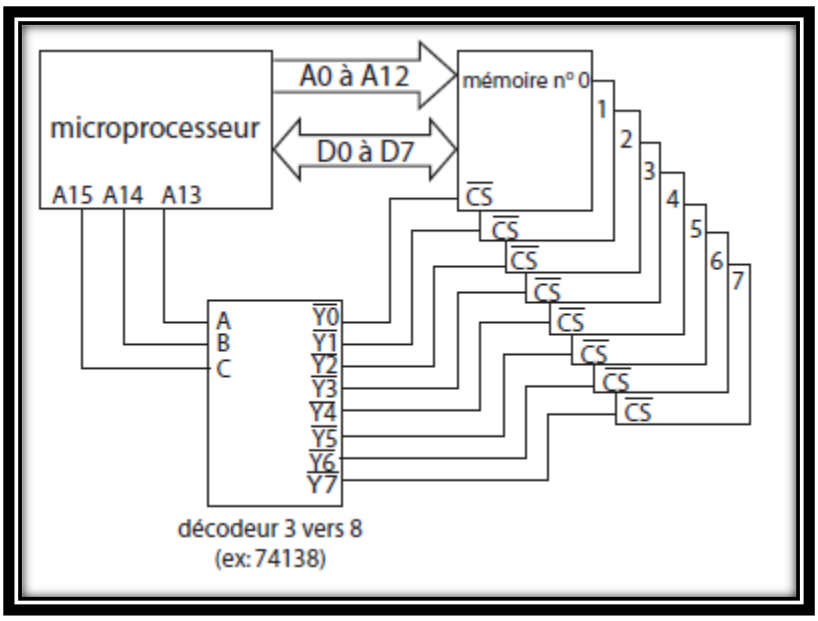


Figure (III.14) : Logigramme de sélection des boitiers des mémoires

Par l'intermédiaire d'un décodeur

CHAPITRE IV

IV.1 Définition :

Afin de pouvoir communiquer avec l'environnement extérieur (écran, souris, clavier, etc.), le microprocesseur utilise les interfaces d'entrées/sorties. Cette connexion se fait à travers les bus de données, d'adresses et de commande (voir Figure IV.1) Dans ce chapitre, nous allons présenter quelques interfaces d'E/S à savoir l'interface parallèle 8255, l'interface série 8250, Les accès à ces périphériques sont appelés les ports (exemple :port série, port parallèle, etc.)

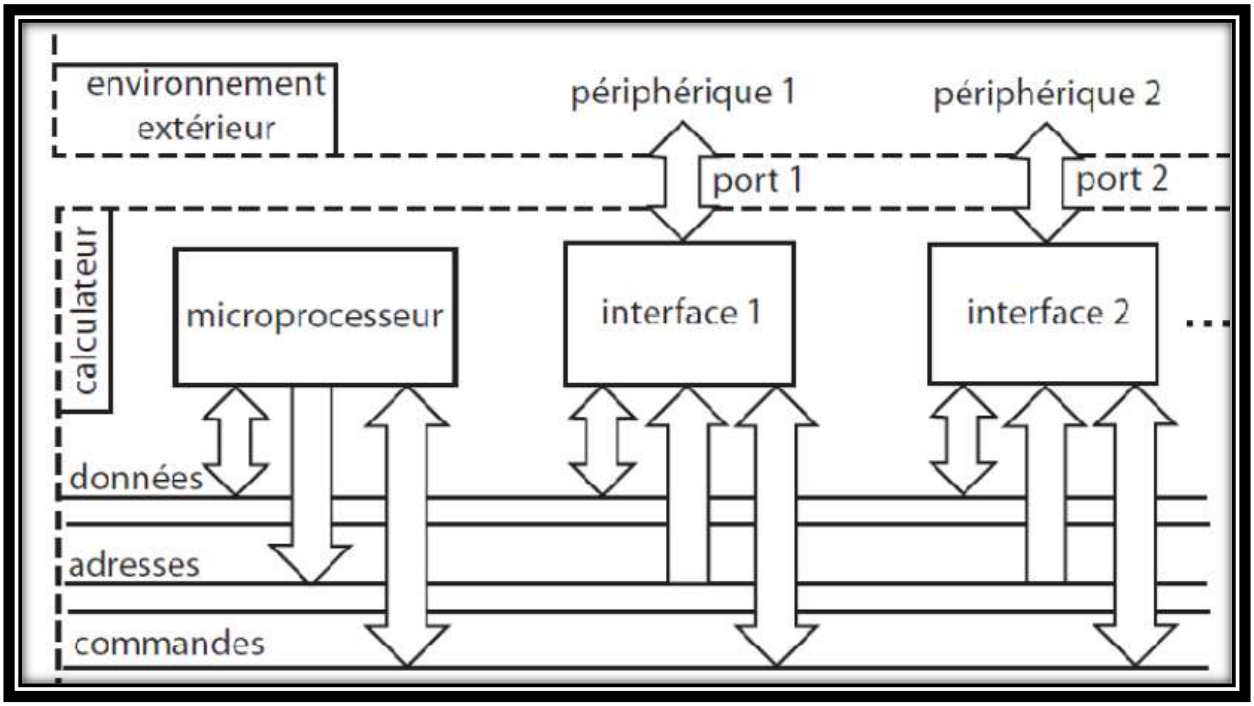


Figure (IV.1) : interfaçage µp8086

Un circuit d'E/S est schématisé par la Figure IV.2

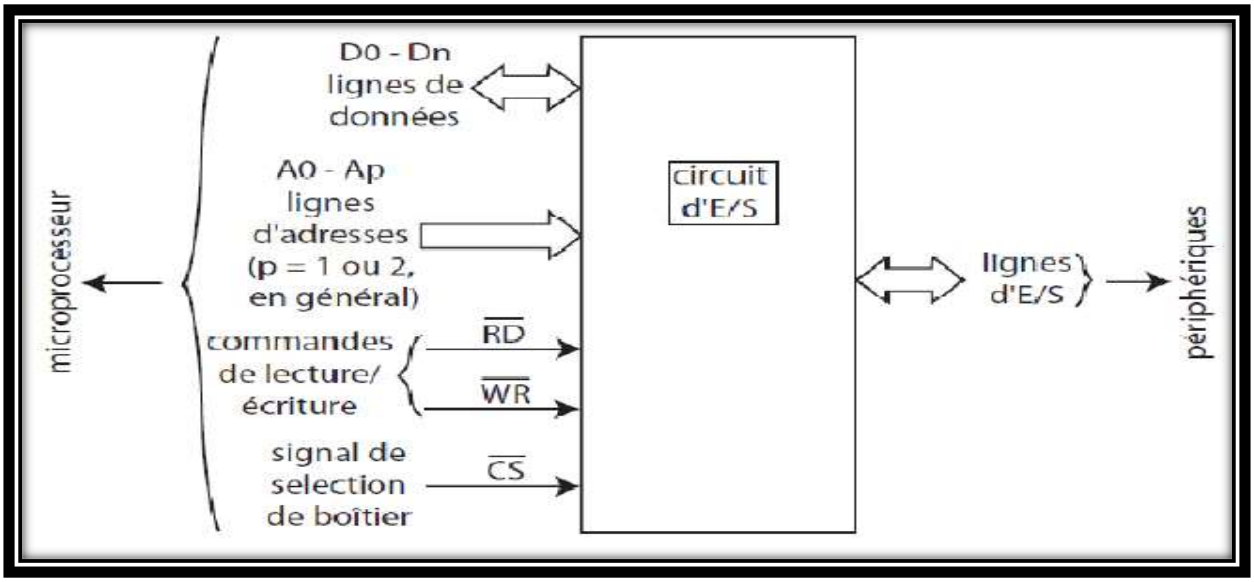


Figure (IV.2) : Schéma synoptique d'un circuit d'E/S

IV.2 Généralité sur interfaçage :

IV.2.1Adressage des ports d'E/S

Afin de gérer les échanges avec les périphériques, un circuit d'E/S possède des registres : les registres de données et les registres de configuration.

Une adresse est assignée à chacun de ces registres et le microprocesseur accède à un port en spécifiant l'un de ces registres.

Les adresses des ports d'E/S peuvent être vues par les microprocesseurs de deux manières :

- Les E/S sont mappées en mémoire : les adresses des ports d'E/S appartiennent au même espace mémoire que les circuits mémoire ; on parle alors d'adressage cartographique. Dans ce cas l'espace d'adressage des mémoires diminue, et l'adressage des ports se fait avec le même nombre de bits que pour les mémoires. Donc toutes les instructions possibles sur les mémoires peuvent être utilisées par les E/S.
- Le microprocesseur considère deux espaces distincts : l'espace d'adressage des mémoires et l'espace d'adressage des ports d'E/S. C'est le cas du microprocesseur 8086. On parle adressage indépendant. Dans ce cas l'espace mémoire adressable ne diminue pas et l'adressage des E/S peut se faire avec un nombre de bits moins important. Les instructions utilisées par les E/S sont quant à elles spécifiques.

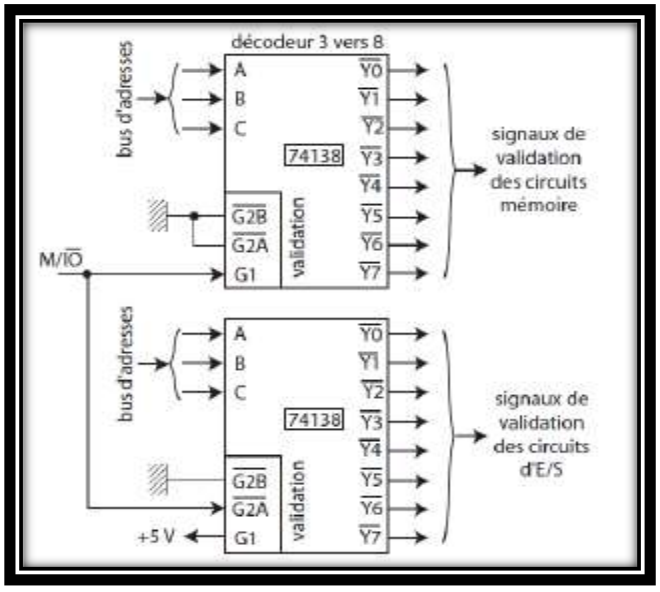


Figure (IV.3) : adressage des E/S par le 8086

Une même adresse peut désigner une case mémoire ou un port d'E/S, le microprocesseur doit donc fournir un signale qui permet de faire la différence entre les deux.

Le 8086 dispose d'un espace mémoire de 1 Mo (adresse d'une case mémoire sur 20 bits) et d'un espace d'E/S de 64 Ko (adresse d'un port d'E/S sur 16 bits). Le signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S est la ligne M/not(IO):

M/not(IO) = 1 alors accès à la mémoire et M/IO =0 pour un accès aux E/S. Ce signal est utilisé pour valider le décodage d'adresse dans les deux espaces : Les instructions de lecture et d'écriture d'un port d'E/S sont respectivement les instructions IN et OUT. Elles placent la ligne M/IO à 0 alors que l'instruction MOV place celle-ci à 1.

Lecture d'un port d'E/S		
IN AL,adresse	lecture d'un port sur 8 bits	si l'adresse du port d'E/S est sur un octet
IN AX,adresse	lecture d'un port sur 16 bits	
IN AL,DX	lecture d'un port sur 8 bits	si l'adresse du port d'E/S est sur deux octets (DX contient l'adresse du port d'E/S
IN AX,DX	lecture d'un port sur 16 bits	
Ecriture d'un port d'E/S		
OUT adresse,AL	écriture d'un port sur 8 bits	si l'adresse du port d'E/S est sur un octet
OUT adresse,AX	écriture d'un port sur 16 bits	
OUT DX,AL	écriture d'un port sur 8 bits	si l'adresse du port d'E/S est sur deux octets (DX contient l'adresse du port d'E/S)
OUT DX,AX	écriture d'un port sur 16 bits	

Tableau (IV.1) : lecture et écriture d'un port d'E/S

Exemples :

1/ lecture d'un port d'E/S sur 8 bits `a l'adresse 300H mov dx,300H ; le registre dx contient la valeur 300 **in al,dx** ; **lecture du contenu du registre dx**

2/ écriture de la valeur 1234H dans le port d'E/S sur 16 bits à l'adresse 49H **mov ax,1234H** ; **le registre ax contient la valeur 1234h**

out 49H,ax ; écriture dans le port d'E/S à l'adresse 49h

IV.3 Les Interfaces parallèles :

Le rôle d'une interface parallèle est de transférer des données du microprocesseur vers des périphériques et inversement, tous les bits de données étant envoyés ou reçus simultanément.

L'interface parallèle est une carte électronique I/O qui fait l'échange de donnée entre le µp et un périphérique en parallèle selon la taille de bus de données (8 ou 16 bits). Les points d'accès aux I/O sont appelés PORTS.

Gestion des ports :

Le µp 8086 fournées un espace adressable pour l'accès ports de taille 64Ko :

16 lignes d'adresse et 16 lignes de données plus quelque lignes de contrôles. Mais on remarque que la mémoire et les ports I/O partage le même bus de données et d'adresse, alors comment différencier l'accès entre eux. Tous simplement si on rappel le cours de l'interfaçage mémoires dans la partie « CHIP SELECT » de décodeur 74138 l'entrée « G », cette entrée est attaquée par le signal « M/IO\ » de bus de contrôle.

- Pour l'accès mémoire : M/IO\ = 1.
- Pour l'accès ports : M/IO\ = 0.

Les instructions utilisées pour R/W d'un port sont « IN/OUT ». Il existe deux types d'accès, accès direct et un autre indirect. Les registres utilisés sont l'accumulateur « AL/AX » et le registre de données « DX ».

On peut citer dans ce stade les 2 interfaces parallèles standard « PORT PARALLELE » et « COUPLEUR PARALLELE PPI 8255».

IV.3.1 Le port parallèle :

Le 8255 est une interface parallèle programmable : elle peut être configurée en entrée et/ou en sortie par programme.

Cette interface est généralement utilisée par les imprimantes, car elles utilisent beaucoup de donnés. Cependant cette interface à une limite physique lié aux problèmes de diaphonie entre les fils, ce qui à pour principale conséquence de limiter la longueur des câbles parallèles. Au-delà de 10 m on commence à avoir une atténuation significative du signal. Ceci dit il existe des amplificateurs de signal qui prennent généralement la forme de boîtier de partage auto commuté.

Description physique : Le port parallèle des PC se présente sous la forme d'une prise DB25 femelle.

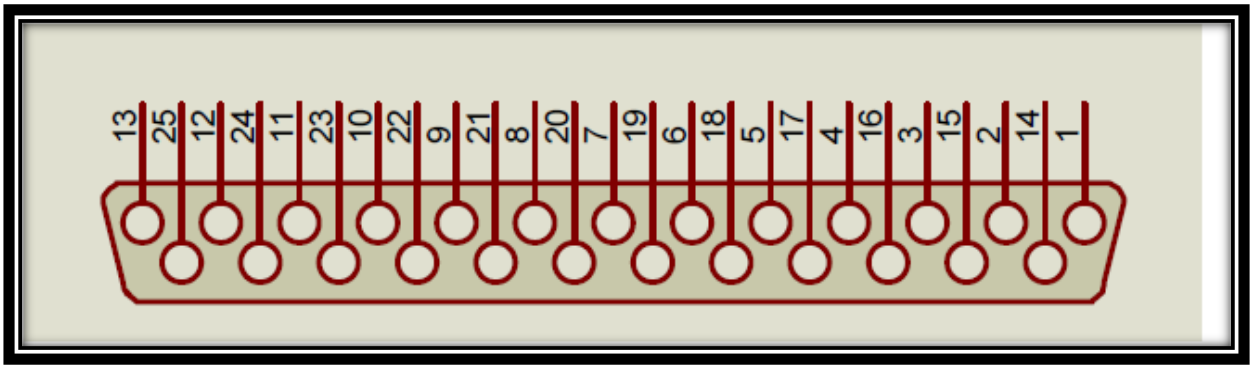


Figure (IV.4) : schéma d'une prise DB25

Description du brochage du connecteur DB25 F :

Broche	Description	E/S
1	-STROBE	Sortie
2	+Bit de données 0	Sortie
3	+Bit de données 1	Sortie
4	+Bit de données 2	Sortie
5	+Bit de données 3	Sortie
6	+Bit de données 4	Sortie
7	+Bit de données 5	Sortie
8	+Bit de données 6	Sortie
9	+Bit de données 7	Sortie
10	-Accusé de réception	Entrée
11	+Occupé	Entrée
12	+Plus de papier	Entrée
13	+Sélection	Entrée
14	-Chargement automatique	Sortie
15	-Erreur	Entrée
16	-Initialisation de l'imprimante	Sortie
17	-Sélection de l'entrée	Sortie
18	-Retour du bit de données 0 (masse)	Entrée
19	-Retour du bit de données1 (masse)	Entrée
20	-Retour du bit de données 2 (masse)	Entrée
21	-Retour du bit de données 3 (masse)	Entrée
22	-Retour du bit de données 4 (masse)	Entrée
23	-Retour du bit de données 5 (masse)	Entrée
24	-Retour du bit de données 5 (masse)	Entrée
25	-Retour du bit de données 7 (masse)	Entrée

1) Description des ports :

Il dispose 3 ports pour la communication avec l'imprimante, ces ports sont le port DATA pour la transmission des caractères, le port STATUS pour indiquer l'état de l'imprimante, le port CONTROL pour contrôler l'échange de données.

Port DATA : est un port de données de taille 8 bits configuré en sortie dans la version standard.

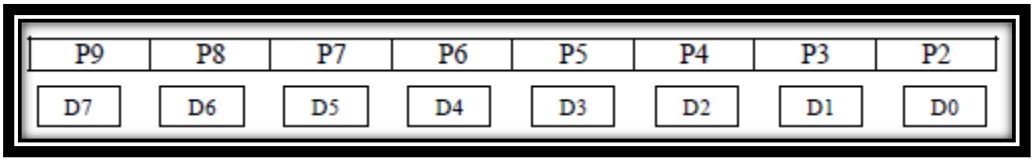


Figure (IV.5) : Port DATA

Port STATUS : est un port de 5 bits configuré en entrée.

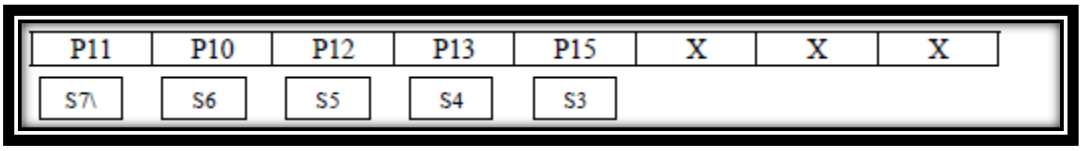


Figure (IV.6) : Port STATUS

Port CONTROL : est un port de 4 bits configuré en sortie.

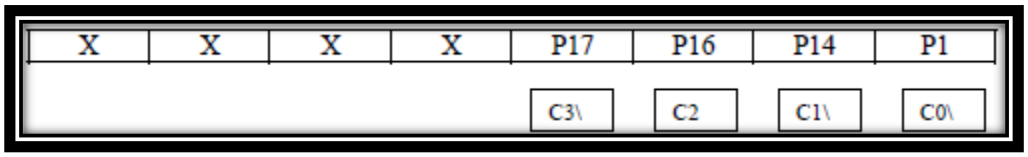


Figure (IV.7) : Port CONTROL

Les adresses des ports parallèles implantés dans les PC compatible PC IBM :

Numéro de LPT	Port DATA	Port STATUS	Port CONTROL
LPT1	0378H	0379H	037AH
LPT2	0278H	0279H	027AH
LPT3	03BCH	03BDH	03BEH

Exemple d'accès :

Lecture du port STATUS :

On remarque que l'adresse du port est une adresse sur 16 bits le mode d'accès indirect par l'utilisation de registre DX.

```
MOV DX, 0379h
IN AL, DX
```

Ecriture dans le port DATA ou CONTROL:

```
MOV AL, 55h
MOV DX, 0378h ; dans le cas de registre de control on mettre 037Ah
OUT DX, AL
```

IV.3.2 Le coupleur parallèle PPI 8255:

Le rôle d'une interface parallèle est de transférer des données du µp vers un périphérique et l'inverse en parallèle. Le 8255 est une interface parallèle programmable, elle peut être configurée en entrée et/ou en sortie par programme.

Brochage du 8255 :

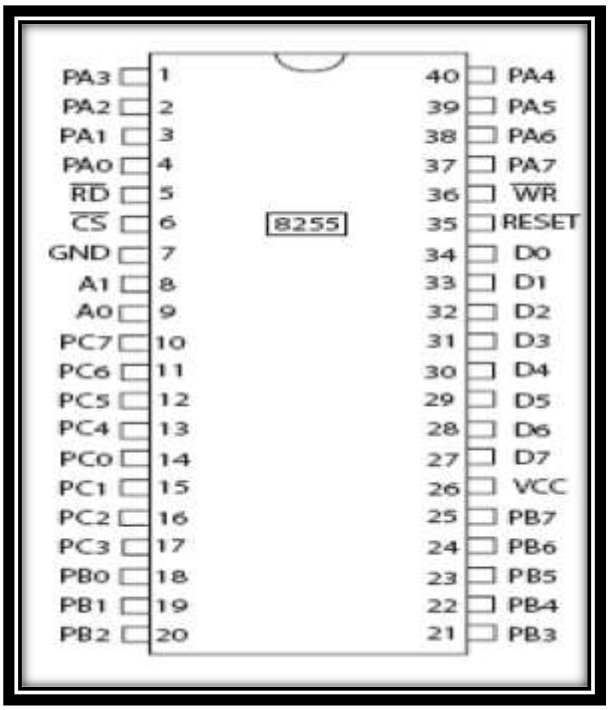


Figure (IV.8) : Brochage de 8255

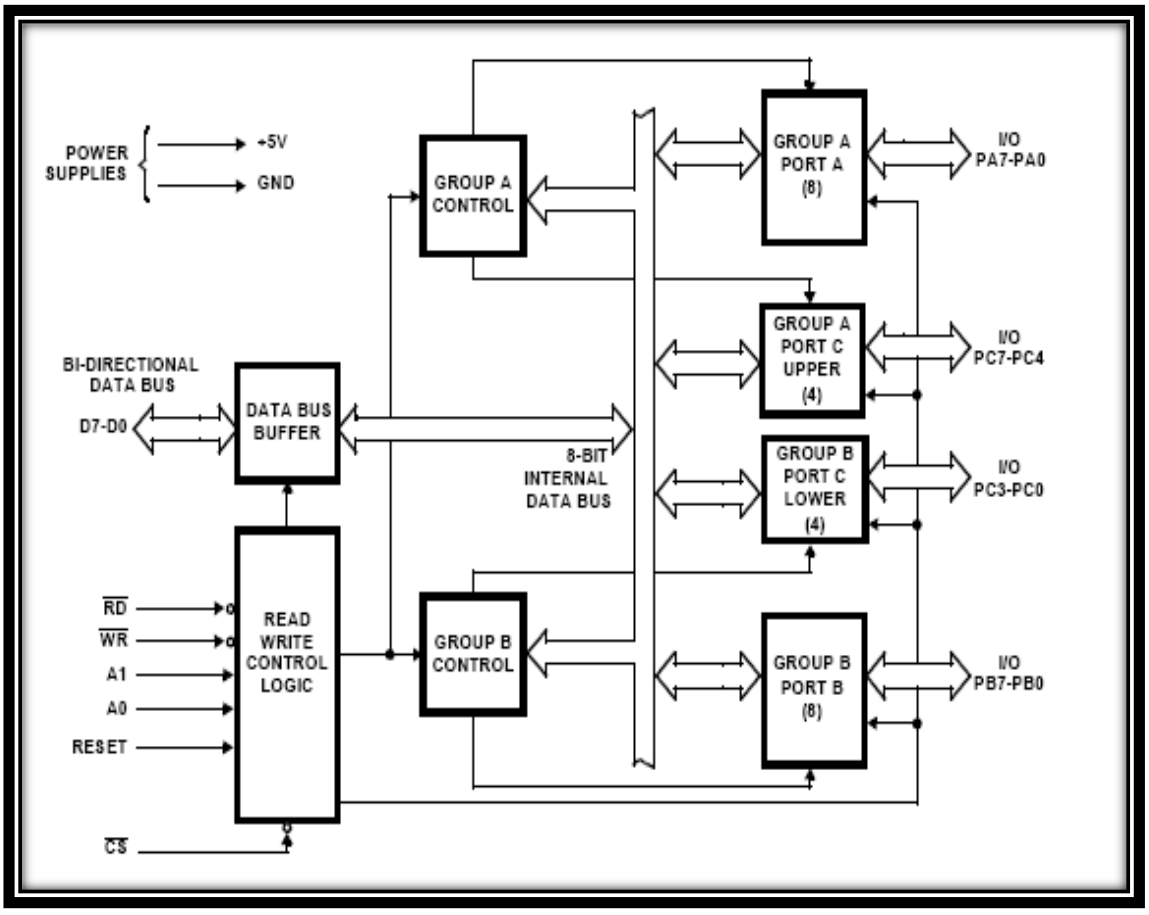


Figure (IV.9) : Brochage fonctionnelle de 8255

1) Description des PINs et l'accès registres :

SYMBOL	PIN	TYPE	Description
VCC	26	-	Pin d'alimentation 5v
GND	7	-	Masse
D0-D7	27-34	I/O	Bus de données 8 bits bidirectionnels
RESET	35	I	Entrée de demande d'initialisation boîtier
CS\	6	I	Entrée d'activation boîtier
RD\	5	I	Entrée de demande de lecture registre
WR\	36	I	Entrée de demande d'écriture registre
A0-A1	8,9	I	Lignes d'adresse des registres
PA0-PA7	1-4,37-40	I/O	Pins du port A
PB0-PB7	18-25	I/O	Pins du port B
PC0-PC7	10-17	I/O	Pins du port C

2) Accès registres :

A1	A0	RD\	WR\	CS\	Opération
0	0	0	1	0	PORTA : DATA BUS
0	1	0	1	0	PORTB : DATA BUS
1	0	0	1	0	PORTC : DATA BUS
1	1	0	1	0	CR :DATA BUS
0	0	1	0	0	DATA BUS :PORTA
0	1	1	0	0	DATA BUS :PORTB
1	0	1	0	0	DATA BUS : PORTC
1	1	1	0	0	DATA BUS :CR
X	X	X	X	1	DATA BUS Haute impédance
X	X	1	1	0	DATA BUS Haute impédance

3) Structure de registre CR :

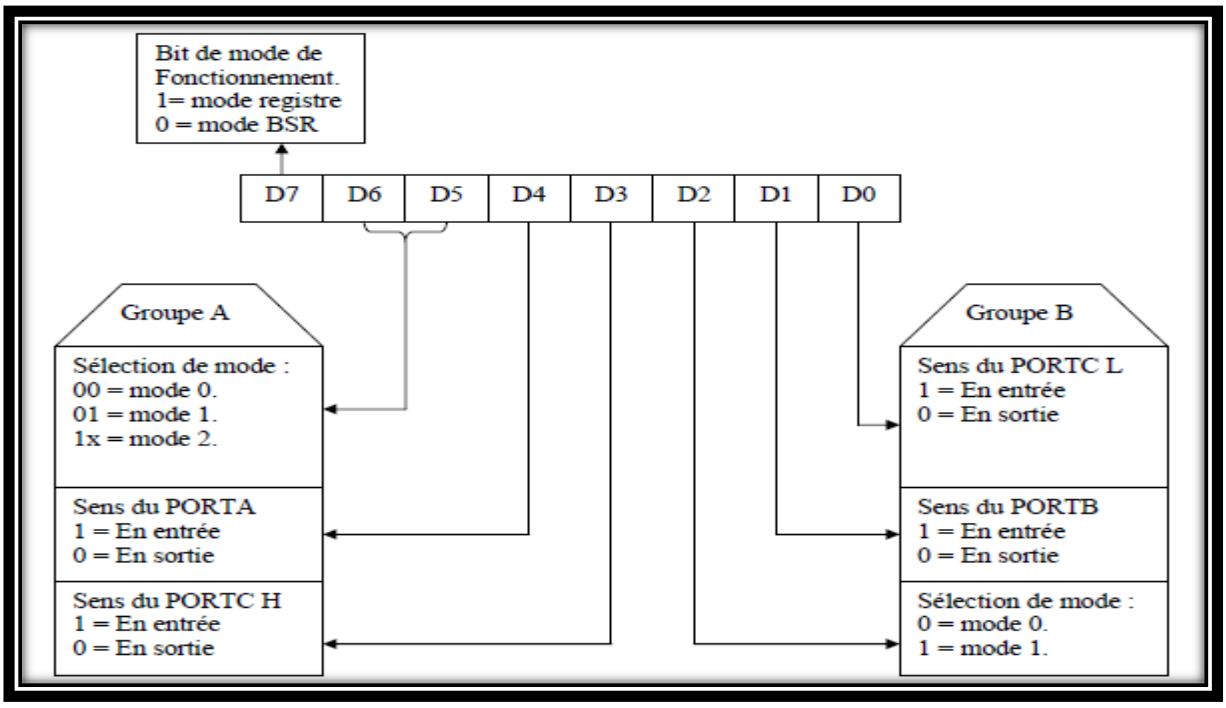


Figure (IV.10) : Structure de registre CR

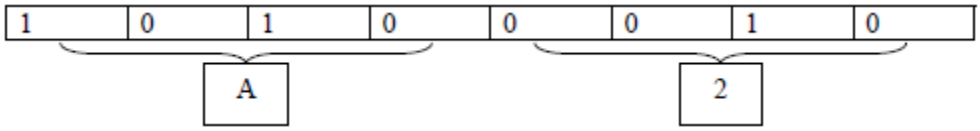
Exemple :
Faites la configuration suivante :

☞ PORTA en mode 1 en sortie.

☞ PORTB en mode 0 en entrée.

☞ PORTC en sortie.

Solution :



MOV AL, 0A2h : la valeur de CR qui assure la configuration.

MOV DX, ADR_CR :l'adresse de registre CR dans le registre DX.

OUT DX, AL :écrire dans le registre CR.

c)Fonctionnement du PPI 8255 :

Le PPI 8255 fonctionne en 4 modes (mode0, mode1, mode2, mode BSR). Ces modes sont activés par la programmation de registre CR.

➢ **Mode BSR (BIT SET RESET):**

Ce mode utilise par le PORTC uniquement qui peuvent être mise à « 1 » ou « 0 » d'une ligne du PORTC individuellement. Dans ce mode la structure de registre CR est modifié mais cette modification ne dérange pas le mode de fonctionnement des ports (A et B).

Remarque : n'oublie pas de configurer le PORTC en sortie dans le registre CR.

Structure de registre CR en mode BSR :

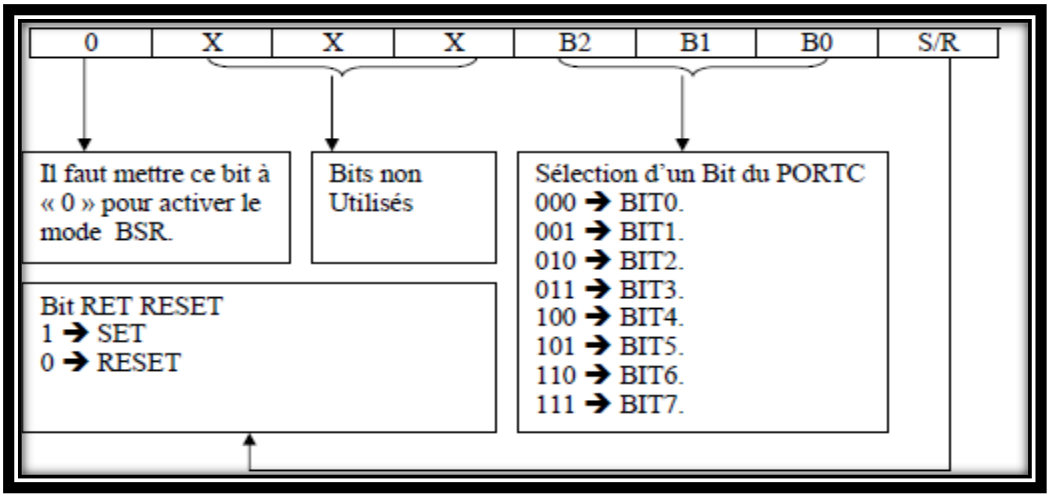


Figure (IV.11) : Structure de registre CR en mode BSR

Exemple :
Mettez la PIN6 du PORTC à « 1 » et la PIN3 à « 0 ».

Configuration de PORTC en sortie CR:B'10000000' = 80h.

Pour mettre la PIN6 à « 1 » le CR :B'00001101' = 0Dh.

Pour mettre la PIN6 à « 1 » le CR :B'00000110' = 06h. ; Configuration des registres et PORTC en sortie.

```
MOV DX, ADR_CR
MOV AL, 80h
OUT DX, AL : PIN6 : 1
MOV AL, 0Dh
OUT DX, AL ; PIN3 : 0
MOV AL, 06h
OUT DX, AL
```

☞ **Mode 0 :**
C'est le mode le plus simple : les ports sont configuré en entrées/sorties de base. Les données écrites dans les registres correspondants sont mémorisées sur les lignes de sorties ; l'état des entrées est recopié dans les registres correspondants et n'est pas mémorisé (bufférisées).

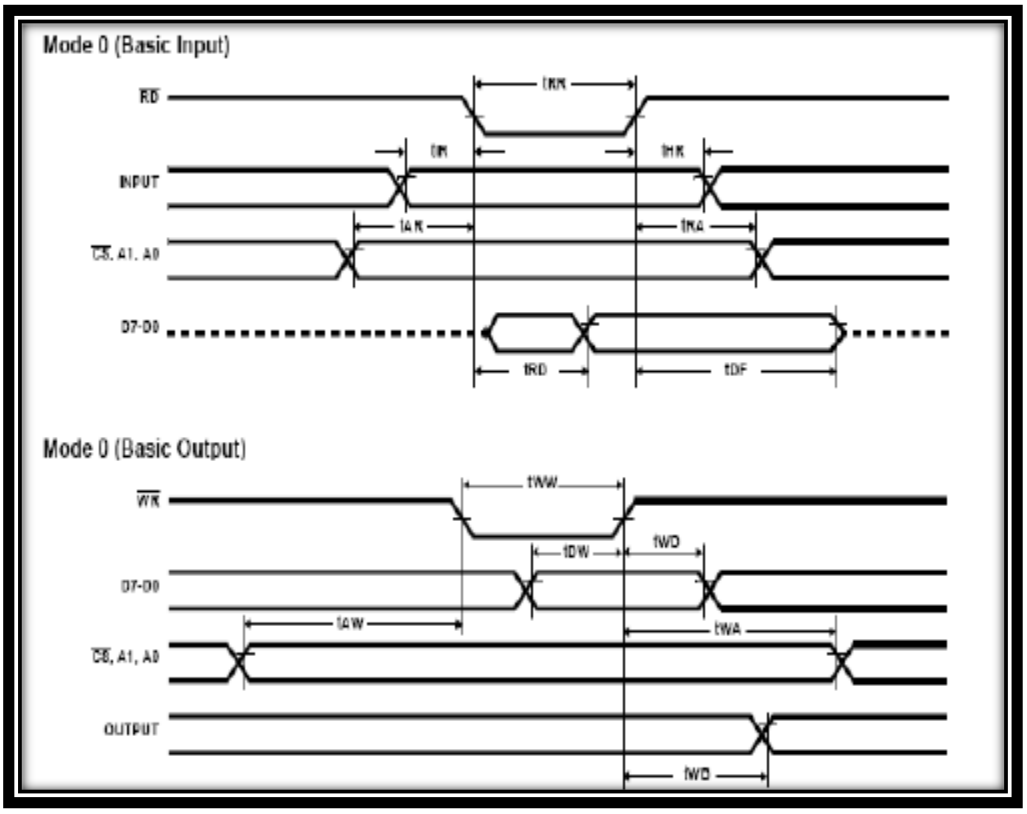


Figure (IV.12) : Chronogramme d'accès Input Output

Différentes mots de configuration en mode 0 :

Valeur CR	Description
80h	A: OUT; B: OUT; C_L: OUT; C_H: OUT.
82h	A: OUT; B: IN; C_L: OUT; C_H: OUT.
81h	A: OUT; B: OUT; C_L: IN; C_H: OUT.
83h	A: OUT; B: OUT; C_L: IN; C_H: IN.
88h	A: OUT; B: IN; C_L: OUT; C_H: OUT.
90h	A: IN; B: OUT; C_L: OUT; C_H: OUT.
89h	A: OUT; B: IN; C_L: IN; C_H: OUT.
91h	A: IN; B: OUT; C_L: IN; C_H: OUT.
8Ah	A: OUT; B: IN; C_L: OUT; C_H: IN.
92h	A: IN; B: OUT; C_L: OUT; C_H: IN.
8Bh	A: OUT; B: IN; C_L: IN; C_H: IN.
98h	A: IN; B: IN; C_L: OUT; C_H: OUT.
99h	A: IN; B: IN; C_L: IN; C_H: OUT.
93h	A: IN; B: OUT; C_L: IN; C_H: IN.
9Ah	A: IN; B: IN; C_L: OUT; C_H: OUT.
9Bh	A: IN; B: IN; C_L: IN; C_H: IN.

☞ **Mode 1 :**

Il est utilisé pour le dialogue avec des périphériques nécessitant un asservissement (contrôle), dans ce mode les registres de 8255 sont regroupés en deux groupes (GROUPE A et GROUPE B) chaque groupe dispose le port du groupe plus quelque lignes du PORTC comme signaux du contrôle.

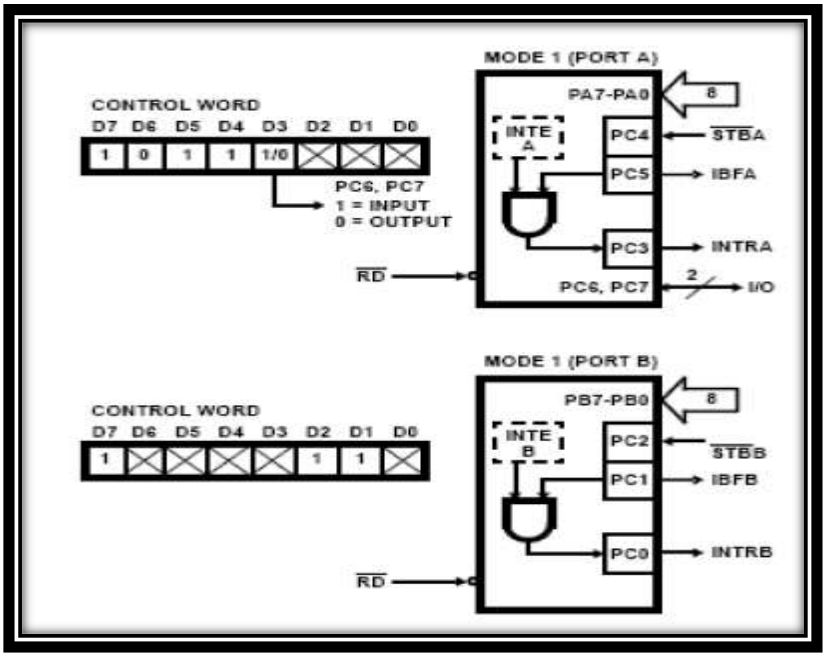


Figure (IV.13) : Mode 1 en entrée

On remarque que le PORTC devient seulement les PINs PC6 et PC7 le reste des PINs serons des PINs du contrôle de groupe A et B.

Description des signaux du contrôle :

STB (STROBE) : ce signal active au niveau bas délivrer par le périphérique qui indique que la donnée est présente dans le BUFFER d'entrée (PORTA ou PORTB).

IBF (INPUT BUFFER FULL) : ce signal active au niveau haut pour indique au périphérique la bonne réception de la donnée.

INTR (INTERRUPT REQUEST) : ligne active au niveau haut lorsque STB\ et INB sont aux niveau haut.

INTE (INTERUPT ENABLE) : c'est un bit utilisé pour autoriser le mode defonctionnement sous interruption.

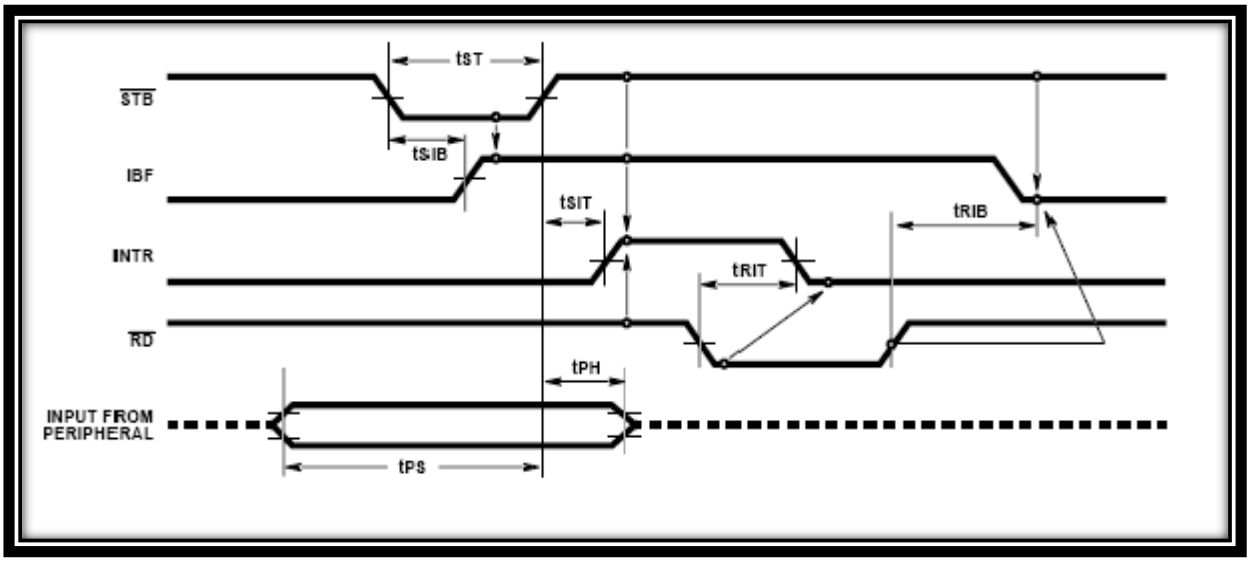


Figure (IV.14) : Chronogramme d'accès mode 1 en entrée

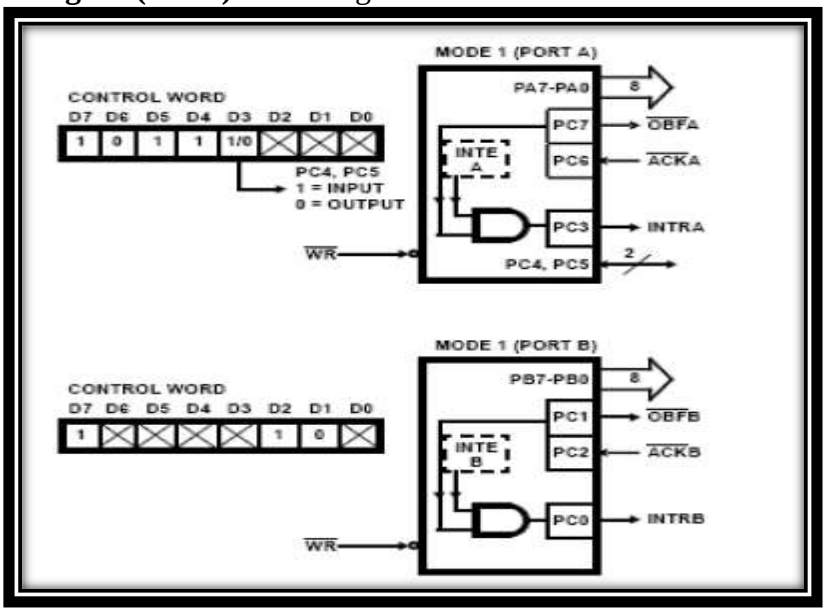


Figure (IV.15) : Mode 1 en sortie

Le PORTC est appelé le mot d'état de groupe A et B :

I/O	I/O	IBF A	INTE A	INTR A	INTE B	IBF B	INTR B

N'oublie pas que les ports (A et B) sont des mémoires en entrée dans ce mode.

Dans ce cas-là le PORTC devient les PINs PC5 et PC4:

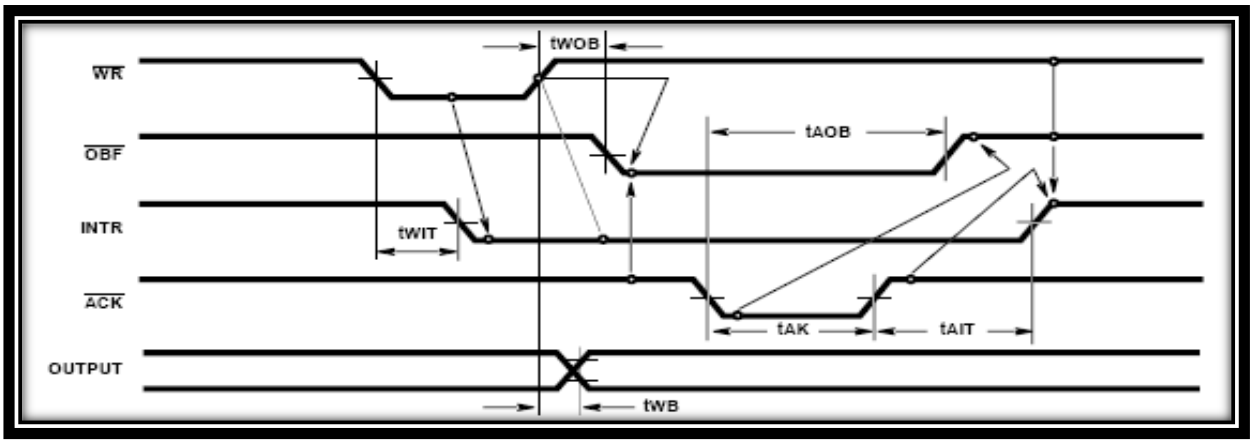


Figure (IV.16) : Chronogramme d'accès mode 1 en sortie (Description des signaux du contrôle)

OBF (OUTPUT BUFFER FULL) : ce signal passe au niveau bas si le µp écrire une donnée dans le BUFFER de sortie, ce signal aussi indique au périphérique que il y a une donnée dans le BUFFER (les lignes du port A ou B).

ACK (ACKNOWLEDGE INPUT) : ce signal délivré par le périphérique pour indiquer la bonne réception de la donnée sous forme impulsion négative.

INTR et **INTE** sont les mêmes signaux du mode 1 en entrée.

Le mot d'état du mode 1 en sortie:

OBF\ A	INTE A	I/O	I/O	INTR A	INTE B	OBF\ B	INTR B

☞ **Mode 2 :**

Dans ce mode est utilisé seulement par le PORTA et peut être considérer comme un bus bidirectionnelle.

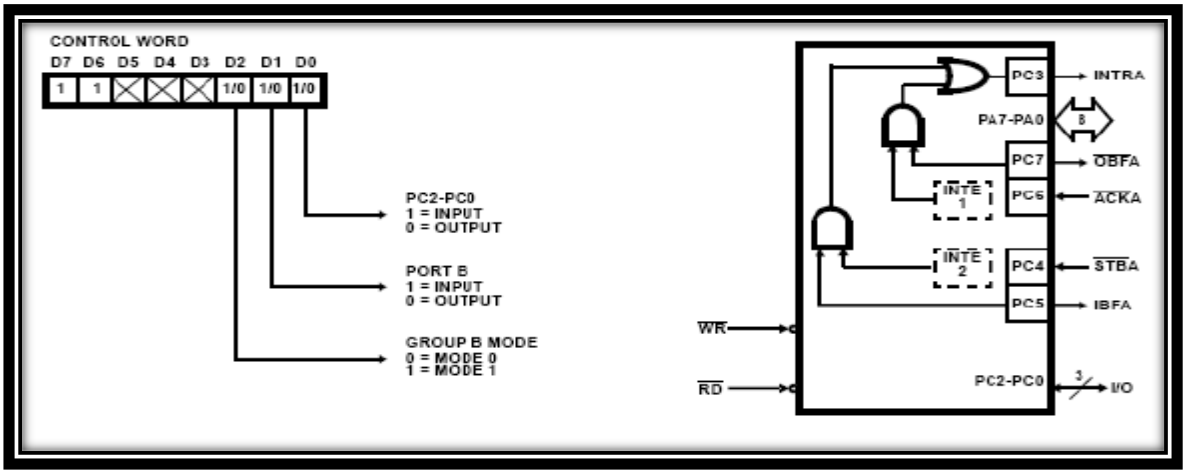


Figure (IV.17) : Mode 2 en sortie

Le groupe A sera donc le PORTA plus 5 lignes du PORTC, et le groupe B si on configure en mode 1 le PORTC est disparu par ce que le mode 1 extraire 3 lignes de ce dernier, mais dans le cas mode 0 le PORTC est de 3 lignes.

Les signaux du contrôle sont les mêmes que les signaux du mode 1 en I/O.

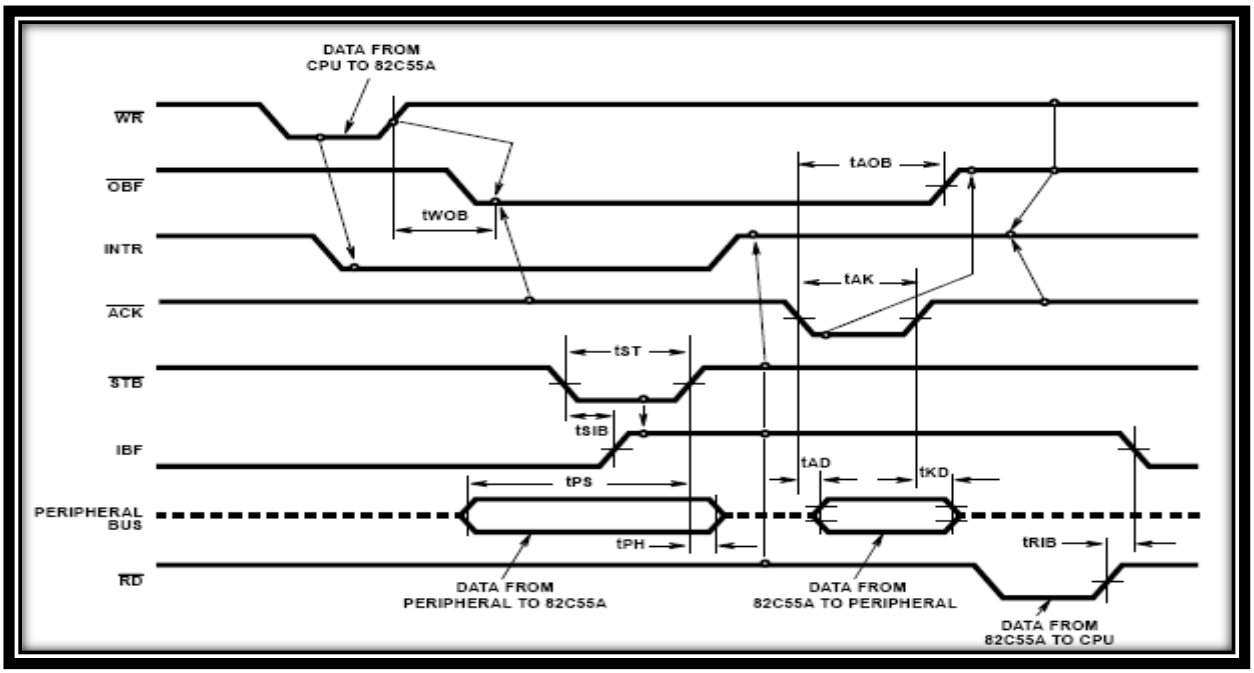


Figure (IV.18) : Chronogramme d'accès mode en entré sortie

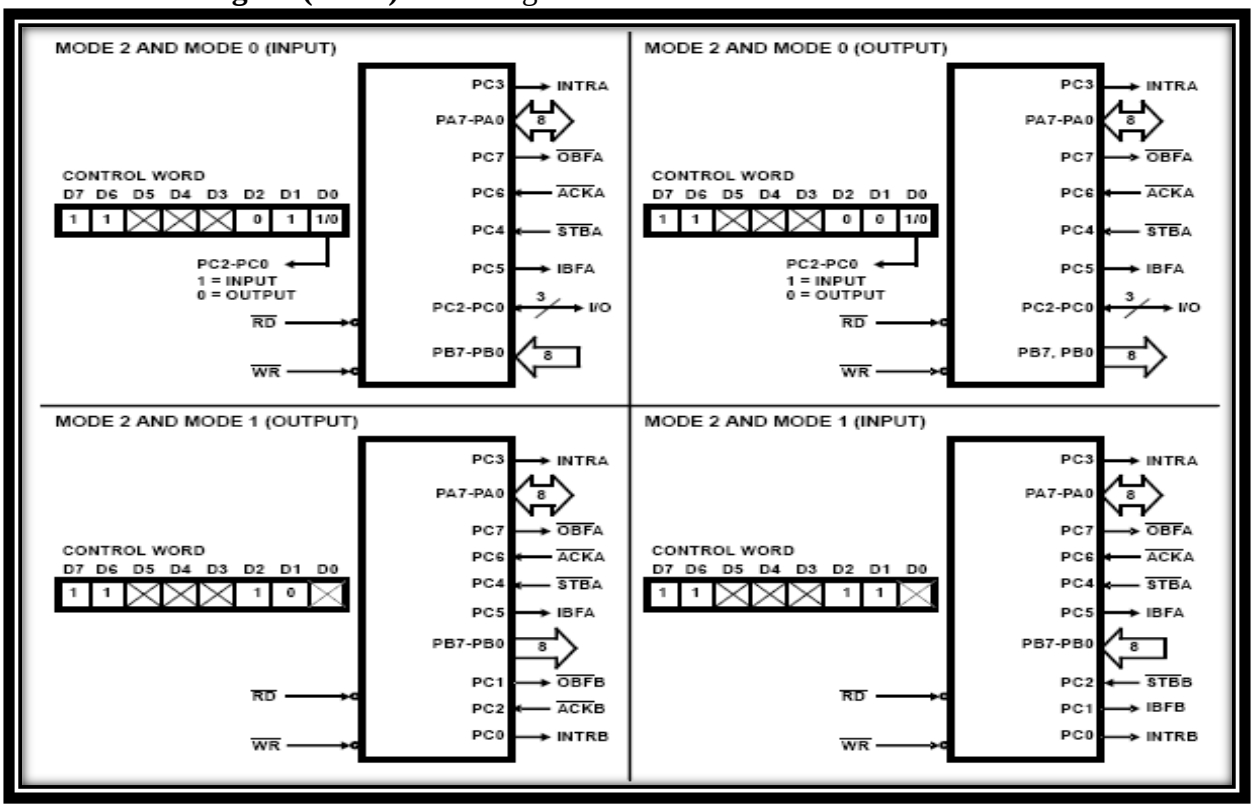


Figure (IV.19) : Différentes combinaisons en mode 2

Les mots d'état pour chaque combinaison :

MODE 2 AND MODE 0 (INPUT) :

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	I/O	I/O	I/O

MODE 2 AND MODE 0 (OUTPUT) :

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	I/O	I/O	I/O

MODE 2 AND MODE 1 (INPUT) :

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	INTE B	IBF B	INTR B

MODE 2 AND MODE 1 (OUTPUT) :

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	INTE B	OBF\ B	INTR B

IV.4 L'interface série

Une interface série permet d'échanger des données entre le microprocesseur et un périphérique bit par bit. Ce qui permet de diminuer le nombre de connexions (1 fil pour l'émission, 1 fil pour la réception). Mais la vitesse de transmission devient plus faible que celle d'une interface parallèle.

Deux types de transmissions séries existent :

- asynchrone : chaque octet peut être émis ou reçu sans durée déterminée entre un octet et le suivant ;
- synchrone : les octets successifs sont transmis par blocs séparés par des octets de synchronisation.

La transmission asynchrone la plus utilisée est celle qui est définie par la norme RS232.

Exemple : transmission du caractère 'E' (code ASCII 45H = 01000101B) sous forme série selon la norme RS232 :

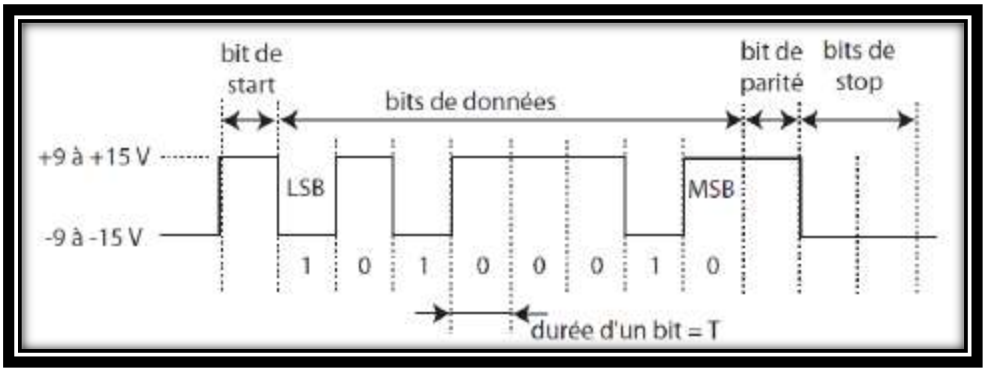


Figure (IV.20) : exemple de transmission série

- L'état 1 correspond à une tension négative comprise entre −9 et −15 V, l'état 0 à une tension positive comprise entre +9 et +15 V. Au repos, la ligne est à l'état 1 (tension négative) ;
- Le bit de start marque le début de la transmission du caractère ;
- Les bits de données sont transmis l'un après l'autre en commençant par le bit de poids faible. Ils peuvent être au nombre de 5, 6, 7 ou 8. Chaque bit est maintenu sur la ligne pendant une durée déterminée T. L'inverse de cette durée définit la fréquence de bit = nombre de bits par secondes = vitesse de transmission. Les vitesses normalisées sont : 50, 75, 110, 134.5, 150, 300, 600, 1200, 2400, 4800, 9600 bits/s ;
- Le bit de parité (facultatif) est un bit supplémentaire dont la valeur dépend du nombre de bits de données égaux à 1. Il est utilisé pour la détection d'erreurs de transmission ;
- Les bits de stop (1, 1.5 ou 2) marquent la fin de la transmission du caractère.

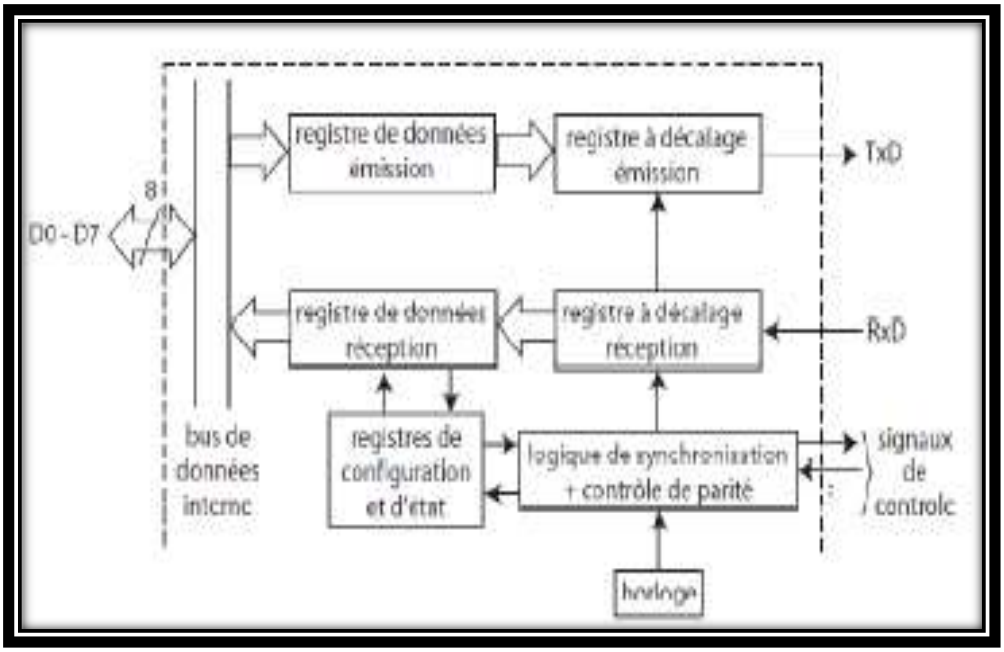


Figure (IV.21) : Principe d'une interface série

Un circuit intégré d'interface série asynchrone s'appelle un UART : Universal Asynchrones Receveur Transmitter) ; une interface série synchrone/asynchrone est un USART.

☞ **Rôle :**

L'UART est conçue pour gérer les signaux en provenance d'un modem. Aussi est-il nécessaire de donner quelles broches sont utilisées par l'ETCD (Equipement Terminal de Communication de Données).

VI.4.1 Circuit intégré 8250

L'interface 8250 possède plus de 8 registres de 8 bits permettant de gérer la communication.

Ces registres sont lus et modifiés par le processeur par le biais des instructions IN et OUT vues plus haut. Comme on ne dispose que de 3 bits d'adresse :
(A0, A1 et A2), plusieurs registres doivent se partager la même adresse La solution est d'utiliser un bit d'un registre spécial, DLAB.

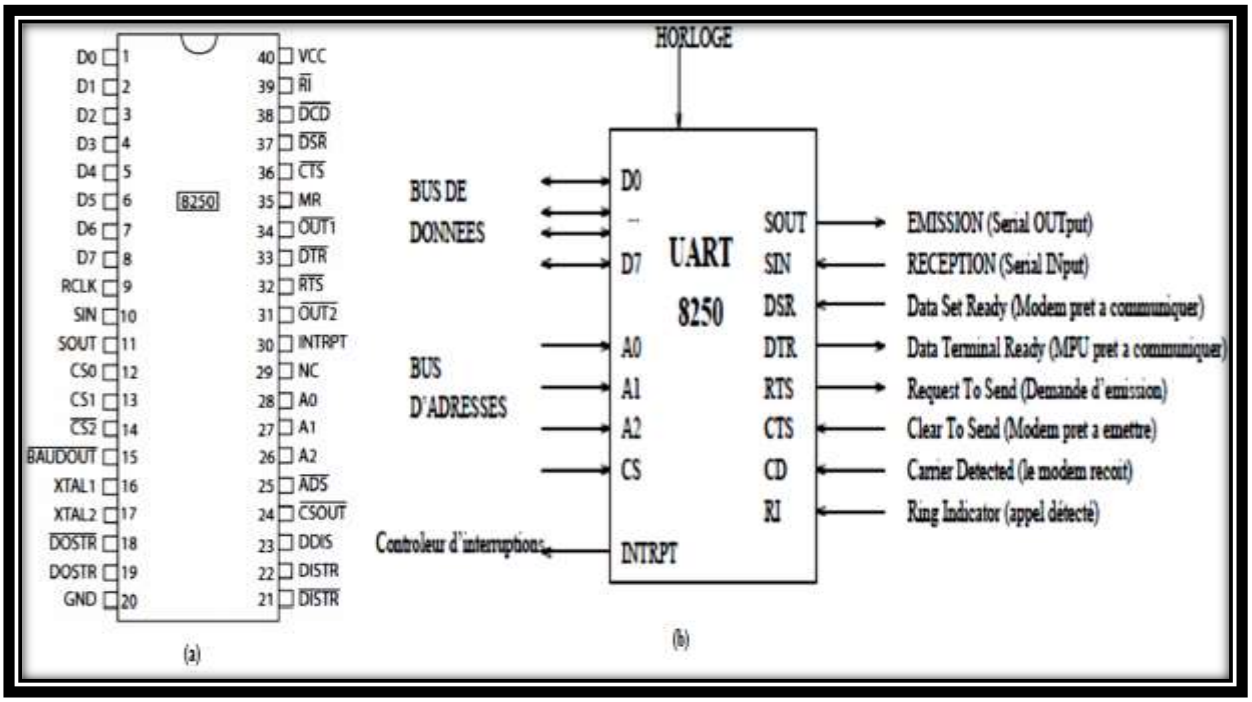


Figure (IV.22) : (a): Brochage du 8250 (b): Bornes du circuit UART 8250

Signal	Signification
T×D	Transmit Data
R×D	Receive Data
RTS	Request to send
CTS	Clear to Send
DTR	Data Terminal Ready
DSR	Data Set Ready
DCD	Data Carrier Detect
RI	Ring Indicator

En fonction de l'état de DLAB (Divisor Latch Access Bit), on a accès soit au registre d'émission / réception, soit au diviseur d'horloge, soit au masque d'interruptions.

Adresse				Registre
DLAB	**A2**	**A1**	**A0**	
0	0	0	0	**RBR** : Receive Buffer(registre de réception)
0	0	0	0	THR: Transmitter Holding Register (registre d'émission)
1	0	0	0	DLL : Divisor Latch LSB (poids faible diviseur horloge)
1	0	0	1	DLM : Divisor Latch MSB (poids fort diviseur horloge)

0	0	0	1	IER:Interrupt Enable Register
X	0	1	0	IIR:Interupt Identification Register
X	0	1	1	LCR:Line Control Register
X	1	0	0	MCR: Modem Control Registre
X	1	0	1	LSR:Line Status Registre
X	1	1	0	MSR: Modem Status Register
X	1	1	1	Not used

Dans ce cours on ne va pas étudier le fonctionnement du 8250 ni sa programmation car d'habitude elle n'est pas utilisée d'une façon individuelle mais elle fait partie du système de communication.

CHAPITRE V

V.1 Introduction

Les microcontrôleurs PIC de la société Microchip, sont depuis quelques années dans le « hit parade » des meilleures ventes. Ceci est en partie dû à leur prix très bas, leur simplicité de programmation, et les outils de développement que l'on trouve sur le NET.

V.2 Généralités sur le microcontrôleur

Aujourd'hui, développer une application avec un PIC n'a rien d'extraordinaire, et tous les outils nécessaires sont disponibles gratuitement.

Les fonctions demandées en vue de réalisation sur les PICs sont de plus en plus complexes, et par la suite, les programmes pour les réaliser demandent de plus en plus de mémoires, l'utilisateur est ainsi à la recherche de langages « évolués » pouvant simplifier la tache de programmation.

V. 2.1 Définition d'un PIC

Un PIC n'est rien d'autre qu'un microcontrôleur, c'est à dire une unité de traitement de l'information de type microprocesseur à laquelle on a ajouté des périphériques internes permettant de réaliser des montages sans nécessiter l'ajout de composants externes. La dénomination PIC est sous copyright de Microchip, donc les autres fabricants ont été dans l'impossibilité d'utiliser ce terme pour leurs propres microcontrôleurs.

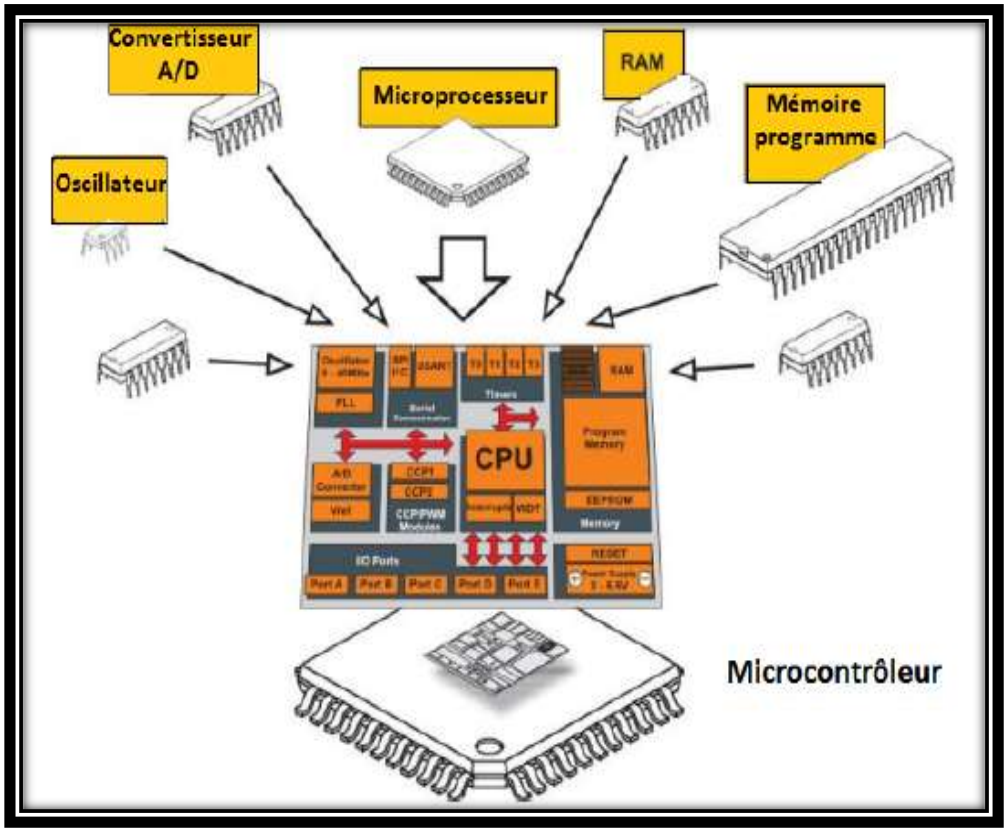


Figure (V.1) : Contenu type d'un microcontrôleur

V.2. 2 Les différentes familles des PICs

La famille des PICs est subdivisée à l'heure actuelle en 3 grandes familles : La famille Base-Line, qui utilise des mots d'instructions (nous verrons ce que c'est) de 12 bits, la famille Mid-Range, qui utilise des mots de 14 bits (et dont font partie les 16F84 et 16F876), et puis la famille High-End, qui utilise des mots de 16 bits

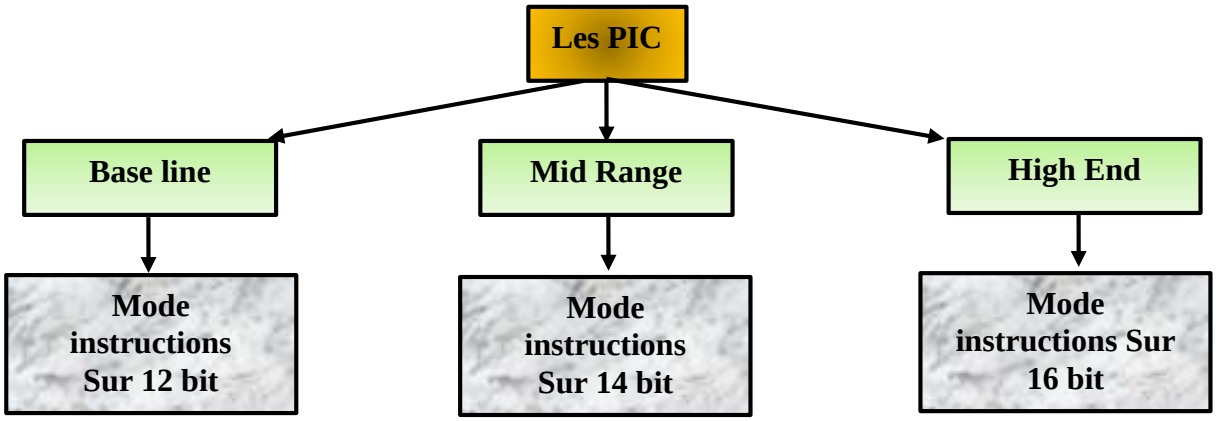


Figure (V.2) : Les différentes familles des PICs

V. 2.4. Identification d'un PIC

Pour identifier un PIC, vous utiliserez simplement son numéro.

PIC **xx (L) XX YY-ZZ**

Les deux premiers chiffres

indiquent la catégorie du PIC, 16 indique un PIC Mid-Range. Vient ensuite parfois une lettre L : Celle-ci indique que le PIC peut fonctionner avec une plage de tension beaucoup

☞ xx : Famille du composant (12, 14 , 16 , 17, 18)

☞ L: Tolérance plus importante de la plage tension

☞ XX : Type de mémoire de programme, XX prendre la lettre

✓ C :pour une EPPROM ou EEPROM

✓ CR : pour une PROM

✓ F : pour une mémoire de type Flash

À noter à ce niveau que seule une mémoire FLASH ou EEPROM est susceptible d'être effacée, donc les PICs de type CR ne peuvent être pas reprogrammés. Pour les versions C, le 16C84 peut être reprogrammé, il s'agit d'une mémoire eeprom. Le 12C508, par exemple, possède une mémoire programme EPROM, donc effaçable uniquement par exposition aux ultraviolets. Donc, l'effacement nécessite une fenêtre transparente sur le chip, qui est une version spéciale développement, et non la version couramment rencontrée.

☞ YY : identification

☞ ZZ : vitesse quartz de cadencement du Microcontrôleur

Finalement vous verrez sur les boîtiers le suffixe « -ZZ » dans lequel ZZ représente la fréquence d'horloge maximale

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

V.3 Architecture du microcontrôleur :

Nous nous limiterons dans ce support de cours à la famille Mid-Range et particulièrement au PIC 16F84, sachant que si on a tout assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur

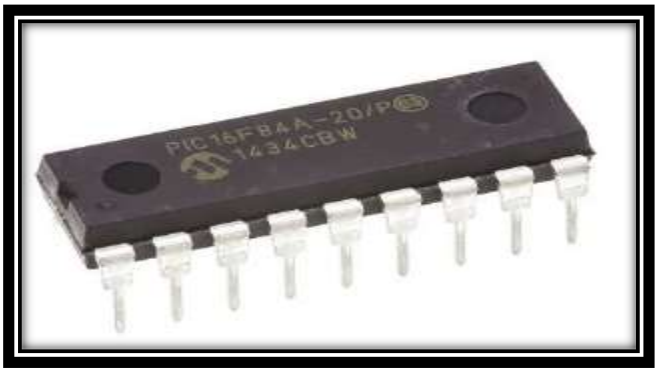


Figure (V.3) : Photo de MIC 16F84A

V.3.1 Présentation de PIC 16F84 :

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

L'organisation générale du PIC 16F84 est composée par 4 blocs principaux comme le montre la figure (V.4) :

-Mémoire de programme

-Mémoire de données

-Processeur -Ressources auxiliaires (périphériques)

La mémoire de programme contient les instructions pilotant l'application à laquelle le microcontrôleur est dédié. Il s'agit d'une mémoire non volatile (elle garde son contenu, même en l'absence de tension), elle est de type FLASH c'est-à-dire qu'elle peut être programmée et effacée par l'utilisateur via un programmateur et un PC. La technologie utilisée permet plus de 1000 cycles d'effacement et de programmation. Pour le PIC 16F84 cette mémoire est d'une taille de 1024*14 bits, c'est à dire qu'elle dispose de 1024 emplacements (de 000h à 3FFh) contenant chacun 14 cases car dans le cas du PIC, les instructions sont codées sur 14 bits. On peut donc stocker 1024 instructions.

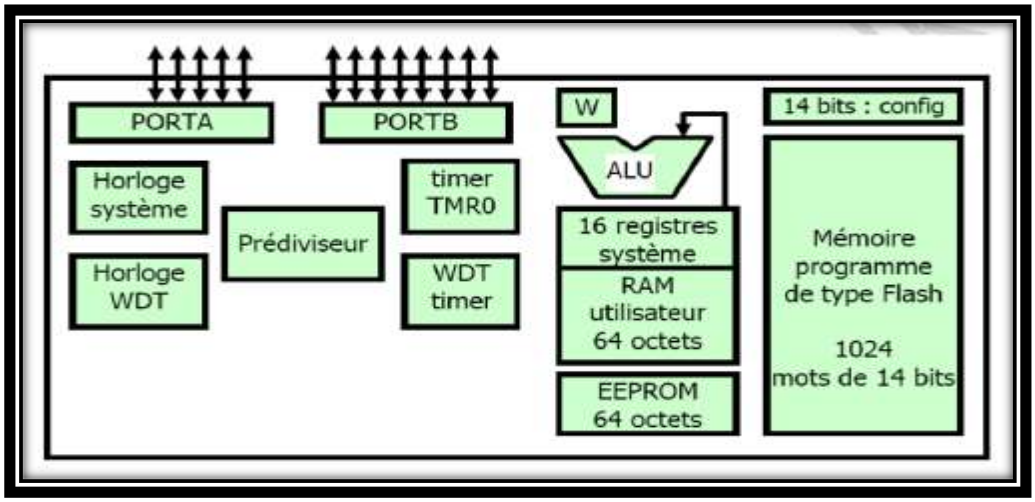


Figure (V.4) : Architecture interne de PIC 16F84A

☞ La mémoire de donnée est séparée en deux parties :

✓ Une mémoire RAM constituée de 64 octets. Elle est organisée en deux banques. On y trouve des registres spéciaux et des cases mémoires libres. Deux banques de 64 octets auxquelles on peut accéder soit de la banque 0 ou la banque 1. Les registres spéciaux servent à provoquer un fonctionnement spécial du PIC.

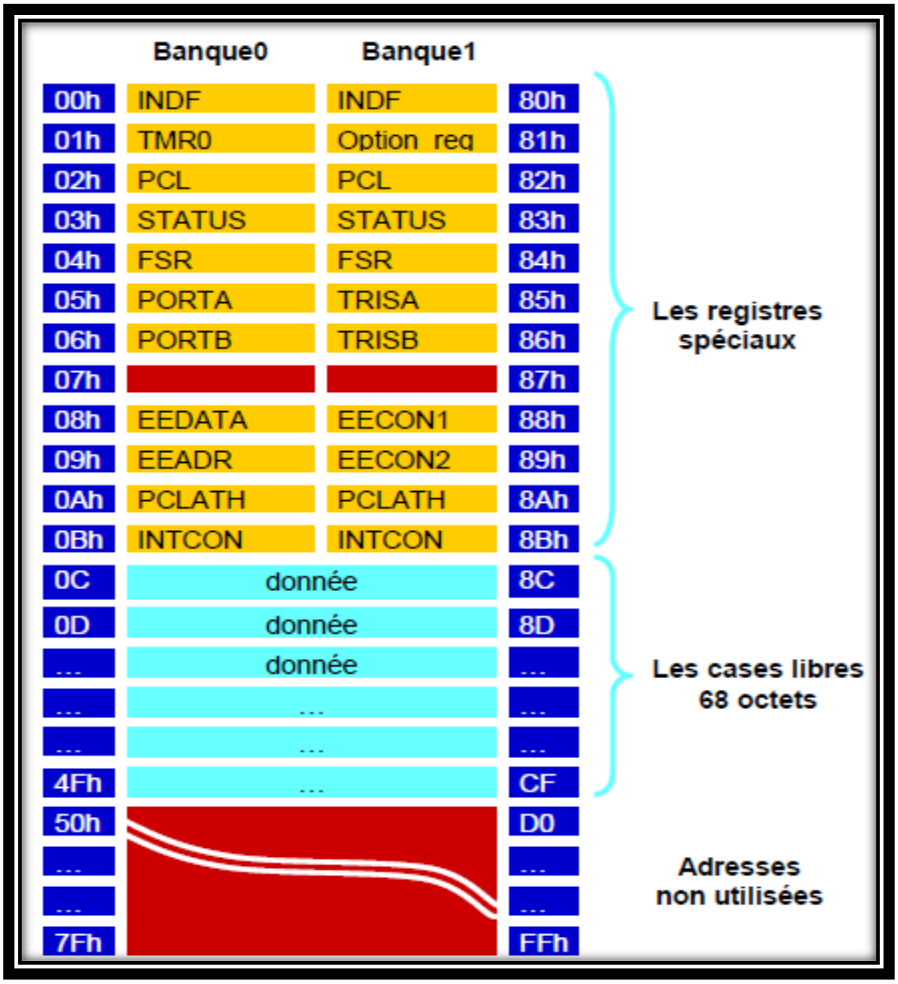


Figure (V.5) : Organisation de la RAM

✓ Une mémoire EEPROM de 64 octets dans laquelle on peut lire et écrire des données (de huit bits soit un octet) et qui possède l'avantage d'être non volatile (les données sont conservées même en l'absence de tension). La lecture et l'écriture dans cette mémoire de données sont beaucoup plus lentes que dans la mémoire de données RAM.

☞ Le processeur est formé de deux parties :

✓ une unité arithmétique et logique (ALU) : C'est cette partie qui effectue physiquement toutes les actions internes dictées par le Contrôleur et Décodeur d'Instructions. Par exemple, l'ALU peut effectuer une addition, une soustraction ainsi que les opérations logiques telles que « ET », « OU », etc. Pour effectuer toutes ces opérations .

✓ un registre de travail noté W sur lequel travail l'ALU.

☞ Les ressources auxiliaires qui sont dans le cas du PIC16F84

✓ ports d'entrées et de sorties.

✓ temporisateur.

✓ interruptions

✓ chien de garde

✓ mode sommeil

V.3.2 Brochage et caractéristiques principales

Le PIC16F84 est un circuit intégré de 18 broches (Fig. (V.5)) :

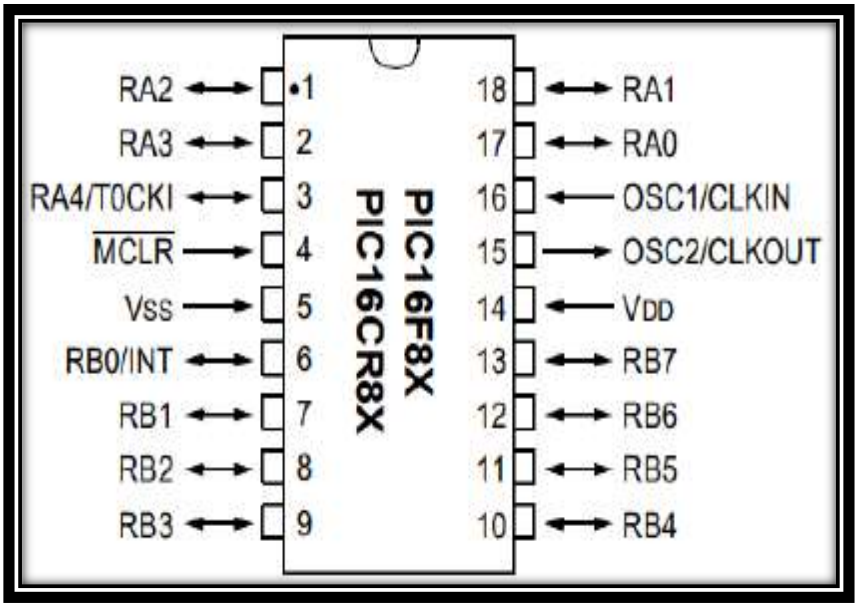


Figure (V.6) : Brochage de MIC 16F84A

Les principaux constituants du PIC16F84A sont :

✓ **Vss** : La masse 0Volts

✓ **VDD** : l'alimentation qui doit délivrer une tension continue comprise entre 3 et 6 Volts

✓ **OSC** 1,2 : Horloge

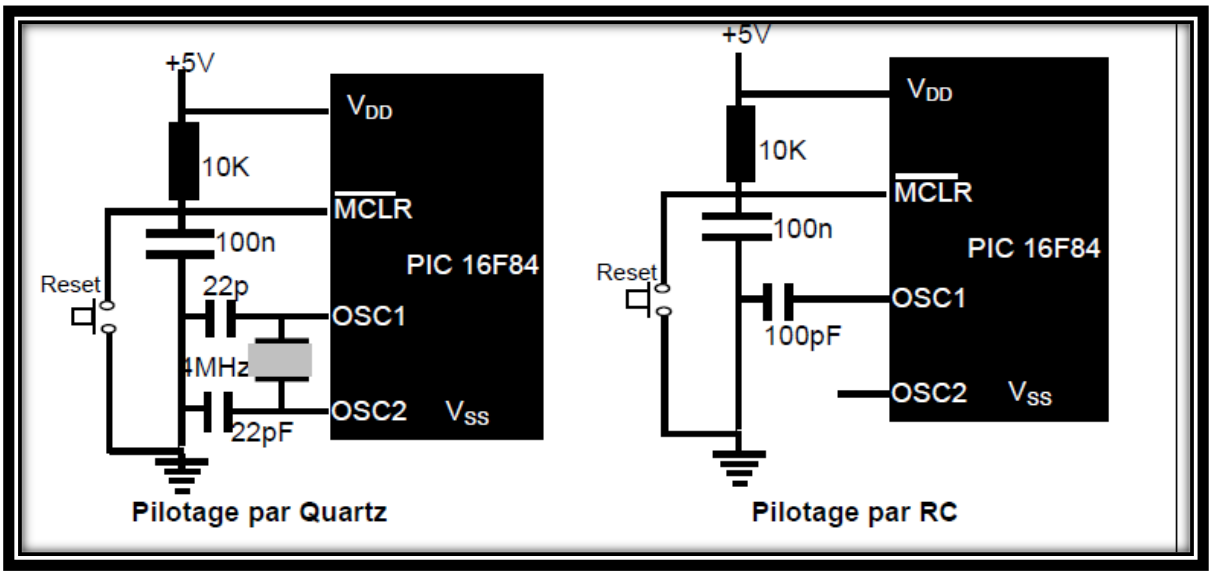


Figure (V.7) : Brochage de MIC 16F84A

Valeur de capacité avec les résonateurs à quartz présente sur le tableau suivant :

Mode	fréquence	OSC1/C1	OSC2/C2
LP (Low-Power Crystal)	32kHz	68-100 pF	68-100 pF
	200kHz	15-30 pF	15-30 pF
XT (Crystal/Resonator)	100kHz	68-150 pF	150-200 pF
	2MHz	15-33 pF	15-33 pF
	4MHz	15-33 pF	15-33 pF
HS (High-Speed Crystal/Resonator)	4MHz	15-33 pF	15-33 pF
	10MHz	15-47 pF	15-47 pF

✓ **RA 0-4** : Port A

✓ **RB 0-7** : Port B

✓ **T0 CKL** : Entrée de comptage

✓ **INT** : Entrée d'interruption

✓ **MCLR** (**Reset : 0V**) : Elle permet lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur. C'est à dire que si un niveau bas (0 Volt) est appliqué sur MCLR le microcontrôleur s'arrête, place tous 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). A la mise sous tension, la patte MCLR étant à zéro, le programme démarre donc à l'adresse 0000, (MCLR=Master CLear Reset).

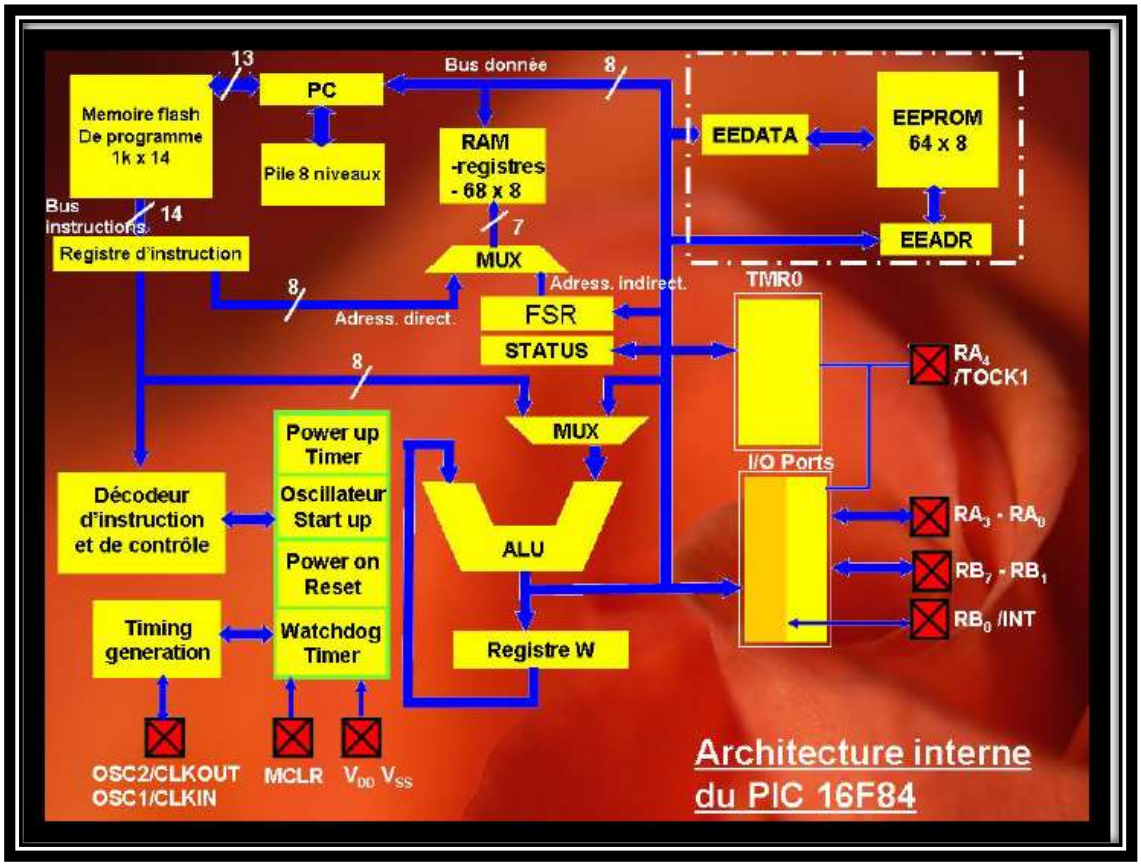


Figure (V.8) : Architecture interne détaillé de PIC 16F84A

V.4 La programmation des microcontrôleurs :

V.4.1 Le jeu instructions :

Le jeu d'instruction de programmation en assembleur du PIC et c'est là que réside tout l'intérêt puisqu'il ne dispose que de 35 instructions qui lui permettent de réaliser toutes les tâches plus que sa les instructions sont codées sur 14 bits.

On peut distinguer trois familles d'instruction :

☞ Instructions sur les registres : octet

☞ Instructions sur les registres : bit

☞ Instructions de contrôle et de saut

V.4.1.1 Instructions sur les registres : octet

Se sont les instructions qui manipulent les données, elles sont codées comme suit:

✓ Les 6 bits de poids fort contiennent le code de l'instruction (opcode)

✓ Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)

✓ Le bit 7 indique où sera placé le résultat de l'opération:

Bit 7=0 -> le résultat sera dans le registre de travail (W)

Bit 7=1 -> le résultat sera dans la même case mémoire RAM (ou le registre)

Instruction orientées octet													
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode						0/1	Adresse du registre concerné						

V.4.1.2 Instructions sur le bit d'un registre :

Ce sont les instructions qui manipulent directement les bits d'un registre, elles sont codées comme suit :

✓ Les 4bits de poids fort contiennent le code de l'instruction (opcode)

✓ Les bits 7, 8 et 9 déterminent le bit concerné par l'opération (bit 0 à bit 7)

✓ Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)

Instruction orientées octet													
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode				Bit concerné			Adresse du registre concerné						

Exp : **BSF Port b, 3**

mnémonique	Description	cycle	bits
ADDWF f,d	d:=W+f	1	C,DC,Z
ANDWF f,d	d:=W AND f	1	Z
CLRF f	f:=0	1	Z
CLRW	W:=0	1	Z
COMF f,d	d:=NOT(f)	1	Z
DECF f,d	d:=f-1	1	Z
DECFSZ f,d	d:=f-1 ; Skip if Zero	1(2)	.
INCF f,d	d:=f+1	1	Z
INCFSZ f,d	d:=f+1 ; Skip if Zero	1(2)	.
IORWF f,d	d:=W OR f	1	Z
MOVF f,d	d:=f (permet de savoir si f=0 en faisant	1	Z
MOVWF f	W:=f	1	.
NOP	n'effectue aucune opération	1	.
RLF f,d	d=f SHR 1	1	C
RRF f,d	d=f SHL 1	1	C

SUBWF f,d	d:= f-W (en complément à 2)	1	C,DC,Z
SWAPF f,d	d:= f[4..7] <--> f[0..3] (inverse les quartets)	1	.
XORWF f,d	d:= W XOR f	1	Z

Instructions sur les registres (bit par bit)

mnémonique	Description	cycle	bits affect
BCF f,b	f[b]:=0 (mets à 0 le bit b de f)	1	.
BSF f,b	f[b]:=1 (mets à 1 le bit b de f)	1	.
BTFSC f,b	teste le bit b de f ; Skip if Clear (0)	1(2)	.
BTFSS f,b	teste le bit b de f ; Skip if Set (1)	1(2)	.

V.4.1.3 Instructions générales de contrôle

Ce sont les instructions qui manipulent les données qui sont codées dans l'instruction directement:

- Les 6 bits de poids fort contiennent le code de l'instruction (Opcode)
- Les 8 bits de poids faible contiennent la valeur numérique ou littérale qui sera utilisée par l'instruction.

Instructions littérales et de contrôle													
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode						Valeur utilisée							

Exemple : **Movlw 01010101b**

V.4.1.4 Instructions de saut et appel

Ce sont les instructions qui provoquent une rupture dans la séquence de déroulement du programme qui est écrit dans la mémoire programme (1k) 10bits

Instructions CALL et GOTO													
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode			Adresse de destination										

Instructions de contrôle et de saut

mnémonique	Description	cycle	bits affect
ADDLW k	W:=W+k	1	C,DC,Z
ANDLW k	W:=W AND k	1	Z
CALL k	appel un sous programme	2	.

CLRWDT	**remet à 0 le timer du chien de garde**	**1**	**TO,PD**
GOTO k	**se branche à l'adresse k**	**2**	**.**
IORLW k	**W:=W OR k**	**1**	**Z**
MOVLW k	**W:=k**	**1**	**.**
RETFIE	**fin d'une interruption**	**2**	**.**
RETLW k	**W:=k , puis effectue un retour de sous programme**	**2**	**.**
RETURN	**effectue un retour de sous programme**	**2**	**.**
SLEEP	**circuit en mode sommeil et stoppe l'oscillateur**	**1**	**TO,PD**
SUBLW k	**W:= k-W**	**1**	**C,DC,Z**
XORLW k	**W:=W XOR k**	**1**	**Z**

V.5 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 spécifique.

La séquence classique de fonctionnement d'une interruption :

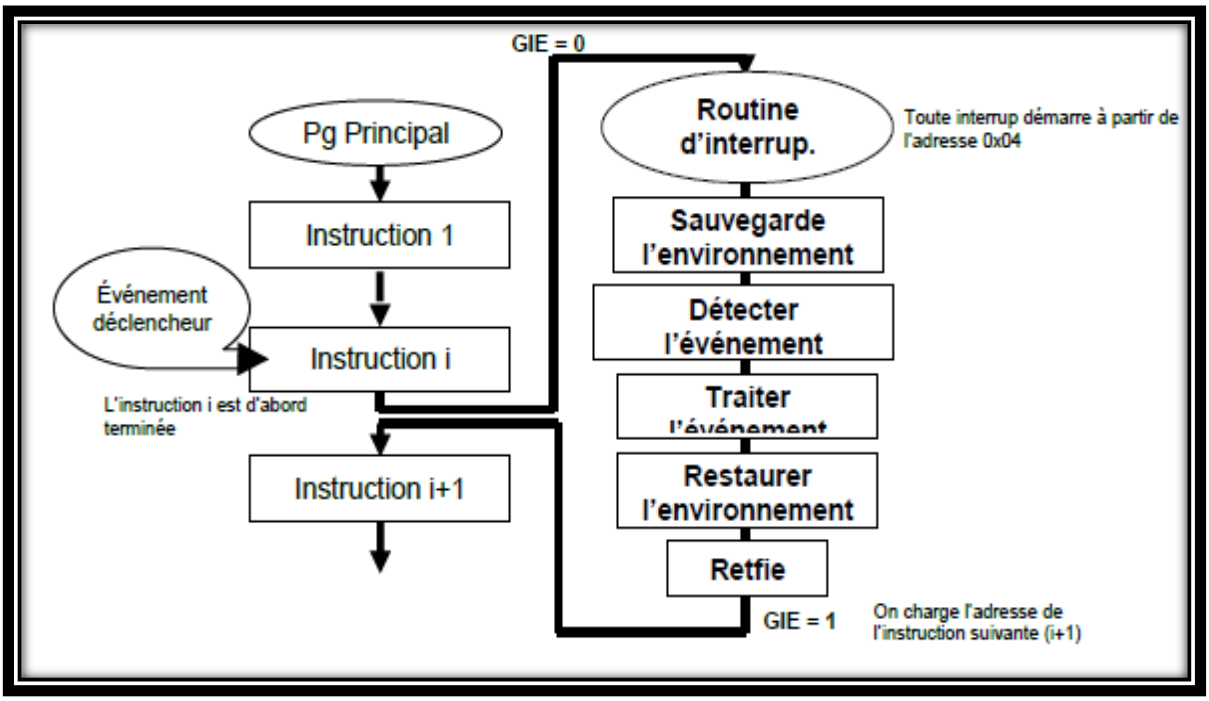


Figure (V.8) : Exemple d'Interruptions

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

☞ 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

a) Le registre INTCON :

Le registre qui gère les interruptions est le registre INTCON qui est un registre de bit :

GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

✓ GIE : permet la validation générale des interruptions

✓ EEIE : permet la validation de l'interruption de EEPROM

✓ TOIE : permet la validation de l'interruption TMR0

✓ INTE : permet la validation de l'interruption sur la broche RB0/INT

✓ RBIE : permet la validation de l'interruption RB4 à RB7

✓ TOIF : bit signal le débordement du TMR0

✓ INTF : bit signal une transition sur RB0/INT

✓ RBIF : bit signal un changement d'état de l'une des entrées RB4 à RB7

☞ Les 3 bits de signal doivent être remis à 0 par le pg.

☞ Le bit qui manque EEIF qui signal fin de l'écriture EEPROM se trouve dans le registre EECON1

b) Sauvegarde et restauration de l'environnement

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

En général c'est :

✓ STATUS

✓ Le registre de travail W

Le registre PC est sauvegardé automatiquement.

Pour sauvegarder W :

movwf w_temp ; il faut réserver un octet dans la RAM dans la zone des déclarations

Pour sauvegarder STATUS:

movf STATUS , w ; écrire la valeur de STATUS dans W

movwf sauve_Status ; sauvegarder STATUS dans sauve_Status

Malheureusement la 1ère instruction modifie le STATUS !!!

Pour remédier on utilise l'instruction SWAP qui permet d'inverser les 4 bits de poids faibles avec ceux de poids forts d'un octet.

Swapf STATUS , W ; swap STATUS dans W

movwf Status_temp ; sauve W dans Status_temp

Pour restaurer STATUS

Swapf Status_temp , W ; ancien STATUS dans W

movwf STATUS ; restaure STATUS

Pour restaurer W il faut faire attention pour ne pas modifier le registre STATUS :

Swapf w_temp , f ; swap w-temp dans elle même

Swapf w_temp , W ; swap w_temp dans W on obtient l'ancien valeur de W

c) Résumé : La structure de base d'une routine d'interruption

Routine d'interruption

Sauvegarde des registres

Org 0x04 ; adresse des interruptions

movwf w_temp ; sauve W

swapf STATUS , w ; sauve 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 ; restaure STATUS

movwf STATUS ; -- -- -- -- -- -- -- --

swapf w-temp , f ; restaure W

swapf w_temp , W ; -- -- -- -- -- --

retfie ; retour de l'interruption (GIE =1 automatiquement)

V.6 Mise en cas pratique Mplab:

Dans cette partie en donne un simple exemple de fonctionnement 16F84A via logiciel MPLAB Microship .Soit le schéma électronique de figure (V.9)

La figure (V.9) représente la réalisation dans le but de lire et afficher l'état de chaque interrupteur (I1 sur D1, I2 sur D2, I3 sur D3 et I4 sur D4)

☞ Toujours en début on met en évidence l'organigramme.

☞ Choisir le pic

☞ Configuré le pic

☞ Org

☞ Initialisation : configuration des ports

☞ Et dans le programme principal on a :

➢ Lire l'état des interrupteurs sur le PORTB

➢ Ecrire l'état des interrupteurs sur le PORTA

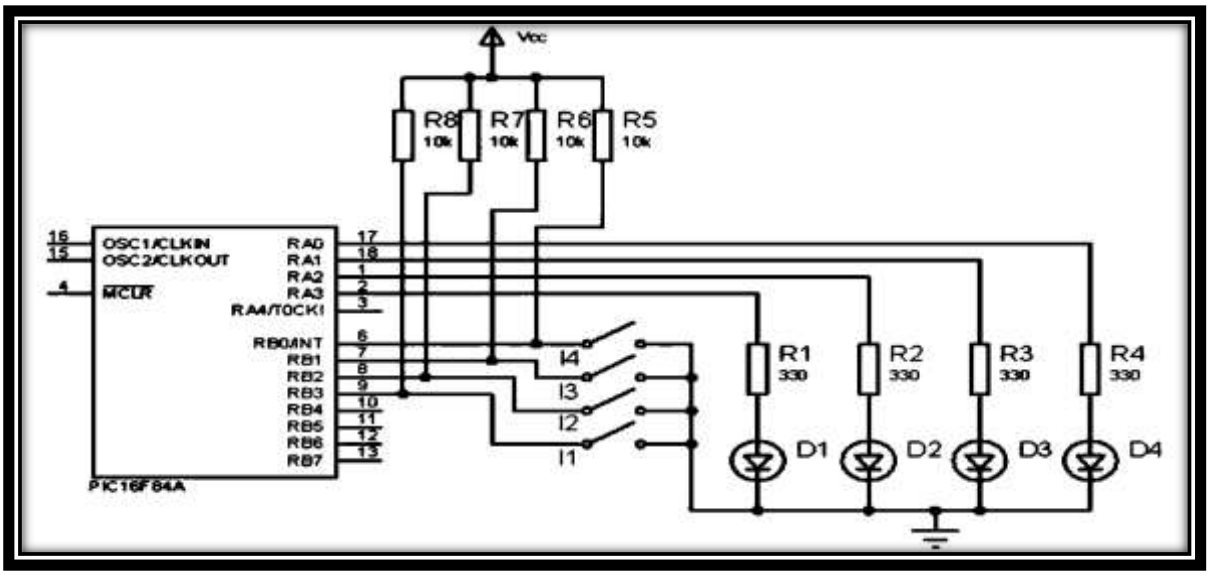


Figure (V.9) : Schéma électronique a base microcontrôleur 16F84A

V.6.1 Organigramme :

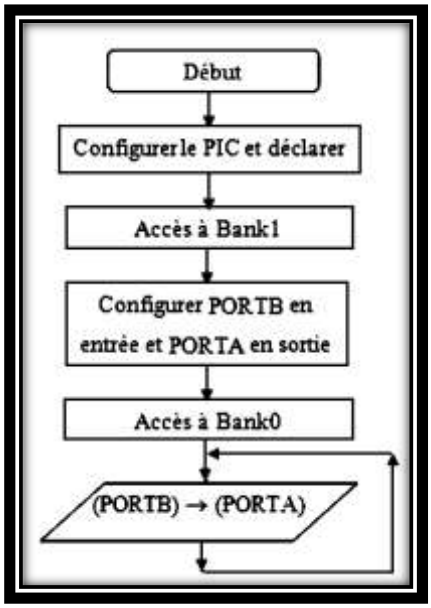


Figure (V.10) : Organigramme de programmation de 16F84A dans le bit de fonctionnement du montage de figure (V.9)

V.6.2 Choisir le pic :

✓ list p=16F84A

✓ #include <p16f84A.inc>

V.6.3 Configuration de pic :

__config _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

Le tableau suivant représente la description de chaque élément de paramètre de configuration

Nom	Valeurs possibles	Signification
CP (Code Protection bit)	ON	Rend impossible la lecture de la mémoire de programme Flash et de l'EEPROM (à travers un programmateur) C'est une protection contre le piratage industriel.
	OFF	Lecture possible
PWRTE (Power-up Timer Enable bit)	ON	A la mise sous tension du µC, lance une temporisation d'environ 72 ms durant laquelle est effectué un RESET interne. Il est conseillé d'utiliser cette configuration.
	OFF	Temporisation désactivée
WDT (Watchdog Timer Enable bit)	ON	Active le watchdog (chien de garde)
	OFF	Désactive le watchdog
OSC (OscillatorSelection bits)	RC	Oscillateur de type Résistance / Condensateur Remarques : économique, réservé aux applications où la précision de la base de temps n'est pas critique
	HS	Oscillateur à quartz haute fréquence (4 MHz à 20 MHz).
	XT	Oscillateur à quartz ou à résonateur céramique
	LP	Oscillateur à quartz de faible puissance (32,768 kHz)

ORG : détermine où le compilateur doit mettre les codes qui suivent cette commande.
ORG 0x000; Adresse de départ après reset

Go to init

V.6.4 configuration des ports :

Les registres PORTA et PORTB se trouvent dans la banque 0 (bank 0) de la mémoire de données, et les registres TRISA et TRISB se trouvent dans la banque 1 (bank 1) de la mémoire de données.

Init

Bsf STATUS,RP0 ; sélection le bank 1

Clrf TRISA ; port A en sortie en peut remplacer par **Movlw 0X00** et **Movwf TRISA**

Movlw 0XFF ; W=1111 1111

Movwf TRISB; PORTB=W=1111 1111 alors le port a configurée en entrée

Bcf STATUS,RP0 ; selection le bank 0

V.6.5 Programme principal

Debut

Movf PORTB,W
MOVLW PORTA
Goto debut
End

```
     LIST      p=16F84A              ; Définition de processeur
     #include <p16F84A.inc>         ; Définitions de variables

     __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
;-------------------------------- Démarrage sur reset ----------------------------------------------

     org 0x000             ; Adresse de départ après reset
     goto    init          ; Adresse 0: initialiser

;-------------------------------- Initialisation ---------------------------------------------------
init
     bsf     STATUS,RP0    ; sélectionner banque 1
     clrf    TRISA         ; PORTA en sortie (banque1)
     movlw   0xFF          ;
     movwf   TRISB         ; PORTB en entrée(banque1)
     bcf     STATUS,RP0    ; repasser banque 0

;-------------------------------- Programme principal ----------------------------------------------
debut
     movf    PORTB,w       ; (PORTB) --> (w)
     movwf   PORTA         ; (w) --> (PORTA)
     goto    debut         ; boucler à l'infini
     end
```

Figure (V.11) : le programme assemblaire total d'Organigramme de figure (V.10)

EXERCICES PROPOSER AVEC CES SOLUTIONS

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 01

a) Convertir en nombres décimaux les nombres binaires suivants : 11, 1101, 100101110.

b) Convertir en nombres binaires les nombres décimaux suivants : 7, 51, 128, 131, 234.

c) Convertir en nombres binaires puis en nombres décimaux les nombres hexadécimaux suivants : 12, *DEBA* et 5*F*3.

d) Quels sont les entiers naturels que l'on peut représenter sur un octet ?

Exercice 02 : Soit le µp 8086 déterminé les indices suivant :

AX :	BX :	CX :
DX :	CS :	DS :
ES :	SS :	

1) Que représente chaque indice

2) Déterminer les registres de travail est les registres de segment

3) Cité les Registres de pointeurs et d'index

4) Définir les flags CF SF OF et ZF

5) Dans un ordinateur quelle est la différence entre un registre et une mémoire ? Comparer la nature, la taille, la vitesse et le moyen d'accès aux informations.

Exercice03 : Soit les indices suivantes :

CS : 1234H	DS : 5672H	SS : 9AB5H
SP : 1234H	IP : 5678H	SI : 9ABCH

Déterminé :

1) Adresse logique de data segment
2) Adresse physique de code segment
3) minimal et maximal de CS

Exercice 04 : Soit le programme assembleur suivant :

```
MOV AL,0DFh
AND AL,11011111b
INC AL
MOV AH,0CCh
OR AH,20h
DEC AH
```

1) Expliqué chaque ligne de programme

2) Détermine le contenue des registres AL , AH et AX

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 05 : Écrire un programme en assembleur 8086 qui calcule la somme de deux nombre A et B. Dans un premier temps, vous supposez que ces deux valeurs sont initialisées dans votre programme.

Exercice 06 :

Écrire un programme en assembleur 8086 qui détermine comparaissant entre A <B, A>B, A=B

Exercice 07

1) Quelles sont les instructions permettant d'accéder à la pile ?

2) Qu'est-ce un registre ?

3) Qu'est-ce un code d'instruction (de quoi est composé)?

4) Quel est l'opération permettant de transcrire un programme du code symbolique (code source) en code machine (ou code objet)

5) Quelle est la différence fondamentale entre une interruption matérielle une interruption logiciel ?

6) Qu'est-ce qu'un code d'instruction ?

7) Quels sont liens physiques entre le processeur et la mémoire ?

Exercice 08 :

Soit µp 8086 :

Ecrire un programme assembla ire qui calcul la somme n

Exercice 09 :

Soit µp 8086 :

Ecrire un programme assembla ire qui calcul la factorielle n

Exercice 10:

Soit le programme suivant:

```
MOV AX,01FFFH
MOV SS,AX
MOV AX,2000H
MOV DS,AX
MOV AX,3000H

MOV AX,02FC3H
MOV DI,AX
MOV AX,060H
MOV [DI],AX
MOV AX, 0DA41H
MOV bp,ax
MOV AX,080H
```

MOV [bp],aX

MOV AX,4000H

MOV ES,ax

MOV AX, 06C71H

MOV SI,ax

MOV AX,050H

MOV [SI],AX

MOV AX,04A41H

MOV bp,ax

MOV aX,0A0H

MOV [BP],AX

MOV BX,200H

MOV SP,bx

MOV AX, 030H

MOV CX, 05H

ESE: PUSH AX

LOOP ESE

HLT

D'après ce programme

1) Détermine les adresses physique minimal (lower) et maximal (hypper) de ES,SS,DS

2) Quelle et le rôle des instructions PUSH et POP

3) Détermine les adresses logiques des valeurs de **060H, 080H ,050H ,0A0H et 30H**

Exercice 11 :

Soit la figure (a) de représente chronogramme un mot de deux octet (5V:1 logique ,0V:0 logique) en va coder sou forme le chronogramme (b)

1) Ecrire un programme assembla ire de µP8086 qui coder la figure (a) au figure (b) puis décoder la figure (b) en (a)

2) Ecrire un programme assembla ire de µP8086 qui détermine le nombre des bits de séquence 11

3) Ecrire un autre programme assembla ire de µP8086 qui détermine le nombre des bits de séquence 10

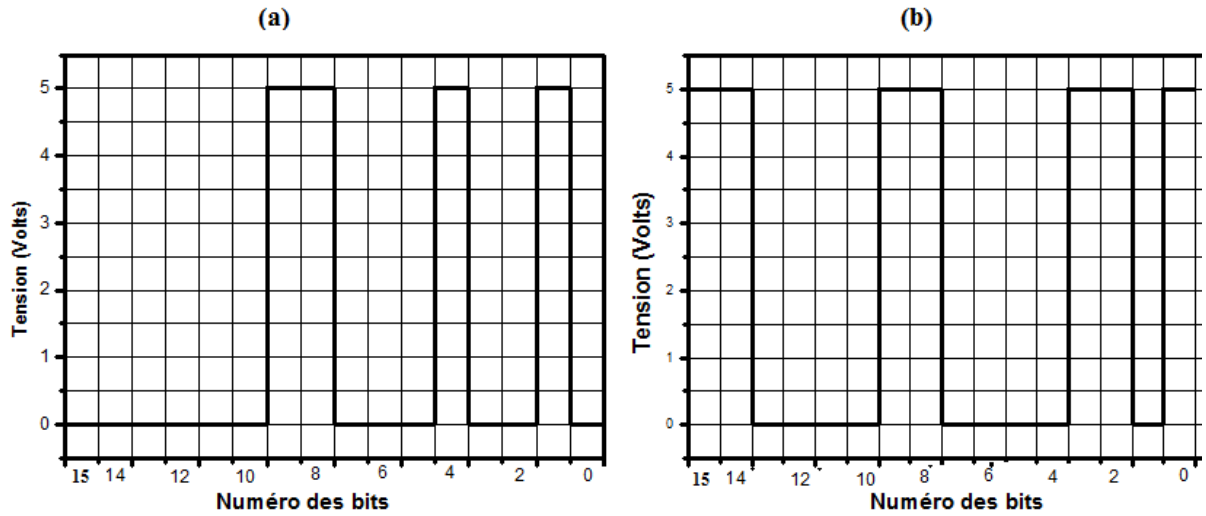
Concernant les question 2 et 3 en prendre l'exemple de 8F1FH

Exemple soit un mot binaire 1100111011111011

Le nombre de séquence 11 et égale à 8

EXERCICES PROPOSER AVEC CES SOLUTIONS

Le nombre de séquence 10 et égale à 3



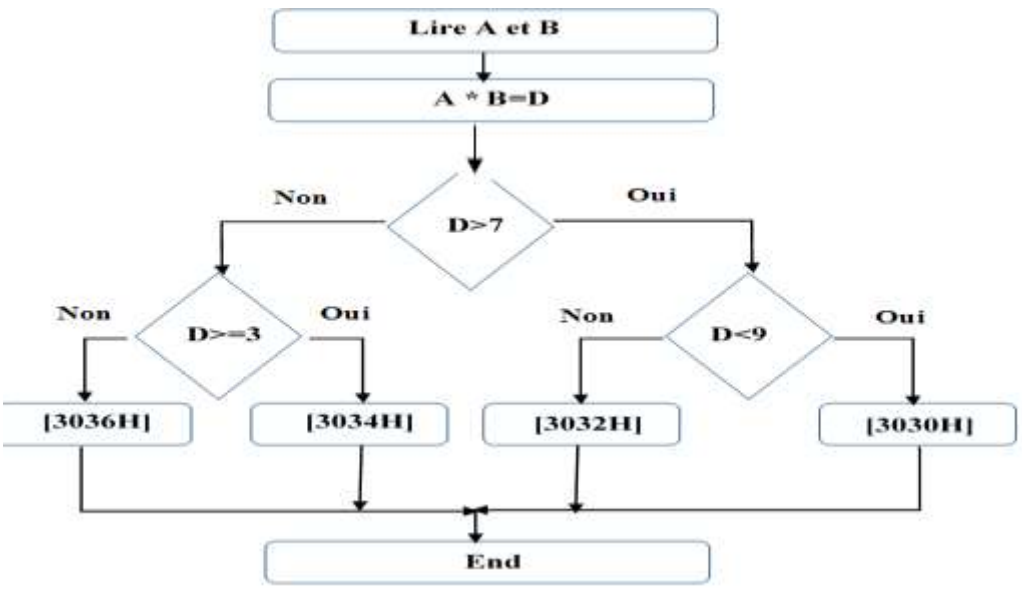
Exercice 12 :
Soit les instructions d'adressage de µp 8086 suivant :

1) MOV CX, DX
2) MOV AL,Gama
3) MOV [Bp+12], CL
4) MOV Al, 60h
5) MOV [DI], DX
6) MOV [DI + 0500],Ax.
7) MOV DX, [Bp+SI+5]
8) MOV AX, [BX+DI+9]

a) Expliqué ces instructions

b) Donnée le type d'adressage de chaque instruction

Exercice 13 :
Soit l'organigramme suivant :



1) Donner explication a un saut conditionnel et non conditionnelle donner l'exemple

2) Ecrire se organier gramme en langage assembleur µp8086

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 14:

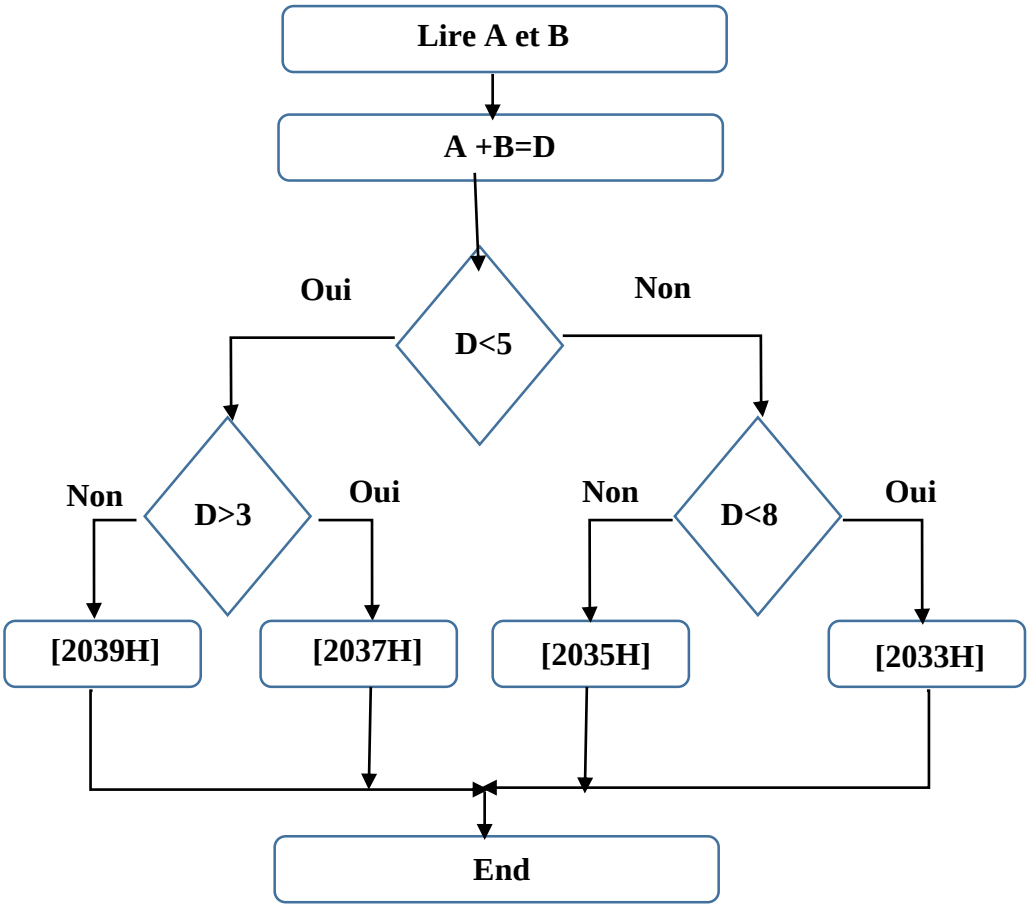
Soit les instructions d'adressage de µp 8086 suivant :

1) MOV CX, BX
2) MOV AL,TETA
3) MOV [Bp+09], CL
4) MOV Al, 09h
5) MOV [DI + 050] , Ax.

Donnée le type d'adressage de chaque instruction

Exercice 15 :

Soit l'organigramme suivant :



1) Ecrire se organier gramme en langage assembleur µp8086
2) Ecrire un programme qui calculer l'équation H :

$$H = \left(\prod_{N=1}^{4}(2N)\right) + \sum_{M=1}^{10}(M)$$

Exercice 16 :

1) Ecrire une seul programme assembla ire de µP 8086 Qui calcule la fonction F suite à des instructions suivant :

a) Si X=1 $F = \sum_{M=1}^{10}(M)$

b) Si X=2 $F = \prod_{N=1}^{4}(N)$

c) Si X=3 $F = (\prod_{N=1}^{4}(N)) \times \sum_{M=1}^{10}(M)$

Remarque : Introduire JE JAE JMP et CMP

2) Soit le programme suivant :

```
M DW 0
MOV CX,4
gh: CMP CX,0
JZ END
MOV bX,CX
MOV AX,1
ghh: CMP BX,0
JZ gf
mul bx
dec bx
jz gf
jmp  ghh
gf:ADD M,Ax
DEC CX
JMP gh
END:endp
```

a) Détermine la fonction F qui calcule ce programme

b) Détermine la valeur de M en Héxadicimal

Exercice 17:

Par utilisation de fonction CALL écrire programme assembla ire qui détermine les sommes suivant dans une programme est stocké dans une adresse logique DS:SI

$$F = \sum_{i=1}^{5} n$$

$$F = \sum_{i=1}^{6} n$$

$$F = \sum_{i=1}^{9} n$$

Exercice 17 :

Par utilisation de fonction CALL écrire programme assembla ire qui détermine la fonction F

Prendre Sp:100H

EXERCICES PROPOSER AVEC CES SOLUTIONS

$$F = \sum_{i=1}^{5} \prod_{N=i}^{5} N$$

Exercice 19 :

Soit le tableau suivant:

T1=[37H,02H,02H,07h,02H,39H,11h,04H,12H,77H,20H, 01H,31H,35h,32H,39H,36h,30H,39H,34H]

On début on a créé une procédure Nommé INITI doit être :

Charger le registre BX,00H

Charger le registre AX,00H

Charger le source d'indexe SI,00H

Charger le data d'indexe DI,00H

Charger le registre CX,14H

a) Créé la procédure INITI en programme assembla ire µp 8086

b) Quelle est le rôle de commande RET4 dans les procédures en langage assembla ire µp 8086

c) Ecrire un programme assembla ire µp 8086 qui déclaré et affiche Tableau T1 a l'exploitation procédure INITI

d) Dans la mémé programme écrire un programme assembla ire µp 8086 qui affiche un Tableau T2 ces éléments contiens les éléments de T1 qui est supérieure à 30h

e) Ecrire un programme assembla ire µp 8086 qui calculer la somme des éléments de Tableau T1qui est inférieure à 20H le résultat doit être dans variable SOM de taille 1 oK et rappelé la procédure INITI

f) Ecrire un programme assembla ire µp 8086 qui calculer le produit des éléments de Tableau T1qui est inférieure à 12H le résultat doit être dans variable PROD de taille 2 oK et rappelé la procédure INITI et stocké au adresse logique DS: SI qui est 1000:0300H

Exercice 20 :

Soit **six boitier**s de mémoire de 8X8 KO et un µc8086

1) Quelle et la déférence entre mémoire et le registre

2) Quelle le nombre de boitier de mémoire de 8 KO supporté par µP 8086

3) Quelle le nombre de bus d'adresse de chaque boitier par µP 8086

4) Quelle le nombre de bus d'adresse pour manipulé les quatre boitiers par µP 8086

5) Quelles et les adresses de débit et la fin de chaque boitier en saie que l'adresse de débit de première boité 80000H avec exploitation d'un décoda ire

Exercice 21 :

1) Quelle est le nombre maximum de ces boitiers mémoire qu'on connecter à un 8086

2) Soient deux boitiers mémoire de 64K, représenter clairement peut on les relier a un µp 8086

EXERCICES PROPOSER AVEC CES SOLUTIONS

a) A partir de l'adresse 00000H donner l'adresse de la dernière case mémoire

b) Soient deux boitiers mémoire de 128K, représenter clairement peut on les relier a un µp 8086 à partir de l'adresse E0000H

c) Donner l'adresse de la dernière case mémoire

Exercice 22 :

Soit interfaçage parallèle µp 8086 à CI 8255

1) Faites la configuration suivante (CW) :

PORTA en mode 1 en sortie.

PORTB en mode 0 en entrée.

PORTC en sortie.

Adresse de PORTA et 80H

2) Donné les adresse de PORTB, PORTC et CW

3) Faites la configuration en langage assembleur

Exercice 23 :

Écrire un programme assembler qui contrôler l'état et la sens direct inverse de moteur DC a laide µP8086 interfacer parallèlement CI 8255 en mode 0 (Voir le tableau et la figure suivant)

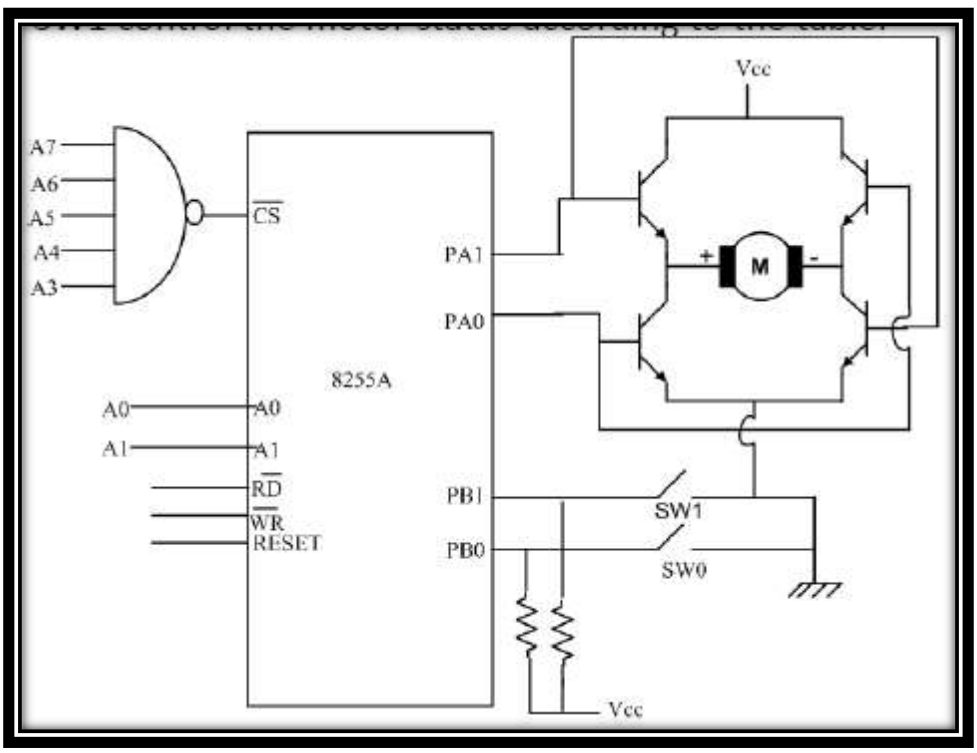


Tableau de fonctionnement :

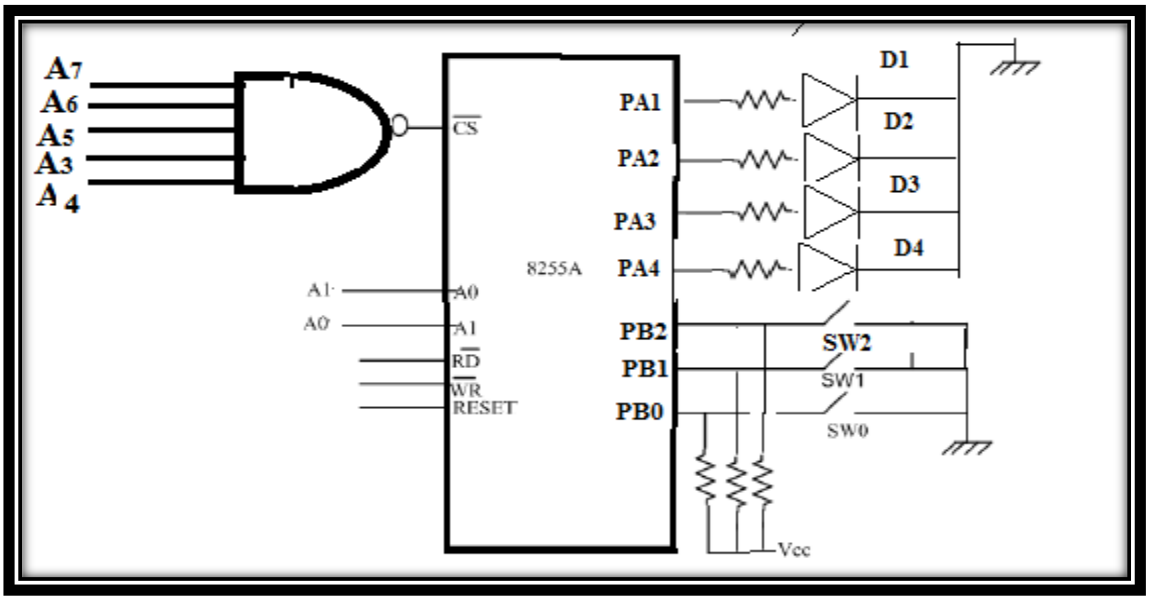
SW_1	SW_0	MOTEUR
0	0	En état arrêt
0	1	En état marche (DIRECT)

1	0	En état marche (INVERSE)
1	1	En état arrêt

Remarque pour écrire le programme il faut détermine la valeur de configuration (CW) et les adresse de PORTA , PORTB et le CW

Exercice 24:

Soit la figure suivant qui représente l'interfaçage parallèle de µp 8086 avec circuit intégré 8255A a Fréquence d'horloge de 5MHZ



Soit le fonctionnement de figure 1

SW2	SW1	SW0	L'état des leds(L)	Durée allumage
OFF	X	X	TOUT LES LEDS EN ETAT **OFF**	
ON	OFF	OFF	TOUT LES LEDS EN ETAT **ON**	1Second
ON	OFF	ON	TOUT LES LEDS EN ETAT **ON**	2Second
ON	ON	OFF	TOUT LES LEDS EN ETAT **ON**	3Second
ON	ON	ON	TOUT LES LEDS EN ETAT **ON**	4Second

1) Détermine les adresses des ports A, B, C et CW (Control Word)

2) Détermine la valeur de CW mode registre en mode**)** 00

3) Soit le procedure DELAY

DELAY PROC NEAR

MOV BX, 64H

PULSE: MOV CX, 50000

HERE: LOOP HERE

DEC BX

JNZ PULSE

EXERCICES PROPOSER AVEC CES SOLUTIONS

RET 4

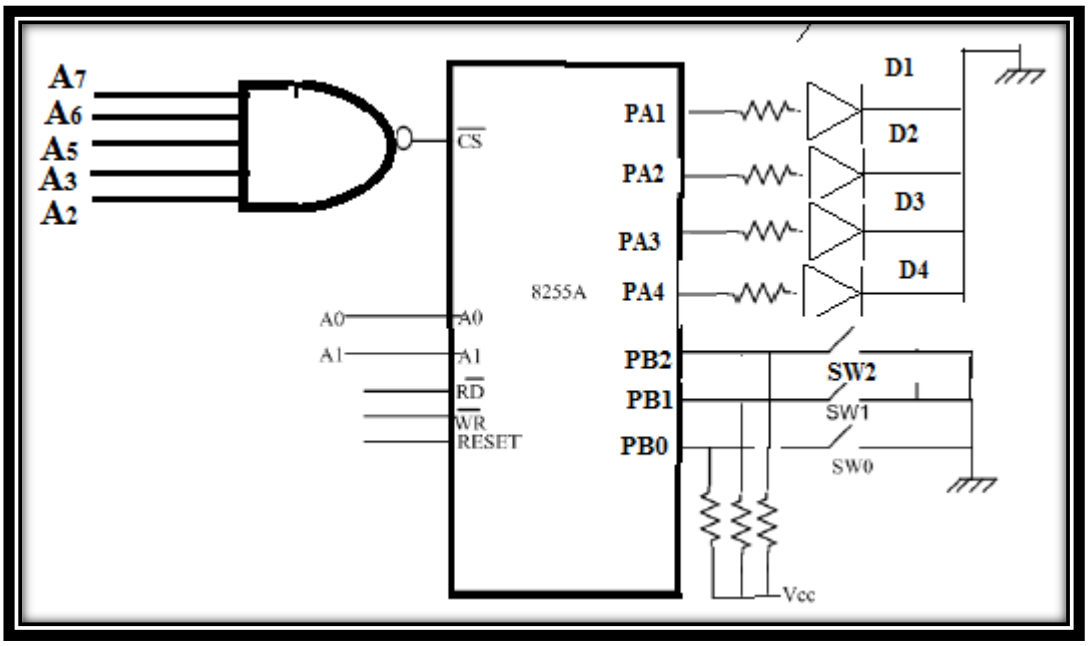
DELAY ENDP

Déterminé le délai de procédure DELAY

4) Ecrire un programme de fonctionnement de figure 1

Exercice 25 :

Soit la figure suivant qui représente l'interfaçage parallèle de µp 8086 avec circuit intégré 8255A a Fréquence d'horloge de 5MHZ



1) Détermine les adresses des ports A, B , C et CW (Control Word)

2) Détermine la valeur de CW mode registre en mode 00

Soit le fonctionnement de figure 1

SW2	SW0	L'état des leds(L)	Durée allumage
OFF	OFF	TOUT LES LEDS EN ETAT **OFF**	
OFF	ON	D2 **ON**	2Second
ON	OFF	D3 **ON**	3Second
ON	ON	D4 **ON**	4Second
D1 ON après activation désactivation D4			5Seconde
Si SW1 ON tous les LED ont état OFF			

3) Ecrire une procédure nommé **DELAY** en assembla ire qui signifié la durée d'allumage 1 seconde en prendre CX=50000

4) Avec l'exploitation de procédure **DELAY** écrire un programme assembla ire qui réalise l'état de tableau

Exercice 26 :

Dans un microcontrôleur PIC16F84A

a) quel est l'élément qui sert à indiquer l'adresse de la prochaine instruction à exécuter ?

b) où doit se situer le programme qui est exécuté à la mise sous tension?

c) quelle la durée d'un cycle (Tcy) pour exécuter une instruction si le PIC16F84A est piloté par un cristal de 4 MHz?

d) où doivent se situer les données utilisées par le programme?

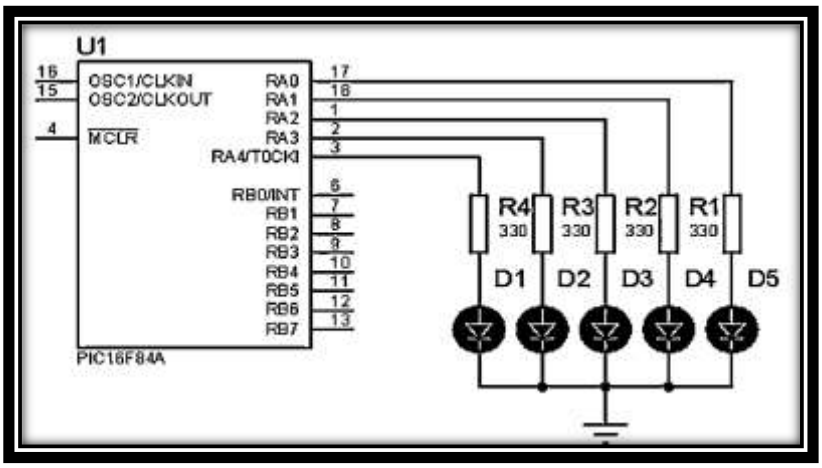
e) quelle est la longueur d'une instruction en bits?

f) quelle est la longueur d'une donnée dans la mémoire programme?

Exercice 27

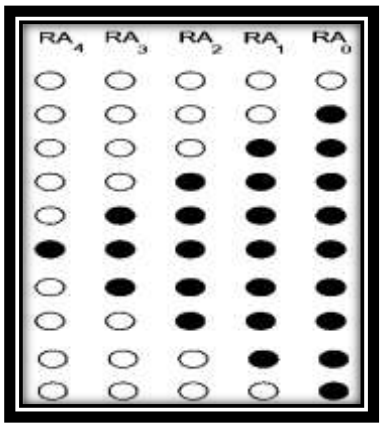
1) .Le PORTA du schéma suivant est configuré en sortie, que doit être le contenu du registre TRISA ?

2) On programme le microcontrôleur pour réaliser un clignotant. On remarque que toutes les leds clignotent sauf la led D1, qui n'est pas allumée. Expliquez le pourquoi et que faut-il faire pour que cette led (D1) fonctionne aussi ?



Exercice 28:

On désire réaliser à l'aide du PIC 16F84A un jeu de lumière qui permet de faire les séquences suivantes :



(Visualiser les états des sorties avec des LED)

«O : LED-Off :● LED-ON »

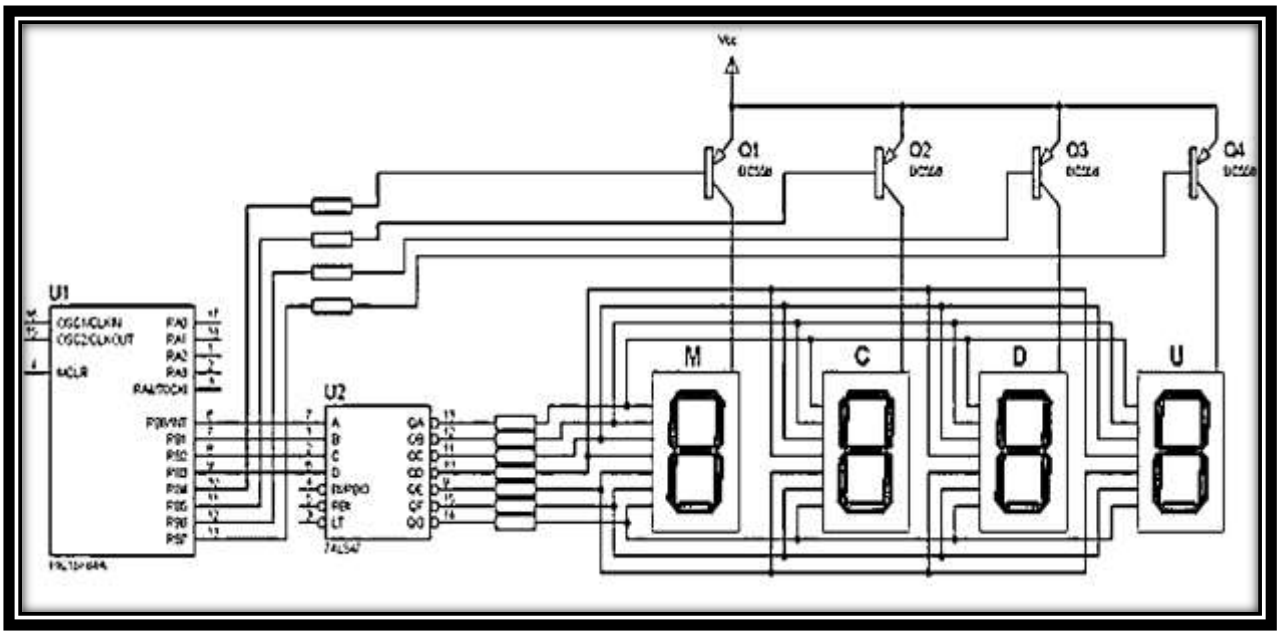
EXERCICES PROPOSER AVEC CES SOLUTIONS

1) Donner le schéma électronique
2) comment doit-être configuré le port A ?et quel sera le contenu du registre TRISA ?

3) donner la suite des données qu'il faut envoyer sur le port A pour réaliser le jeu de lumière désiré

Exercice 29 :

Soit le montage suivant :



1. Comment doit-être configuré le port B ?et quel sera le contenu du registre TRISB ?
2. Pour afficher le 6 sur l'afficheur U, que doit-on avoir sur le port B ?
3. Compléter le tableau suivant pour afficher le 5 sur l'afficheur D, le 3 sur l'afficheur C et le sur l'afficheur M

Exercice 30 :

Expliquer le fonctionnement des instructions **mov** suivantes :

1. movlw k

2. movwf f

3. movf f,d

a. movf X,w

b. movf X,f

Exercice 31:

Expliquer le fonctionnement des instructions arithmétiques suivantes :

1. addlw k

2. addwf f,d

a. addwf X,w

b. addwf X,f

3. sublw k

4. subwf f,d

a. subwf X,w

b. subwf X,f

5. incf f,d

6. decf f,d

Exercice 32 :

1) Expliquer le fonctionnement des instructions de remise à zéro suivante :

a. clrf f

b. clrw

2) Expliquer le fonctionnement des instructions de positionnement des bits d'un registre suivantes :

a. bsf f,b

b. bcf f,b

Exercice 33 :

Expliquer le fonctionnement des instructions de test sur bit suivantes

f (registre)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
b	7	6	5	4	3	2	1	0

1. BTFSC f,b

2. BTFSS f,b

3. DECFSZ f,d

4. **INCFSZ f,d**

Exercice 34 :

Expliquer le fonctionnement des instructions suivantes :

1. GOTO k (GOTO étiquette)

2. CALL k et RETURN

Exercice 35 :

Expliquer le fonctionnement des instructions suivantes :

1. RLF f,d

2. RRF f,d

3. SWAPF f,d

Exercice 36 :

Exécuter les instructions suivantes et donner les états successifs.

```
movlw    0x05
```

```
movwf   xdata
movlw   0x01
movwf   ydata
movf    xdata,w
incf    ydata,w
addlw   0x01
sublw   0x05
decf    xdata,f
```

Exercice 37 :

Exécuter les instructions suivantes et donner les états successifs.

```
movlw   0xD5
movwf   X
movlw   0x93
movwf   Y
movlw   0xF0
andwf   X,f
iorwf   Y,f
comf    Y,f
```

Exercice 38 :

Au départ (PCL) = 0x10, on exécute la partie du programme suivante :

```
Adresse (PCL)
        0x10  Movlw   0xFD
              xorlw   0xFF
              movwf   MEM1
              addlw   0x01
              movwf   MEM2
              andwf   MEM1,w
              movwf   MEM3
              decf    MEM3,f
              addwf   PCL,f
              movlw   0xFD
              iorlw   0xFF
              incf    MEM2
```

Après exécution, quels seront les contenus de MEM1, MEM2, MEM3, W et PCL ?

Exercice 39 :

Soit le registre STATUS :

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

1. Quel est le rôle du bit Z ?

2. Quel est le rôle du bit DC ?

3. Quel est le rôle du bit C ?

L'état du registre STATUS au départ est :

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

4) On exécute la partie du programme suivante :

```
movlw   0x18
movwf   STATUS
movlw   0xFF
addlw   0x01
```

Donner les états successifs du registre W et du registre STATUS

5. Quelle est la durée de cette partie du programme, si le PIC16F84A est piloté par un quartz de 8 MHz?

Exercice 40 :

Exécuter les instructions suivantes et donner les états successifs.

```
Etiquette   Instructions
            movlw   0x02
            movwf   xdata
            decfsz  xdata,f
            goto    saut
            movlw   0x0A
            movwf   xdata
saut        decfsz  xdata,f
            goto    saut
            decf    xdata,f
```

Exercice 41 :

Exécuter les instructions suivantes et donner les états successifs.

```
Etiquette   Instructions
            movlw   0xFF
            movwf   xdata
```

```
        incfsz      xdata,f
        goto        boucle
        movlw       0x01
        movwf       xdata
boucle  decfsz      xdata,w
        goto        boucle
        decf        xdata,f
```

Exercice 42:

On se place dans le cadre de l'outil de développement MPLAB de Microchip. En assembleur, quelle est la directive qui permet de :

a. définir (de choisir) le pic à utiliser.

b. définir les noms des variables réservées au pic.

c. configurer les modes de fonctionnement du pic. Donner un exemple pour le pic 16F84

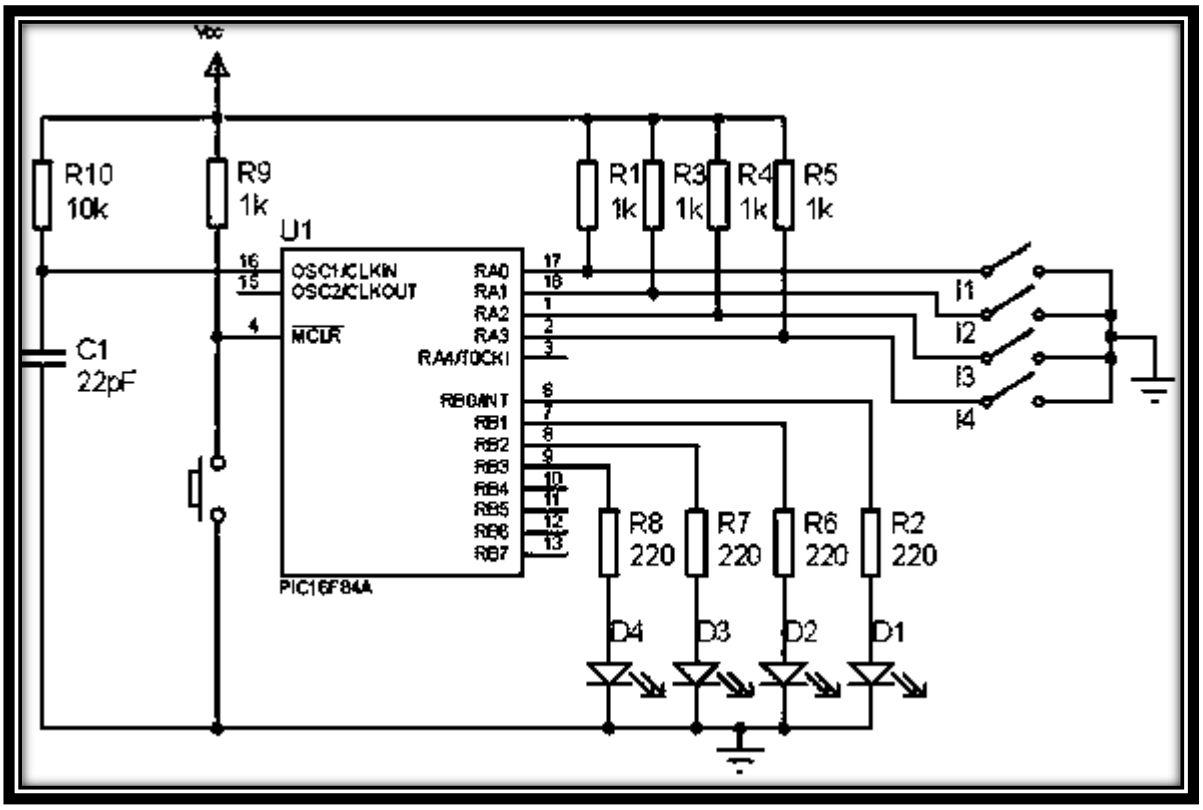
d. indiquer au pic l'adresse de début où les codes doivent être placés.

e. définir les noms des variables de l'utilisateur

f. indiquer la fin d'un programme

Exercice 43:

Soit le schéma suivant :



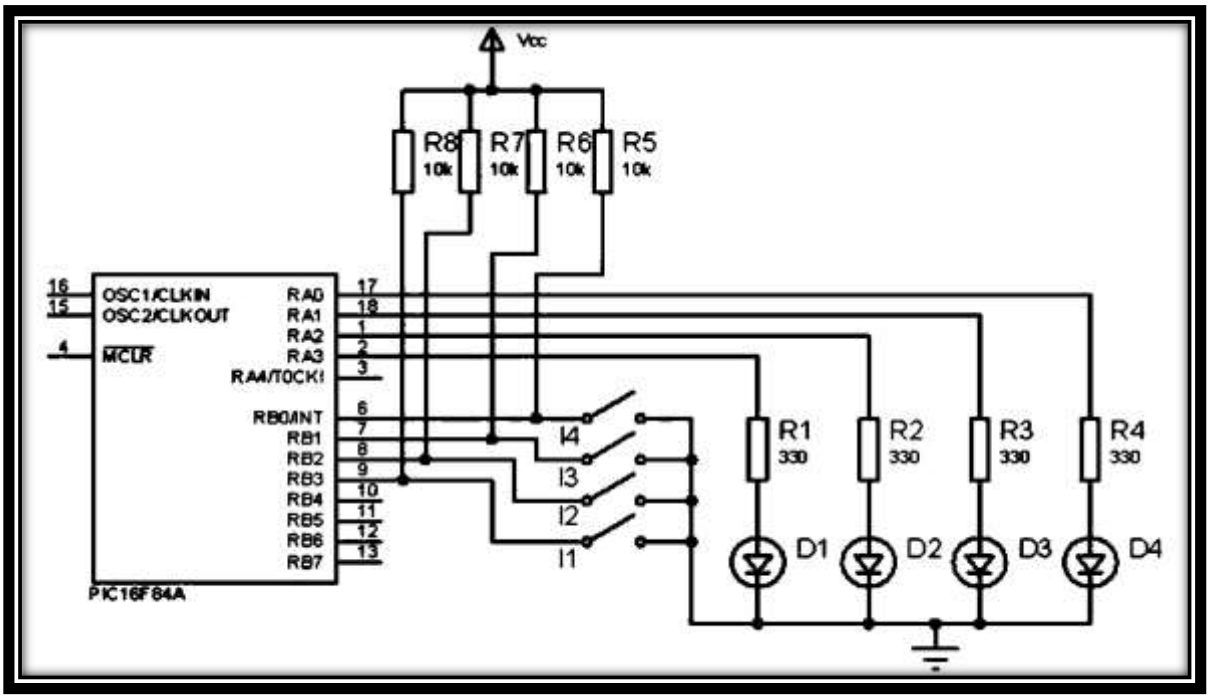
EXERCICES PROPOSER AVEC CES SOLUTIONS

On désire réaliser un programme qui permet d'afficher l'état des interrupteurs branchés au PORTA sur les leds branchées au PORTB (In → Dn avec n=1,2,3,4).

1. Quel est le pic utilisé et comment doit-on le déclarer ?
2. Comment doit-on définir les noms des variables réservées au pic?
3. Comment doit-on configurer les modes de fonctionnement du pic?
4. Où se trouvent les registres PORTA et PORTB et les registres TRISA et TRISB?
5. Comment doit être configurer le PORTA et le PORTB et que doit-on écrire dans TRISA et TRISB ?
6. Que faut-il faire pour accéder aux registres TRISA et TRISB ; dans le cas du 16F84A ? et que faut-il faire pour accéder aux registres PORTA et PORTB lorsqu'on est dans la banque1.
7. Ecrire les instructions qui permettent de configurer le PORTA et le PORTB.
8. Donner l'algorithme et l'organigramme du programme principal.
9. Ecrire le programme principal.
10. Ecrire le programme entier

Exercice 44

On désire réaliser le montage suivant pour lire et afficher l'état de chaque interrupteur (I1 sur D1, I2 sur D2, I3 sur D1 et I4 sur D4).



A. On donne l'algorithme suivant :

1) Choisir et configurer le PIC

2) Initialisation : configuration des ports

3) Lire l'état des interrupteurs sur le PORTB

4) Ecrire l'état des interrupteurs sur le PORTA

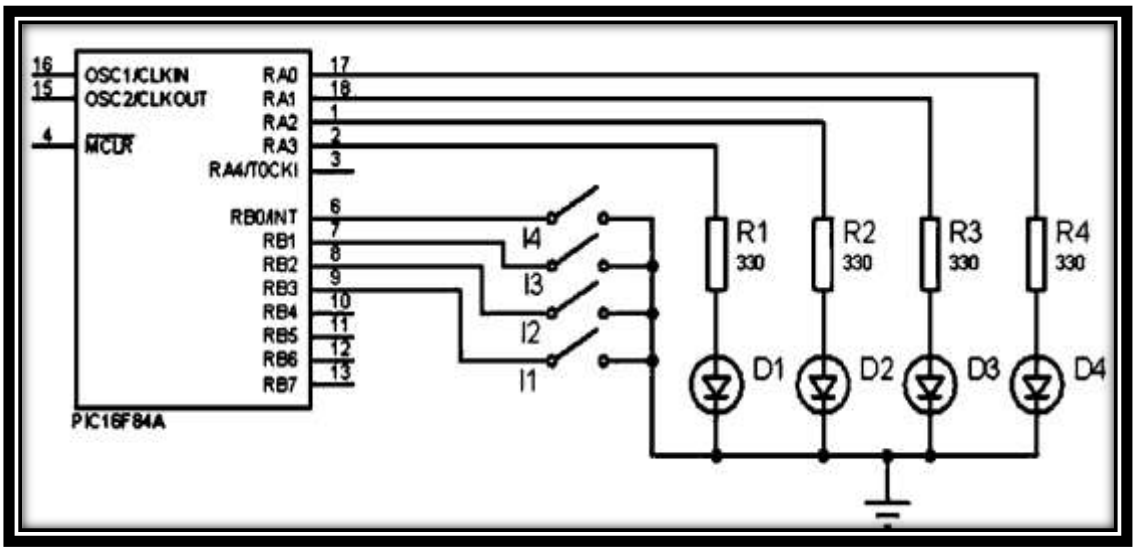
5) Aller à 3 (boucler)

6) Donner l'organigramme

7) Ecrire le programme assembleur

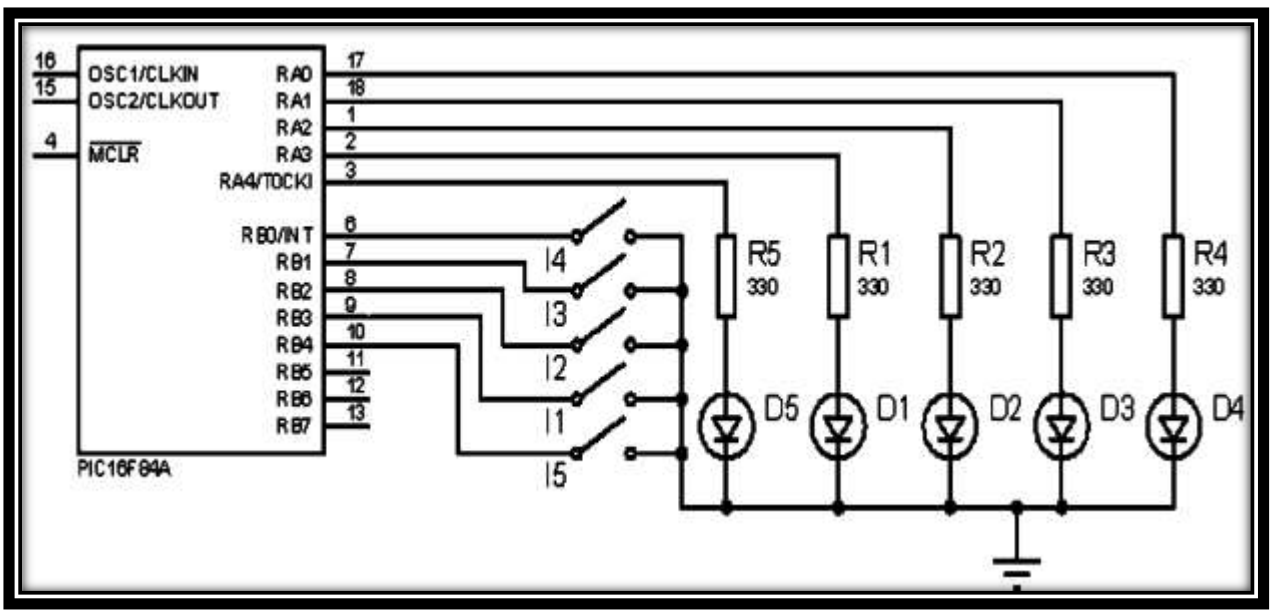
8) Ecrire le programme mikroC

B. On désire utiliser les résistances pull-up du PORTB, on modifie le schéma :



Modifier l'initialisation de PIC pour faire fonctionner le programme

C. On désire utiliser le montage pour contrôler l'état de 5 portes d'une voiture. On ajoute un interrupteur, une résistance et une led de la manière suivante :



On utilise les programmes de la question B, mais, la led D5 reste éteinte.

1) Modifier le schéma pour que le montage fonctionne.

2) Après modification du schéma, on remarque que la led D5 fonctionne inversement par

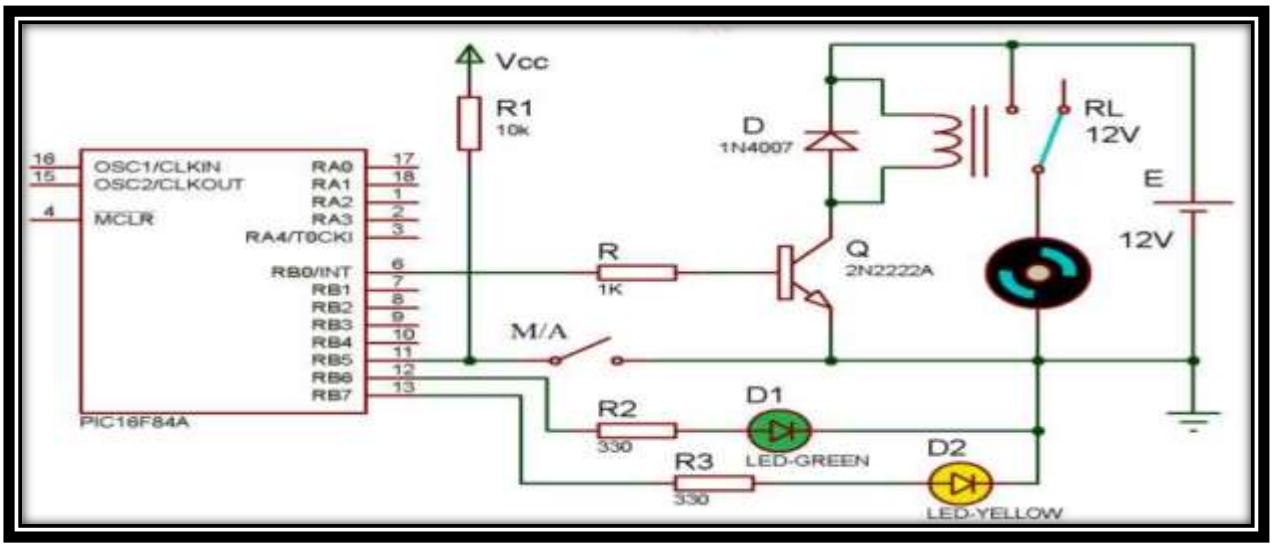
rapport aux autres leds. Expliquer le pourquoi.

3) Modifier le programme pour faire fonctionner toutes les leds de la même manière

On désire réaliser l'addition hexadécimale (4 bits) selon le schéma suivant

Exercice 45

Soit le montage suivant :



Le montage fonctionne de la manière suivante :

•Lorsque l'interrupteur M/A est ouvert (RB5 = 1) le moteur est arrêté (RB0 = 0), la led verte (RB6=0) est éteinte et la led jaune est allumée (RB7=1).

•Lorsque l'interrupteur M/A est fermé (RB5 = 0) le moteur tourne (RB0 = 1), la led verte est allumée (RB6 = 1) et la led jaune est éteinte (RB7 = 0).

On donne l'algorithme suivant :

1) Choisir et configurer le PIC

2) Initialisation : configuration des ports

3) Test : RB5 = 0 ? (interrupteur fermé ?)

4) Sinon → RB7 = 1, RB6 = 0 et RB0 = 0, aller à 3

5) Si oui → RB7 = 0, RB6 = 1 et RB0 = 1, aller à 3

1. Donner l'organigramme

2. Ecrire un programme en assembleur

Exercice 46 :

Soit un sous-programme (appelé : retard) dont l'algorithme est le suivant :

1) COUNT = 10

2) Décrémenter COUNT de 1 (COUNT = COUNT -1)

3) Si : COUNT = 0 ? Alors : retour de sous-programme Sinon : aller à 2

1. Donner l'organigramme de ce sous-programme

2. Ecrire le sous-programme en assembleur

3. Combien de cycles, dure ce sous-programme ? et quelle est la durée de ce sous-programme si le PIC utilisé est piloté par un quartz de 8 Mhz ?

4. Quelle est l'instruction qui permet d'appeler un sous-programme ?

Exercice 47 : Soit le sous-programme de temporisation suivant :

```
tempo movlw 0x05
      movwf cptr3
b3    clrf cptr2
b2    clrf cptr1
b1    decfsz cptr1, f
      goto b1
      decfsz cptr2, f
      goto b2
      decfsz cptr3, f
      goto b3
     return
```

1. Quelle est la durée de ce sous-programme si le PIC utilisé est piloté par un quartz de 10 Mhz ?

2. Quelle est la durée de ce sous-programme si le PIC utilisé est piloté par un quartz de 4 Mhz et cptr3 = 203 = 0xCB?

Exercice 48 :

Soit pic 16F84 A de Fréquence 16 MHz et 8 leds vert de led_0 a led_7 branché de RB_0 a RB_7

Ecrire un programme assembleur qui cliente les leds successivement dans 'un cycle de led_7 a led_0 duré 2 seconde pour chaque led

Exercice 49 :

Soit le Microcontrôleur 16 F84A

1) Signifie 16 : F: A:

Soit le début de programme MPLAB suivant :

```
LIST p=16f84A ;
INCLUDE "p16f84A.inc" ;
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF
cblock 000C;
ctr3:1;
ctr2:1;
ctr1:1;
```

endc;

org 00

goto init

init;

2) Signifier les Syntaxes suivants :

CP_OFF:

XT_OSC:

WDT_OFF:

Détermine le temps de chaque cycle

3) Soit le procédure suivant :

Delay

movlw N;

movwf ctr3;

t3 clrf ctr2;

t2 clrf ctr1;

t1 decfsz ctr1;

GOTO t1;

decfsz ctr2;

GOTO t2;

decfsz ctr3;

GOTO t3;

return

END

-Détermine N pour que la durée de ce procédure et a peut prés 1Seconde

4) Ecrire un programme d'initialisation suivant :

Port A0 et A3 en entré Port B0 jusqu'a B4 en sortie :

5) Ecrire un programme principale qui et réagir le fonctionnement suivant :

-Quand A0 et active les leds des ports B (B0…..B4) allume alternativement de LED0 a LEDD4 et de LED4 a LED0 avec une durée de 1Seconde pour chaque led

-Quand A3 et active tous les leds des ports B (B0…..B4) atteindre

-Faire une rappelle de procédure **Delay**

-Exploité les Instructions **RLF** et **RRF**

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 01

a) $(11)_2 = 21 + 20 = (3)_{10}$ $(1101)_2 = 23 + 22 + 20 = (13)_{10}$ $(100101110)2 = 28 + 25 + 23 + 22 + 21 = (302)_{10}$

b) Pour les 4 premiers nombres on va directement décomposer en puissance de 2 : $7 = 4+2+1 = 2^2 + 2^1 + 2^0 = (111)_2$ $51 = 32+16+2+1 = 2^5 + 2^4 + 2^1 + 2^0 = (11\ 0011)_2$ $128 = 2^7 = (1000\ 0000)_2$ $131 = 128+2+1 = 2^7 + 2^1 + 2^0 = (1000\ 0011)_2$ Pour 234 on va appliquer la méthode suivante des divisions successives : $234 = 2\times117+\mathbf{0}$ $117 = 2\times58+\mathbf{1}$ $58 = 2\times29+\mathbf{0}$ $29 = 2\times14+\mathbf{1}$ $14 = 2\times7+\mathbf{0}$ $7 = 2\times3+\mathbf{1}$ $3 = 2\times1+\mathbf{1}$ $1 = 2\times0+\mathbf{1}$ Ce sont les restes des divisions qui forment les bits du résultat. Le premier reste obtenu donne le bit de poids faible, le second donne celui de poids 1 et ainsi de suite. Donc 234 = 111010102.

c) Pour la conversion hexadécimale Binaire : $(12)_{16} = (0001\ 0010)_2$ $(DADA)_{16} = (1101\ 1010\ 1101\ 1010)_2$ $(5F3)_{16} = (0101\ 1111\ 0011)_2$ Pour la conversion hexadécimale décimal : $(12)^{16} = 1\times16^1 + 2\times16_0 = (18)_{10}$ $(DADA)_{16} = 13\times16^3 + 10\times16^2 + 13\times16^1 + 10\times16^0 = (56026)_{10}$ $(5F3)_{16} = 5\times16^2 + 15\times16_1 + 3\times16_0 = (1\ 523)_{10}$

d) Avec un octet (8 bits) on peut coder 2^8 valeurs : tous les entiers naturels de 0 (codé 0000 0000 en binaire) à $2^8 - 1 = 255$ (codé 1111 1111 en binaire).

Exercice 02 :

1) AX : registre accumula taire BX : registre de Base CX : registre de compteur

DX : registre de donnée CS : Code segment DS :Data segment

ES : Extra segment SS : Stack segment

2) Déterminer les registres de travail : AX, BX, CX,DX

les registres de segment : CS,DS,ES,SS

3) Registres de pointeurs et indexe: BP,SP,DI,SI

4) CF : Carry Flag

SF: Singe Flag

OF: Overflow Flag

ZF : Zero Flag

5) Registre: petit taille avec une grand vitesse .

Mémoire : petite taille et vitesse lent

Exercice03 : Soit les indices suivantes :

CS : 1234H DS : 5672H SS : 9AB5H

SP : 1234H IP : 5678H SI : 9ABCH

Adresse logique de data segment : DS:SI (5672:9ABC)

Adresse physique de code segment (CS0+IP)=12340+5678=179B8

minimal et maximal de CS :

Minimal (CS0+0000)=12340H

Maximal (CS0+FFFF)=2233FH

Exercice 04 :

1) MOV AL,0DF h DF→AL

AND AL,1101 1111b (DF AND AL)→AL

INC AL AL+1→AL

MOV AH,0CC h CC→AH

OR AH,20h (AH OR 20)→AH

DEC AH AH-1→AH

2) AL=1101 1111

AL=1101 1111

AL=E0 H

AH=110 01100

AH=110 11100

AH=DB H

AX=DBE0 H

Exercice 06 :

```
A dw 950  ;
B dw 900
MOV AX,A
MOV BX,B
SUB AX,BX
Jz eticcc
js etic
mov A,'B>'
jmp eticc
etic:mov A,'B<'
jmp eticc
eticcc:MOV A,'=B'
eticc:hlt
```

Exercice 7 :

1) a)Mettre sur la pile PUSH registre 16 bits

EXERCICES PROPOSER AVEC CES SOLUTIONS

Retirer de la pile POP registre 16 bits

b) Petite mémoire pour stocker temporairement les données

2) Code correspondant à une opération a effectuée

3) Composé d'un code opération et d'une opérande (eventuellement)

4) L'opération s'appelle l'assembleur.

5) Déclenché par un dispositif extérieur au processeur.

6) Une instruction est une opération élémentaire que le processeur peut remplir peut accomplir.

Le code opérande définissant les paramètres de l'action. Le code opérande dépend de l'opération.

Il peut s'agir d'une donnée ou bien d'une adresse mémoire.

7) Les bus: bus d'adresses bus de commande bus de contrôle.

Exercice 08:

```
MOV aX,009
MOV CX,AX
FAC dw 0
FA  dw 0
DEC CX
alm:add aX,cx
mov FAC,AX
MOV FA,DX
loop alm
hlt
```

Exercice 09:

```
MOV aX,009
MOV CX,AX
FAC dw 0
FA  dw 0
DEC CX
alm:mul CX
mov FAC,AX
MOV FA,DX
loop alm
hlt
```

Exercice 10 :

SEGMENT	LOWER	HYPER
ES	**40000+0000**	**40000+FFFF**
SS	**1FFF0+0000**	**1FFF0+FFFF**
DS	**20000+0000**	**20000+FFFF**

3) POP: Sp+2 PUSH :Sp-2

	060H	080H	050H	0A0H	30H
ADD logique	**2000:2FC3**	**1FFF: 0DA41**	**2000:06C71**	**1FFF: 04A41**	**1FFF:01F6**

Exercice 11 :

1)

```
MOV AX,0C30DH
AND AX,0300H
OR AX,0012H
OR AX,0C31DH
AND AX,0C30DH
XOR AX,0C01FH
XOR AX,0C01F
```

2)

```
MOV AX,8F1FH
MOV dx,0FH
HH: ROL AX,01
JnC  KK
MOV CX,AX
ROL CX,01
JnC  KK
inc Bx
kk:DEC DX
CMP DX,00H
Ja HH
hlt
```

3)

```
MOV AX,8F1FH
MOV dx,0FH
HH: ROL AX,01
JnC  KK
MOV CX,AX
ROL CX,01
JC  KK
```

inc Bx

kk:DEC DX

CMP DX,00H

Ja HH

hlt

Exercice 12 :

1) Adressage register 4) Adressage immédiat 7) Adressage basé indexé

2) Adressage direct 5) Adressage indirect par registre 8) Adressage basé indexé

3) Adressage basé 6) Adressage indexé

Exercice 13 :

Cmp ax,7

JA ET9

CMP AX,3

JAE ETN3

MOV [3036],AX

JMP ENDD

ETN3 : MOV [3034],AX

JMP ENDD

ETN3 CMP,9

JA ETT9

MOV [3032],AX

JMP ENDD

ETT9: MOV [3030],AX

ENDD :end;

Exercice 14 :

1) Adressage register 4) Adressage immédiat

2) Adressage direct 5) Adressage indirect par registre

3) Adressage basé 6) Adressage indexé

Exercice 15 :

1) MOV AX,03

MOV BX,01

ADD AX,BX

```
Cmp ax,5
JL ET3
CMP AX,8
JA ET88
MOV [2035], AX
JMP ENDD
ET88 : MOV [2030],AX
JMP ENDD
ET3: CMP ax, 3
JA ETT3
MOV [2039], AX
JMP ENDD
ETT3: MOV [2037], AX
ENDD:HLT ;
```

2)

```
MOV CX,0
MOV AX,0
gh: CMP CX,10
JAe ENDSOMME
INC CX
add ax,cx
jmp  gh
ENDSOMME:Mov bx,ax
MOV CX,0
MOV AX,1
ghh: CMP CX,4
JAe ENDMULTI
INC CX
MUL CX
jmp  ghh
ENDMULTI:ADD Ax,Bx
Endp
```

Exercice 16 :

1)

```
MOV AX,3
cmp ax,1
JE GOF
CMP ax,2
JE GOFF
CMP AX,3
JE GOFFF
JMP END
GOF:MOV CX,0
gh: CMP CX,10
JAe END
INC CX
add ax,cx
jmp  gh
GOFF:MOV CX,0
MOV AX,1
ghh: CMP CX,4
JAe END
INC CX
MUL cx
jmp  ghh
GOFFF:MOV CX,0
MOV AX,1
ghO: CMP CX,10
JAe ENDSOMME
INC CX
MUL Cx
jmp  ghO
ENDSOMME:Mov bx,ax
MOV CX,0
MOV AX,0
ghhO: CMP CX,4
JAe ENDMULTI
```

```
INC CX
ADD Ax,CX
jmp  ghhO
ENDMULTI:MUL Bx
endp
END: HLT
```

2) $F = \sum_{i=1}^{4} \prod_{N=i}^{4} N$

F=21H

Exercice 17:

```
mov bx,0800H
 MOV ds,bx
 MOV si,0400H
 mov bx,05h
 call som
 mov bx,06h
 call som
 mov bx,09h
 call som
 hlt
 som proc
    mov ax,00H
    L:ADD ax,bx
    dec bx
    cmp bx,00H
    JA L
    inc si
    inc si
    MOV [SI],ax
    ret
som endp
```



Exercice 18 :

```
a  dw 0
MOV DX,0100H
MOV SP,DX        pour évité camflit
mov Cx,00H
JK:INC Cx
CMP CX,05H
JA FIN
mov bx,Cx
call multA
ADD a,AX
JMP JK
FIN:HLT
multA proc
MOV AX,01H
KL:  MUL BX
INC BX
CMP BX,05H
JLE KL
ret 4
```

multA endp

Exercice 19 :

a) INITI PROC NEAR

MOV BX,00H

MOV SI,00H

MOV DI,00H

MOV AL,00H

MOV CX,14H

RET 4

INITI ENDP

b) RET4:Pour retour au programme principale

C) + D)

DATA SEGMENT

TAB1 DB

[37H,01H,00H,07h,00H,39H,12h,00H,12H,77H,20H,01H,31H,35h,32H,39H,36h,30H,39H,34H]

TAB2 DB 20 (0)

DATA ENDS

STACK SEGMENT

DB 128 DUP(0)

Ends

Code segment

Mov ax,data

Mov ds,ax

Mov ax,stack

Mov ss,ax

Call init

OK: MOV dL,TAB1[SI]

MOV AH,02H

INT 21H

INC SI

LOOP OK

Call init

OO:mov DL,tab1[SI]

```
cmp DL,30H
JLE  ETI
INC BL
MOV tab2[di],DL
MOV AH,02H
INT 21H
INC di
ETI:INC SI
LOOP OO
HLT
INITI  PROC NEAR
MOV BX,00H
MOV SI,00H
MOV DI,00H
MOV AL,00H
MOV CX,14H
RET  4
INITI  ENDP
```

E)

```
DATA SEGMENT
TAB1 DB [37H,01H,02H,07h,02H,39H,10h,04H,12H,77H,20H,
01H,31H,35h,32H,39H,36h,30H,39H,34H]
 TAB2 DB 20 (0)
DATA ENDS
 som db 0
 STACK SEGMENT
DB 128  DUP(0)
Ends
Code segment
Mov ax,data
Mov ds,ax
Mov ax,stack
Mov ss,ax
Call init
OO : mov DL,tab1[SI]
```

```
cmp DL,20H
JAE  ETI
ADD BH,DL
ETI:INC SI
LOOP OO
MOV SOM,BH
HLT
INITI  PROC NEAR
MOV BX,00H
MOV SI,00H
MOV DI,00H
MOV AL,00H
MOV CX,14H
RET  4
INITI  ENDP
```

F)

```
DATA SEGMENT
TAB1
DB[37H,02H,02H,07h,02H,39H,11h,04H,12H,77H,20H,01H,31H,35h,32H,39H,36h,30H,39H,34H]
 TAB2 DB 20 (0)
DATA ENDS
 PROD dW 0
STACK SEGMENT
DB 128  DUP(0)
Ends
Code segment
Mov ax,data
Mov ds,ax
Mov ax,stack
Mov ss,ax
CALL INITI
INC AX
O:mov BL,tab1[SI]
cmp BL,13H
JAE  ETII
```

```
MUL BL
ETII:INC SI
LOOP O
MOV PROD,AX
MOV CX,1000H
MOV DS,CX
MOV CX,0300H
MOV Si,CX
MOV [SI],AX
HLT
INITI PROC NEAR
MOV BX,00H
MOV DI,00H
MOV SI,00H
MOV AL,00H
MOV CX,14H
RET  4
INITI ENDP
```

Exercice 20

1) Quelle et la déférence entre mémoire et registre

Registre : capacité minimale a une grande vitesse

Mémoire: capacité d'important à une vitesse lent

2) Quelle le nombre de boitier de mémoire de 8KO supporté parµP8086:

Nb=1024/8=128 BOITS

3) Quelle le nombre de bus d'adresse de chaque boitier par µP 8086

NOMB=Ln(8X1024)/Ln(2)=13ligne

4) Quelle le nombre de bus d'adresse pour manipulé les quatre boitiers par µP 8086

NOMB=Ln(6X8X1024)/Ln(2)=15.50= 16ligne

5) Quelles et les adresses de débit et la fin de troisième boitier en saie que l'adresse de débit de première boité 80000H

A19																		A0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**Début B1**
1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	**Fin B1**

80000H	**Début B1 (H)**
83FFEH	**Fin B1(H)**

A19																		A0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**Début B2**
1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**Fin B2**
80001H																				**Début B2 (H)**
83FFFH																				**Fin B2(H)**
A19																		A0	
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**Début B3**
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	**Fin B3**
84000H																				**Début B3 (H)**
87FFEH																				**Fin B3(H)**
A19																		A0	
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**Début B4**
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**Fin B4**
84001H																				**Début B4 (H)**
87FFFH																				**Fin B4(H)**
A19																		A0	
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**Début B5**
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	**Fin B5**
88000H																				**Début B5 (H)**
8BFFEH																				**Fin B5(H)**
A19																		A0	
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**Début B6**
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**Fin B6**
88001H																				**Début B6 (H)**
8BFFFH																				**Fin B6(H)**

Exercice 21:

1)NOMBRE MAXIMALE=1024/32=32BOITE

2) Nombre de ligne pour chaque boite =ln(64000)/ln(2)=16ligne pour chaque boitier

Nombre de ligne pour chaque boite =ln(64000*2)/ln(2)=17 ligne pour deux boitiers

EXERCICES PROPOSER AVEC CES SOLUTIONS

A)

	A19			A16															A0	Boitiers	
00000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Déb	1
1FFFE	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	Fin	
00001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Déb	2
1FFFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	fin	

b) en supose C0000H a la place E0000H

Nombre de ligne pour chaque boite =ln(128000*2)/ln(2)=18 ligne pour deux boitiers

	A19		A17																A0	Boitiers	
C0000	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Déb	1
FFFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	Fin	
C0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Déb	2
FFFFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	fin	

Exercice 22 :

1	0	1	0	0	0	1	0

CW:10100010 (A2H)

PORT	ADRESSE
PORTA	80H
PORTB	81H
PORTC	82H
CW	83H

Configuration en programmation assembleurs

PORTA EQU 0080H

PORTB EQU 0081H

PORTB EQU 0082H

CW EQU 0083H

MOV AL, A2H

OUT CW, AL

Exercice 23:

Tableau des addresses

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	PORT
1	1	1	1	1	0	0	0	**PORTA**
1	1	1	1	1	0	0	1	**PORTB**
1	1	1	1	1	0	1	0	**PORTC**
1	1	1	1	1	0	1	1	**CW**

PORTA: F8H

PORTB: F9H

PORTC: FAH

CW: FBH

1	0	0	0	0	0	1	0

CW:10000010 (82H)

Programme

```
PORTA EQU 00F8H
PORTB EQU 00F9H
CW EQU 00FBH
MOV AL, 82H
OUT CW, AL; Control word sent to CR
MAIN : IN AL, PORTB;
AND AL, 00000011B;
JPE STOP; (JPE=Jump si parity , p=1) le nombre de 1 et pair
TEST AL, 00000001B;
JZ FORWARD ; si SW0 0, resultat devienn zero (Z=1)
MOV AL, 00000001B;
OUT PORTA, AL; moteur torn en  direction inverse
JMP MAIN;
STOP : MOV AL, 00000000B;
OUT PORTA, AL;
JMP MAIN;
FORWARD: MOV AL, 00000010
OUT PORTA, AL
JMP MAIN
```

Autre méthode :en utilise l'instruction CMP

```
PORTA EQU 00F8H
PORTB EQU 00F9H
CW EQU 00FBH
MOV AL, 82H
OUT CW, AL; Control word sent to CR
MAIN : IN AL, PORTB;
CMP AL, 00000011B;
JE STOP;
```

```
CMP AL, 00000000B;
JE STOP
CMP AL,00000001B
JE FORWARD
MOV AL, 00000001B;
OUT PORTA, AL;
JMP MAIN;
STOP : MOV AL, 00000000B;
OUT PORTA, AL;
JMP MAIN;
FORWARD: MOV AL, 00000010B
OUT PORTA, AL
JMP MAIN
```

Exercice 24:

1)

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	**PORT**
1	1	1	1	1	0	0	0	**PORTA**
1	1	1	0	1	0	1	0	**PORTB**
1	1	1	0	1	0	0	1	**PORTC**
1	1	1	0	1	0	1	1	**CW**

Port A:F8H

Port B:FAH

Port C:F9H

CW :FBH

Remarque d'après la figure de l'exercice 24 en remarque que A_0 de µp branché au A_1 de CI8255 et A1 de µp branché au A_0 de CI8255 alors en inverse les adresse entre PORTB et PORTC

2) Détermine la valeur de CW mode registre en mode 00:

1	0	0	0	0	0	1	0

CW:82H

3) $64H=4x16^0+6x16^1=100$

$T= (50000/5x\ 10^6) \times 100=1seconde$

4)

```
PORTA EQU 00F8H
PORTB EQU 00FAH
PORTC EQU 00F9H
```

```
CW EQU 00FBH
MOV AL, 82H
OUT CW, AL; Control word sent to CR
MAIN : IN AL, PORTB;
ROR AL,03
JNC STOP
ROL AL,03
Cmp al,00000100b
JE CAL1S
Cmp al,00000101b
JE CAL2S
Cmp al,00000110b
JE CAL3S
MOV AL,00011110b
OUT  PORTA,AL
Call DELAY
Call DELAY
Call DELAY
Call DELAY
Jmp main
CAL3S : MOV AL,00011110b
OUT  PORTA,AL
Call DELAY
Call DELAY
Call DELAY
Jmp main
CAL2S : MOV AL,00011110b
OUT  PORTA,AL
Call DELAY
Call DELAY
Jmp main
CAL1S : MOV AL,00011110b
OUT  PORTA,AL
Call DELAY
Jmp main
```

STOP: MOV AL,00000000b

OUT PORTA,AL

Jmp main

END

DELAY PROC NEAR

MOV BX, 64H

PULSE: MOV CX, 50000

HERE: LOOP HERE

DEC BX

JNZ PULSE

RET 4

DELAY ENDP

Exercice 25 :

1)

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	**PORT**
1	1	1	0	1	1	0	0	**PORTA**
1	1	1	0	1	1	0	1	**PORTB**
1	1	1	0	1	1	1	0	**PORTC**
1	1	1	0	1	1	1	1	**CW**

PORTA: 00ECH

PORTB: 00EDH

PORTC: 00EEH

CW: 00EFH

2)

1	0	0	0	0	0	1	0

CW: 082H

3) **DELAY** PROC NEAR

MOV BX, 64H

PULSE: MOV CX, 50000

HERE: LOOP HERE

DEC BX

JNZ PULSE

RET 4

DELAY ENDP

Remarque utilise les instructions suivant:

Cmp : comparaison, JE ou JNE: (saut si égal ou non égal)

Jmp: Saut inconditionnelle

IN: entrée, OUT: sortie

4)

```
PORTA EQU 00ECH
PORTB EQU 00EDH
PORTC EQU 00EEH
CONTR EQU 00EFH
MOV AL, 82H
OUT CONTR, AL; Control word sent to CR
MAIN : IN AL, PORTB;
Cmp al,00000101b
JE LED4
Cmp al,00000100b
JE LED3
Cmp al,00000001b
JE LED2
And  al,00000010b
Cmp al,00000010b
JE TOUJOURS OFF
MOV AL,00000000b
OUT  PORTA,AL
Jmp main
TOUJOURS OFF:MOV AL, 00000000b
OUT  PORTA,AL
Jmp main
LED4: MOV AL,00010000b
OUT  PORTA,AL
Call delay
Call delay
Call delay
Call delay
MOV AL,00000010b
OUT  PORTA,AL
Call delay
```

```
Call delay
Call delay
Call delay
Call delay
Jmp main
LED3: MOV AL,00001000b
OUT  PORTA,AL
Call delay
Call delay
Call delay
Jmp main
LED2: MOV AL,00000100b
OUT  PORTA,AL
Call delay
Call delay
Jmp main
END
DELAY PROC NEAR
MOV BX, 64H
PULSE: MOV CX, 50000
HERE: LOOP HERE
DEC BX
JNZ PULSE
RET  4
DELAY ENDP
```

Exercice 26 :

a) C'est le compteur programme qui sert à indiquer l'adresse de la prochaine instruction à exécuter.

b) À la mise sous tension, le programme qui est exécuté doit être dans la mémoire programme (FLASH ROM).

c) La durée d'un cycle (Tcy) pour exécuter une instruction est telle que : Tcy = Tosc x 4 Fcy = Fosc/4 avec Tosc la période de l'horloge du microcontrôleur PIC. Fosc = 4MHz , Fcy = 4MHz / 4 = 1MHz , Tcy = 1 / Fcy = 1µs

d**)** Les données utilisées par le programme doivent être dans la mémoire de données (RAM)

e) La longueur d'une instruction est de 14 bits.

I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0

f) La longueur d'une donnée dans la mémoire programme est de 8bits

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Donnée codée sur 8 bits

Exercice 27 :

1) Le port A désigné par PORTA est un port de 5 bits (RA0 à RA4). La configuration de direction pour chaque bit du port est déterminée avec le registre de direction : TRISA.

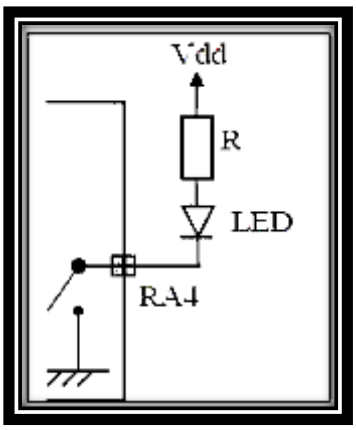
- Bit n de TRISA = 0 (Output) → bit n de PORTA configuré en sortie
- Bit n de TRISA = 1 (Input) → bit n de PORTA configuré en entrée

Pour configurer le PORTA en sortie, il faut mettre des zéros (0) dans TRISA

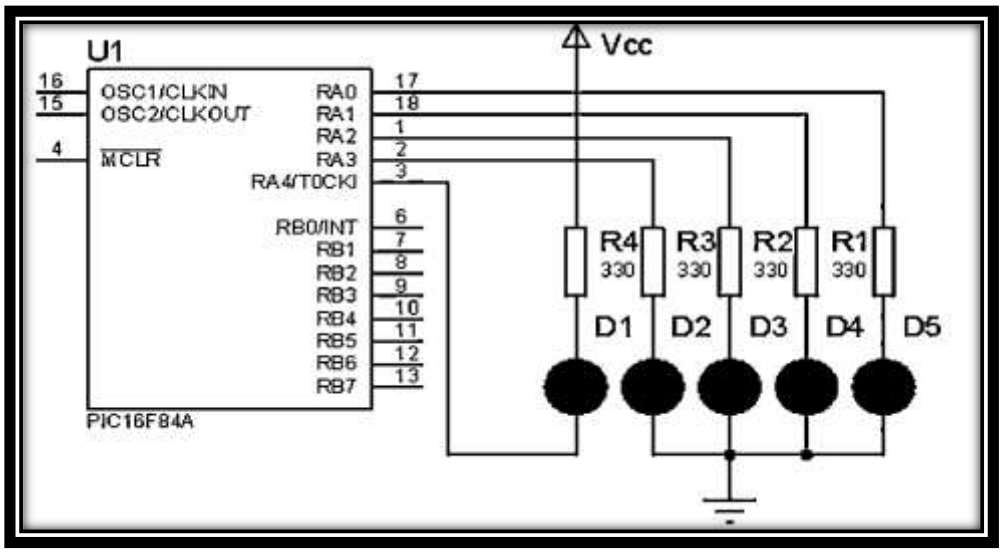
TRISA	X	X	X	0	0	0	0	0

2) RA4 est une E/S à drain ouvert, si on veut l'utiliser comme sortie il ne faut pas oublier de mettre une résistance de rappel vers Vcc.

Exemple : allumer une LED

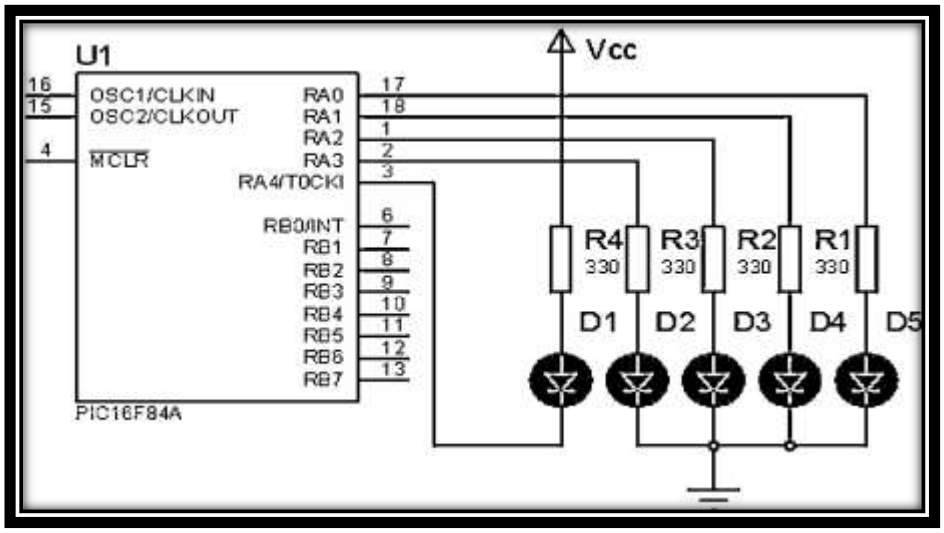


Donc, il faut modifier le schéma pour obtenir le schéma suivant :



Exercice 28 :

1) Voir : exercice 26 question 2



2) Voir : exercice26 question 1

TRISA	X	X	X	0	0	0	0	0

3)

RA4	RA3	RA2	RA1	RA0	Données en hexa
0	0	0	0	0	00x00
0	0	0	0	1	00x01
0	0	0	1	1	00x03
0	0	1	1	1	00x07
0	1	1	1	1	00x0F
1	1	1	1	1	00x1F
0	1	1	1	1	00x0F
0	0	1	1	1	00x07
0	0	0	1	1	00x03
0	0	0	0	1	00x01

Exercice 29:

1) Le microcontrôleur est branché au décodeur 7segment 74LS47 par RB0, RB1, RB2 et RB3 qui amènent le code de la donnée à afficher sur les afficheurs. Pour sélectionner, le microcontrôleur doit commander l'un des transistors de type PNP, par RB4 (afficheur M), RB5 (afficheur C), RB6 (afficheur D) et RB7 (afficheur U). Le PORTB doit être configuré en sortie, et :

TRISB	0	0	0	0	0	0	0	0

2) pour afficher le 6 sur l'afficheur U, on doit avoir sur le port B :

PORTB	0	1	1	0	0	1	1	0
	Q4 seul saturé				Code du 6			

3) Pour afficher 1356, on doit mettre :

4)

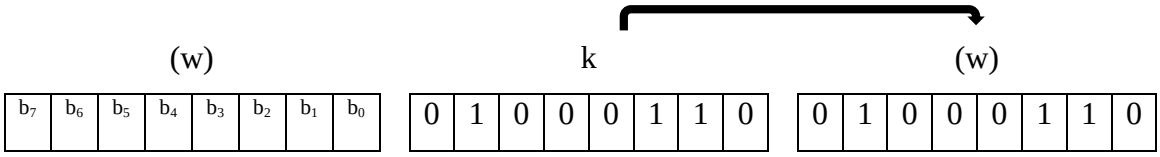
Chiffre	Afficheur	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	Transistor
6	U	0	1	1	1	0	1	1	0	Q4
5	D	1	0	1	1	0	1	0	1	Q3
3	C	1	1	0	1	0	0	1	1	Q2
1	M	1	1	1	0	0	0	0	1	Q1
		Choix de l'afficheur (transistor)				Code du chiffre à afficher				

Exercice 30 :

1. movlw k

Avant execution: k = 0x46 = B'01000110' et (w) = b7b6b5 b4 b3b2b1b0

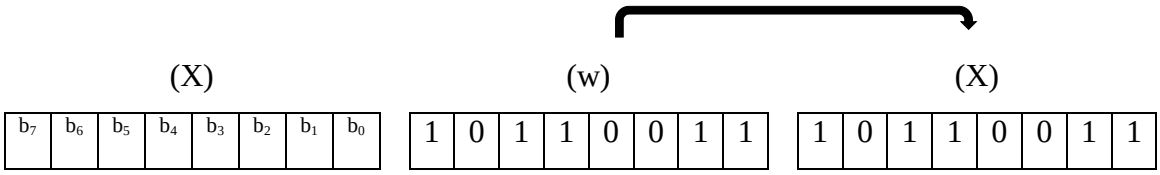
movlw 0x46



2. movwf f

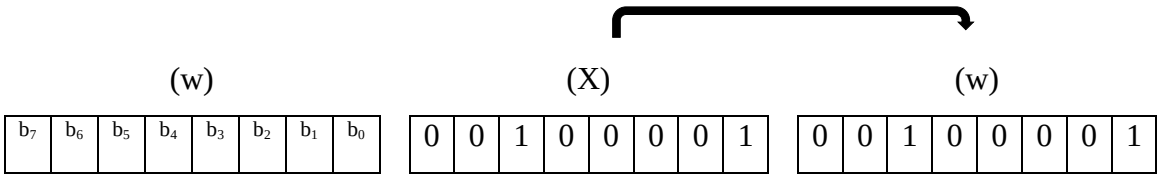
Avant execution: (w) = 0xB3 = B'10110011 et (X) = b7b6b5 b4 b3b2b1b0

movwf X

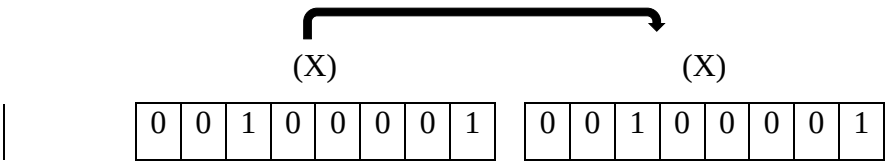


3. movf f,d

a. movf X,w Avant exécution : (X) = 0x21 =B'00100001'



b,movf X,f Avant exécution : (X) = 0x21 =B'00100001



Exercice 31 :

1. addlw k k + (w) →□(w)

Avant exécution : (w) = 0x67 = B'01100111' et k = 0x09 = B'00001001'

addlw 0x09

```
  0 0 0 0 1 0 0 1  ← 0x09
+ 0 1 1 0 0 1 1 1  ← (w)
-----------------
  0 1 1 1 0 0 0 0  → (w) = 0x09 + 0x67 = 0x70
```

Après exécution : **(w) = 0x70**

2.addwf f,d

a. addwf X,w (X) + (w) →□(w)

Avant exécution : (X) = 0x36 et (w) = 0x5

addwf X,w

```
  0 0 1 1 0 1 1 0  ← (X)
+ 0 1 0 1 1 1 0 0  ← (w)
-----------------
  1 0 0 1 0 0 1 0  → (w) = 0x36 + 0x5C = 0x92
```

Après exécution : (X) = 0x36 et (w) = 0x92

b.addwf X,f (X) + (w) →□(X)

Avant exécution : (X) = 0x59 et (w) = 0xA2

addwf X,f

```
  0 1 0 1 1 0 0 1  ← (X)
+ 1 0 1 0 0 0 1 0  ← (w)
-----------------
  1 1 1 1 1 0 1 1  → (X) = 0x59 + 0xA2 = 0xFB
```

Après exécution : (X) = 0xFB et (w) = 0xA2

3.sublw k k – (w) →(w)

Avant exécution : (w) = 0x06 = B'00000110'

sublw 0x09

```
  0 0 0 0 1 0 0 1  ← 0x09
- 0 0 0 0 0 1 1 0  ← (w)
-----------------
  0 0 0 0 0 0 1 1  → (w) = 0x09 - 0x06 = 0x03
```

Après exécution : (w) = 0x03

4.subwf f,d

a. subwf X,w (X) – (w) → (w)

Avant exécution : (X) = 0x56 et (w) = 0x3C

subwf X,w

```
  0 1 0 1 0 1 1 0  ← (X)
- 0 0 1 1 1 1 0 0  ← (w)
-----------------
  0 0 0 1 1 0 1 0  → (w) = 0x56 - 0x3C = 0x1A
```

Après exécution : (X) = 0x56 et (w) = 0x1A

b subwf X,f (X) – (w) → (X)

Avant exécution : (X) = 0xA9 et (w) = 0x52

subwf X,f

```
  1 0 1 0 1 0 0 1  ← (X)
- 0 1 0 1 0 0 1 0  ← (w)
-----------------
  0 1 0 1 0 1 1 1  → (X) = 0xA9 - 0x52 = 0x57
```

Après exécution : (X) = 0x57 et (w) = 0x52

5.incf f,d

a. incf X,w (X) + 1 → (w)

Avant exécution : (X) = 0x09 et (w) = b7b6b5 b4 b3b2b1b0

incf X,w

```
  0 0 0 0 1 0 0 1  ← (X)
+ 0 0 0 0 0 0 0 1  ← 1
-----------------
  0 0 0 0 1 0 1 0  → (w) = 0x09 + 0x01 = 0x0A
```

Après exécution : (X) = 0x09 et (w) = 0x0A

b.incf X,f (X) + 1 → (X)

Avant exécution : (X) = 0x09

incf X,f

```
  0 0 0 0 1 0 0 1  ← (X)
+ 0 0 0 0 0 0 0 1  ← 1
-----------------
  0 0 0 0 1 0 1 0  → (X) = 0x09 + 0x01 = 0x0A
```

Après exécution : (X) = 0x0A

6. decf f,d

a. decf X,w (X) - 1 → (w)

Avant exécution : (X) = 0x09 et (w) = b7b6b5 b4 b3b2b1b0

decf X,w

```
0 0 0 0 1 0 0 1  ← (X)
0 0 0 0 0 0 0 1  ← 1
-----------------
0 0 0 0 1 0 0 0  → (w) = 0x09 - 0x01 = 0x08
```

Après exécution : (X) = 0x09 et (w) = 0x08

b. decf (X) - 1 → (X)

Avant exécution : (X) = 0x09

decf X,f

```
0 0 0 0 1 0 0 1  ← (X)
0 0 0 0 0 0 0 1  ← 1
-----------------
0 0 0 0 1 0 0 0  → (w) = 0x09 - 0x01 = 0x08
```

Après exécution : (X) = 0x08

Exercice 32 :

1.

a. clrf 0 → (X)

Avant exécution : (X) = b7b6b5 b4 b3b2b1b0

clrf X

Après exécution : (X) = 0x00

b. clrw 0 → (w)

Avant exécution : (w) = b7b6b5 b4 b3b2b1b0 clrw

Après exécution : (w) = 0x00

2

f (registre)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
b	7	6	5	4	3	2	1	0

a. bsf f,b b : bit

Avant exécution : (X) = b7b6b5 b4 b3b2b1b0

bsf X,4 Met le bit 4 de X à 1

Après exécution : (X) = b7b6b5 1 b3b2b1b0

b. bcf f,b

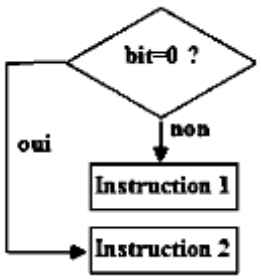
Avant exécution : (X) = b7b6b5 b4 b3b2b1b0

bcf X,2 Met le bit 2 de X à 0

Après exécution : (X) = b7b6b5 b4 b30b1b0

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 33 :



BTFSC f,b : Teste le bit b de f.

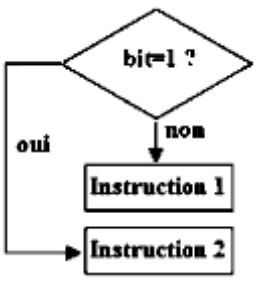
Si ce bit vaut 0 alors l'instruction suivante est ignorée.

f : registre b : numéro de bit (de 0 à 7)

BTFSC registre, bit saut si bit = 0 (bit Clear)

Instruction 1 (si bit = 1 exécute instruction 1)

Instruction 2 (si bit = 0 exécute instruction 2)



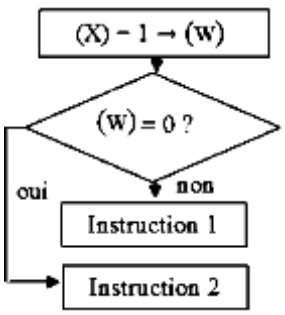
.BTFSS f,b : Teste le bit b de f. Si ce bit vaut 1 alors l'instruction suivante es

f : registre b : bit

BTFSS registre, bit saut si bit = 1 (bit Set)

2Instruction 1 (si bit = 0 exécute instruction 1)

Instruction 2 (si bit = 1 exécute instruction 2)



3DECFSZ f,d : 2 opérations sont exécutées.

1. 1ere opération : (f) - 1→ (destination).

a. Si d = 0 (destination) = (w).

b. Si d = 1 (destination) = (f).

2. 2eme opération : test : (f) – 1= 0 ?

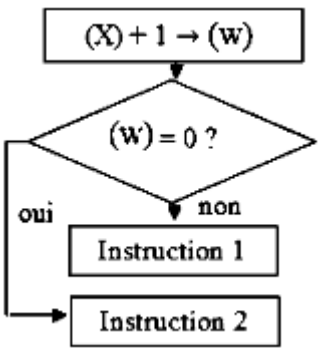
a. Si (f) - 1 = 0, alors l'instruction suivante est ignorée.

b. Si (f) - 1 ≠ 0, alors l'instruction suivante est exécutée.

DECFSZ X,f : saut si (X) = 0

Instruction 1 (instruction 1 exécutée si (X) ≠ 0)

Instruction 2 (instruction 2 exécutée si (X) = 0)



INCFSZ f,d : 2 opérations sont exécutées.

1ere opération : (f) + 1→ (destination).

a. Si d = 0 (destination) = (w).

b. Si d = 1 (destination) = (f).

2. 2eme opération : test : (f) + 1= 0 ?

a. Si (f) + 1 = 0, alors l'instruction suivante est ignorée.

b. Si (f) + 1 ≠ 0, alors l'instruction suivante est exécutée.

INCFSZ X,w : saut si (w) = 0

Instruction 1 (instruction 1 exécutée si (w) ≠ 0)

Instruction 2 (instruction 2 exécutée si (w) = 0)

Exercice 34 :

1. GOTO k (GOTO étiquette) : Branchement inconditionnel à l'adresse k.

Etiquette	Instructions
	GOTO toto
	……..
	……..
	……..
toto	MOVLW 02
	MOVWF PORTB

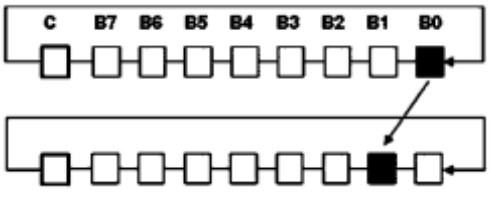
CALL k : Appel à un sous-programme.

RETURN : Fin d'un sous-programme (associée à call).

Etiquette	Instructions
	CALL TP
	……..
	……..
	……..
TP	MOVLW 02
	MOVWF PORTB
	……..
	……..
	RETURN

Exercice 35 :

1. RLF f,d : Rotation logique à gauche (décalage circulaire à gauche) de la variable stockée à l'adresse f. Résultat dans f si d = 1 ou dans w si d = 0.



EXERCICES PROPOSER AVEC CES SOLUTIONS

a. Avant exécution : (X) = 0x21 =B'00100001' et (w) = $b_7b_6b_5$ b_4 $b_3b_2b_1b_0$

rlf X,w

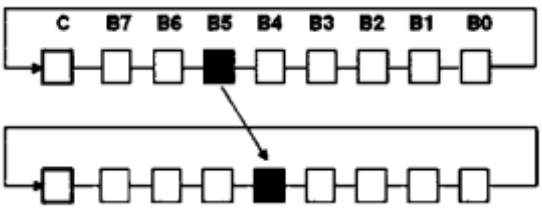
	X									W							
	0	0	1	0	0	0	0	1	→	0	1	0	0	0	0	1	0

b. Avant exécution : (X) = 0x21 =B'00100001

rlf X,f

	X									X							
	0	0	1	0	0	0	0	1	→	0	1	0	0	0	0	1	0

2. RRF f,d : Rotation logique à droite (décalage circulaire à droite) de la variable stockée à l'adresse f. Résultat dans f si d = 1 ou dans w si d = 0.



a. Avant exécution : (X) = 0x21 =B'00100001' et (w) = $b_7b_6b_5$ b_4 $b_3b_2b_1b_0$

rrf X,w

	X									W							
	0	0	1	0	0	0	0	1	→	0	0	0	1	0	0	0	0

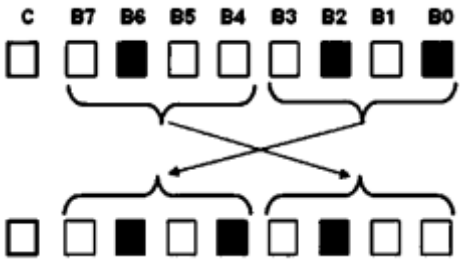
b. Avant exécution : (X) = 0x21 =B'00100001

rrf X,f

	X									X							
	0	0	1	0	0	0	0	1	→	0	0	0	1	0	0	0	0

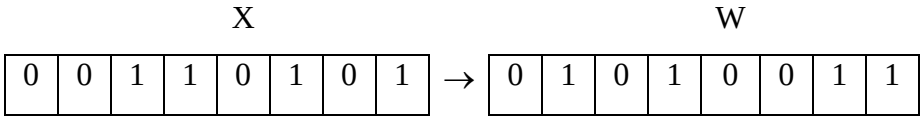
3.SWAPF f,d : Permute les deux quartets de f.

Résultat dans f si d = 1 ou dans w si d = 0.



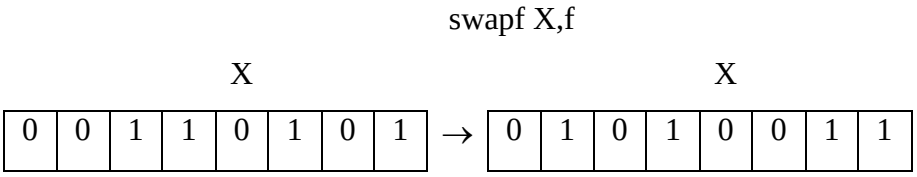
a. Avant exécution : (X) = 0x35 =B'00110101' et (w) = b7b6b5 b4 b3b2b1b0

swapf X,w



b.Avant exécution : (X) = 0x35 =B'00110101'

swapf X,f



Exercice 36 :

Instructions	Description	Etats des registres après exécution
movlw 0x05	0x05 → (w)	(w) = 0x05
movwf xdata	(w) → (xdata)	(xdata) = 0x05
movlw 0x01	0x01 → (w)	(w) = 0x01
movwf ydata	(w) → (ydata)	(ydata) = 0x01
movf xdata,w	(xdata) → (w)	(w) = 0x05
incf ydata,w	(ydata) +1 → (w)	(w) = 0x01 + 1 = 0x02
addlw 0x01	(w) + 0x01 → (w)	(w) = 0x02 + 0x01 = 0x03
sublw 0x05	0x05 – (w) → (w)	(w) = 0x05 - 0x03 = 0x02
decf xdata,f	(xdata) -1 → (xdata)	(xdata) = 0x05 - 0x01 = 0x04

Exercice 37:

Instructions	Description	Etats des registres après exécution
movlw 0xD5	0xD5 → (w)	(w) = 0xD5
movwf X	(w) → (X)	(X) = 0x05
movlw 0x93	0x93 → (w)	(w) = 0x93
movwf Y	(w) → (Y)	(Y) = 0x93
movlw 0xF0	(0xF0) → (w)	(w) = 0xF0
andwf X,f	(X).(w) → (X)	(X) = 0xD5 AND 0xF0 0xD5 = 1101 0101 0xF0 = 1111 0000 (X) = 1101 0000 = 0xD0

iorwf Y,f	(Y) OR (w) → (Y)	(Y) = 0x93 OR 0xF0 0x93 = 1001 0011 0xF0 = 1111 0000 (Y) = 1111 0011 = 0xF3
comf Y,f	NOT(Y) → (Y)	(Y) = NOT(0xF3) (Y) = NON (1111 0011) (Y)= 0000 1100 = 0x0C

Exercice 38 :

PCL	Instructions	Description	Etats des registres apprès exécution
0x10	movlw 0xFD	0xFD→ (w)	(w)=0xFD
0x11	xorlw 0xFF	0xFF⊕(w)→(w)	(w)=0xFF⊕0xFD (w)=1111 1111⊕1111 1101 (w)=0000 001=0x02
0x11	xorlw 0xFF	0xFF⊕(w)→(w)	(w)=0xFF⊕0xFD (w)=1111 1111⊕1111 1101 (w)=0000 001=0x02
0x12	movwf MEM1	(w)→ (MEM1)	(MEM1)=0x02
0x13	addlw 0x01	0x01+(w)→ (w)	(w)= 0x01+0x02=0x03
0x14	movwf MEM2	(w)→ (MEM2)	(MEM2)=0x03
0x15	Andwf MEM1,w	(MEM1).(w)→ (w)	(w)= 0x01.0x02=0x02
0x16	movwf MEM3	(w)→ (MEM3)	(MEM3)=0x02
0x17	decf MEM3,f	(MEM3)-1→(MEM3)	(MEM3)= 0x02-0x01=0x01
0x18	addwf PCL,f	(PCL)+(w)→ (PCL)	(PCL)=0x18 + 0x01 + 0x02 = 0x1B PCL pointe l'adresse de la prochaine instruction : 0x18 + 0x01
0x19	movlw 0xFD	}	Instructions non exécutées (Instructions sautées)
0x1A	iorlw 0xFF		
0x1B	incf MEM2,f	(MEM2)+1→(MEM2)	(MEM2)= 0x03+0x01=0x04
0x1C	Instruction suivante		(PCL)=0x1C

MEM1 = 0x02, MEM2 = 0x04, MEM3 = 0x01 , (w) = 0x02 et (PCL) = 0x1C

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 39 :

Soit le registre STATUS

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

1. Lorsque le microcontrôleur traite une instruction (arithmétique, logique ou remise à zéro : CLR), le bit Z indique si le résultat est nul (Z=1) ou non (Z=0).

2. Lorsque le microcontrôleur traite une instruction arithmétique le bit DC indique si on a une retenue sur les 4 bits de poids faible.

3. Lorsque le microcontrôleur traite une instruction arithmétique le bit C indique si on a une retenue sur 8 bits.

L'état du registre STATUS au départ est :

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

1. On exécute la partie du programme suivante

		w								STATUS							
		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
movlw	0x18	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
movlf	STATUS	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
movlw	0xFF	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0	0
addlw	0x01	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

2. Le PIC16F84A est piloté par un quartz de 8 MHz ; donc la fréquence de fonctionnement est : 8 MHz /4 = 2 MHz et la durée d'un cycle est : 1 / 2 MHz = 0,5 µs.

		Durée en cycle
movlw	0x18	1
movlf	STATUS	1
movlw	0xFF	1
addlw	0x01	1
	Total	4

La durée de cette partie du programme est : 4 x 0,5 µs = 2 µs

Exercice 40 :

Etiquette	Instructions	Description	Etat des registres	
	movlw	0x02	0x02 → (w)	(w) = 0x02
	movwf	xdata	(w) → (xdata)	(xdata) = 0x02
	decfsz	xdata,f	(xdata) – 1 = (xdata)	(xdata) = 0x01
	goto	saut	Aller à saut, si (xdata) ≠ 0	(xdata) = 0x01

	movlw	0x0A	Instruction non exécutée	(w) = 0x02
	movwf	xdata	Instruction non exécutée	(xdata) = 0x01
saut	decfsz	xdata,f	(xdata) – 1 = (xdata)	(xdata) = 0
	goto	saut	Instruction non exécutée	(xdata) = 0
	decf	xdata,f	(xdata) – 1 = (xdata)	(xdata) = 0xFF

Exercice 41 :

Etiquette	Instructions	Description	Etat des registres	
	movlw	0xFF	0xFF → (w)	(w) = 0xFF
	movwf	xdata	(w) → (xdata)	(xdata) = 0xFF
	incfsz	xdata,f	(xdata) + 1 → (xdata)	(xdata) = 0
	goto	boucle	Instruction non exécutée	(xdata) = 0
	movlw	0x01	0x01 → (w)	(w) = 0x01
	movwf	xdata	(w) → (xdata)	(xdata) = 0x01
boucle	decfsz	xdata,w	(xdata) – 1 → (w)	(w) = 0
	goto	boucle	Instruction non exécutée	(w) = 0
	decf	xdata,f	(xdata) – 1 → (xdata)	(xdata) = 0

Exercice 42 :

a. La directive qui permet de définir (de choisir) le pic à utiliser est : **list**
Exemples : list p=16F84A list p=16F877 list p=16F628
b. La directive qui permet de définir les noms des variables réservées au pic, est :

#include

Exemples : #include <p16f84A.inc> # include <p16f877.inc>
Les variables réservées au pic sont les noms des registres tels que PORTA, TRISA, PORTB, TRISB, STATUS …etc, et les noms des bits tels que C, DC, Z, RP0 …etc.
c. La directive qui permet de configurer les modes de fonctionnement du pic est :

config

Exemple de configuration :
__config _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

Pour tableau configuration voir le tableau de la page **80**

d. ORG : détermine où le compilateur doit mettre les codes qui suivent cette commande.

Ex: ORG 0x000.

e. CBLOCK : détermine un bloc mémoire pour réserver des octets pour différentes variables.

ENDC : Fin de la déclaration pour le bloc mémoire.

Exemple :

CBLOCK 0x20 ; Déclaration d'un bloc mémoire à l'adresse 0x20

TEMP : 1 ; Réservation d'un octet à l'adresse 0x20

TABLE : 8 ; Réservation de 8 octets à partir de l'adresse 0x21

ENDC ; Fin de la déclaration du bloc mémoire

f. END : directive fin de programme

Exercice 43 :

1. Le pic utilisé est le 16F84A, et on doit le déclarer par :

list p=16F84A

2. On définit les noms des variables réservées au pic par :

#include<p16f84A.inc>

3. Pour configurer les modes de fonctionnement du pic, on doit répondre à des questions telles que :

Doit-on utiliser un Code Protection ? NON : _CP_OFF

A la mise sous tension du µC, doit-on lancer une temporisation d'environ 72 ms durant laquelle est effectué un RESET interne ? OUI : _PWRTE_ON

Est-ce qu'on a besoin d'utiliser le Watchdog ? NON : _WDT_OFF

Quel est le type d'horloge utilisé? Horloge RC : _RC_OSC

On obtient, alors:

__config _CP_OFF & __PWRTE_ON & WDT_OFF & _RC_OSC

4. Les registres PORTA et PORTB se trouvent dans la banque 0 (bank 0) de la mémoire de données, et les registres TRISA et TRISB se trouvent dans la banque 1 (bank 1) de la mémoire de données.

5. Les interrupteurs sont des entrées et les leds sont des sorties, donc RA0, RA1, RA2 et RA3 doivent être configurés en entrée et RB0, RB1, RB2 et RB3 doivent être configurés en sortie.

Pour configurer RA0, RA1, RA2 et RA3 en entrée, il faut mettre des uns (1) dans TRISA

TRISA	x	x	x	x	1	1	1	1

Pour configurer RB0, RB1, RB2 et RB3 en sortie, il faut mettre des zéros (0) dans TRISB

TRISB	x	x	x	x	0	0	0	0

Puisque les autres lignes du PORTB et du PORTA ne sont pas utilisés on va prendre :

TRISA	1	1	1	1	1	1	1	1	TRISB	0	0	0	x	0	0	0	0

6. Pour accéder aux registres TRISA et TRISB ; dans le cas du 16F84A, il faut mettre le bit RP0 du registre STATUS à 1

bsf STATUS ,RP0 ; mettre 1 dans RP0 : sélectionner bank 1

et pour accéder aux registres PORTA et PORTB lorsqu'on est dans la banque1, il faut mettre le bit RP0 à 0.

bcf STATUS ,RP0 ; mettre 0 dans RP0 : sélectionner bank 0

7. Instructions de configuration des PORTA et PORTB

```
bsf     STATUS ,RP0     ; sélectionner bank 1
movlw   0xff            ; configure les lignes du
movwf   TRISA           ; PORTA en entrée
movlw   0x00            ; configure les lignes du
movwf   TRISB           ; PORTB en sortie
bcf     STATUS          ,RP0 ; sélectionner bank 0
```

8. Algorithme et organigramme du programme principal

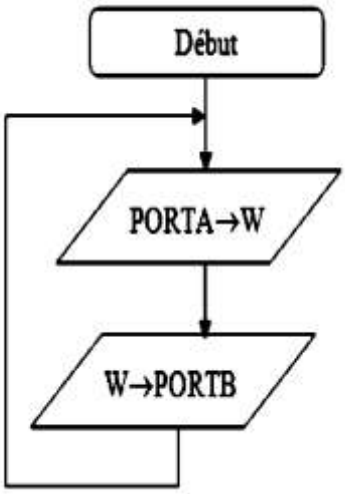
Algorithme

1) lire le PORTA : mettre PORTA dans W

2) transférer W sur le PORTB

3) recommencer (aller à 1)

Organigramme



9. Programme principal

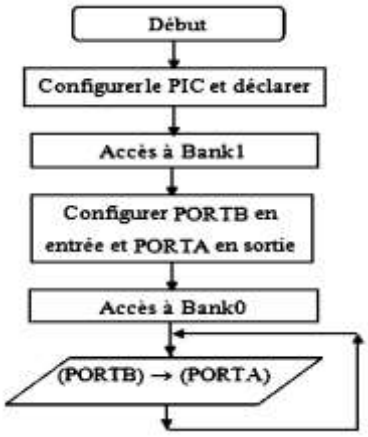
```
boucle  movf;   PORTA,W   met PORTA dans W
        movwf;  PORTB     met W dans PORTB
        goto    boucle
```

10. Programme entier en assembleur

```
; ----------  Directives assembleur pour le choix et la configuration du PIC ----------

            list  p=16F84A

            #include <p16F84A.inc>

            __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

; ------------------------------ Démarrage sur RESET ---------------------------------

        org     0x00                    ; adresse du début du programme

        goto    initialisation
```

Exercice 44:

1. Organigramme



1. Programme assembleur

```
     LIST       p=16F84A                ; Définition de processeur
     #include <p16F84A.inc>             ; Définitions de variables

     __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
; ------------------------------ Démarrage sur reset ---------------------------------
     org 0x000              ; Adresse de départ après reset
     goto    init           ; Adresse 0: initialiser
; ------------------------------ Initialisation --------------------------------------
init
     bsf     STATUS,RP0     ; sélectionner banque 1
     clrf    TRISA          ; PORTA en sortie (banque1)
     movlw   0xFF           ;
     movwf   TRISB          ; PORTB en entrée(banque1)
     bcf     STATUS,RP0     ; repasser banque 0
; ------------------------------ Programme principal ---------------------------------
debut
     movf    PORTB,w        ; (PORTB) --> (w)
     movwf   PORTA          ; (w) --> (PORTA)
     goto    debut          ; boucler à l'infini
     end
```

```
; ---------------------------- Initialisation des PORTS ------------------------------

initialisation

        bsf         STATUS , RP0    ; sélectionner bank 1

        movlw       0xFF            ; configure les lignes du PORTA

        movwf       TRISA           ; PORTA en entrée

        movlw       0x00            ; configure les lignes du PORTB

        movwf       TRISB           ; PORTB en sortie

        bcf         STATUS , RP0    ; sélectionner bank 0

; ------------------------------ Programme principal ---------------------------------

boucle

        movf        PORTA , W       ; met PORTA dans W

        movwf       PORTB           ; met W dans PORTB

        goto        boucle

        end
```

```
void main() {
    TRISB = 0xFF;          //PORTB en entrée
    TRISA = 0;             //PORTA en sortie

    while(1){
    PORTA = PORTB;         // (PORTB) --> (PORTA)
    }
}
```

B.

Pour activer les résistances PULL-UP du PORTB, il faut mettre à 0 le bit $\overline{\text{RBPU}}$ (bit 7) du registre OPTION

```
    LIST      p=16F84A              ; Définition de processeur
    #include <p16F84A.inc>          ; Définitions de variables

    __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

;-------------------------------- Démarrage sur reset ----------------------------------------------
    org 0x000             ; Adresse de départ après reset
    goto    init          ; Adresse 0: initialiser

;-------------------------------- Initialisation ---------------------------------------------------
init
    bsf     STATUS,RP0    ; sélectionner banque 1
    clrf    TRISA         ; PORTA en sortie (banque1)
    movlw   0xFF          ;
    movwf   TRISB         ; PORTB en entrée(banque1)
    bcf     OPTION_REG,7; activer les résistances PULL-UP du PORTB
    bcf     STATUS,RP0    ; repasser banque 0

;-------------------------------- Programme principal ----------------------------------------------
debut
    movf    PORTB,w       ; (PORTB) --> (w)
    movwf   PORTA         ; (w) --> (PORTA)
    goto    debut         ; boucler à l'infini
    end
```

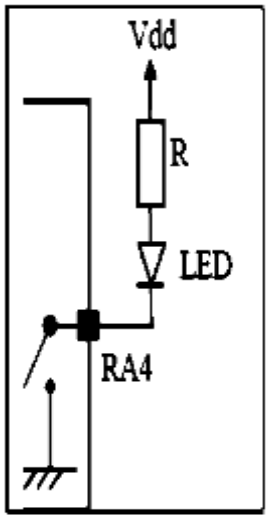
Programme mikroC :

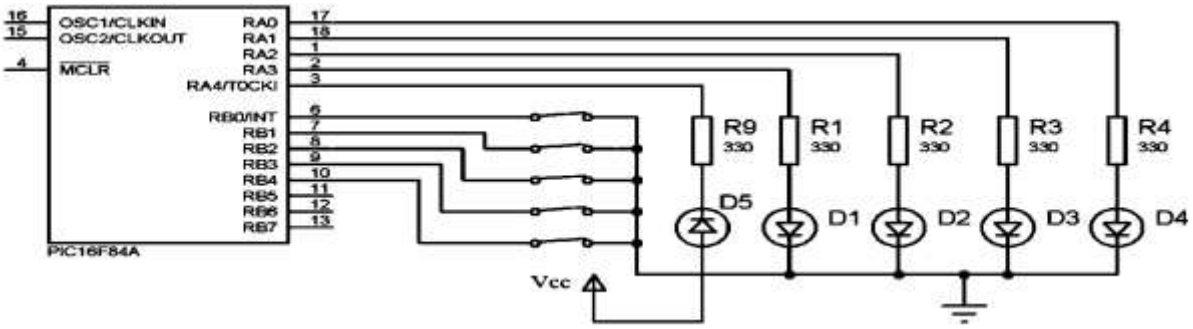
```
void main() {
    TRISB = 0xFF;          //PORTB en entrée
    TRISA = 0;             //PORTA en sortie
    OPTION_REG.B7 = 0;     // activer les résistances PULL-UP du PORTB

    while(1){
    PORTA = PORTB;         // (PORTB) --> (PORTA)
    }
}
```

C.

1. RA4 est une E/S à drain ouvert. Pour l'utiliser en sortie, on doit la relier à l'alimentation via une résistance.

Exemple : allumer une LED





2. la led D5 s'allume lorsque RA4 = 0 et s'éteint lorsque RA4 = 1; par contre les autres leds s'allument lorsque RA3RA2RA1RA0 = 1111 et s'éteignent lorsque RA3RA2RA1RA0 = 0000.

3.

Rappel : $0 \oplus 0 = 0$ et $0 \oplus 1 = 1 \Rightarrow 0 \oplus x = x$

$1 \oplus 0 = 1$ et $1 \oplus 1 = 0 \Rightarrow 1 \oplus x = \bar{x}$

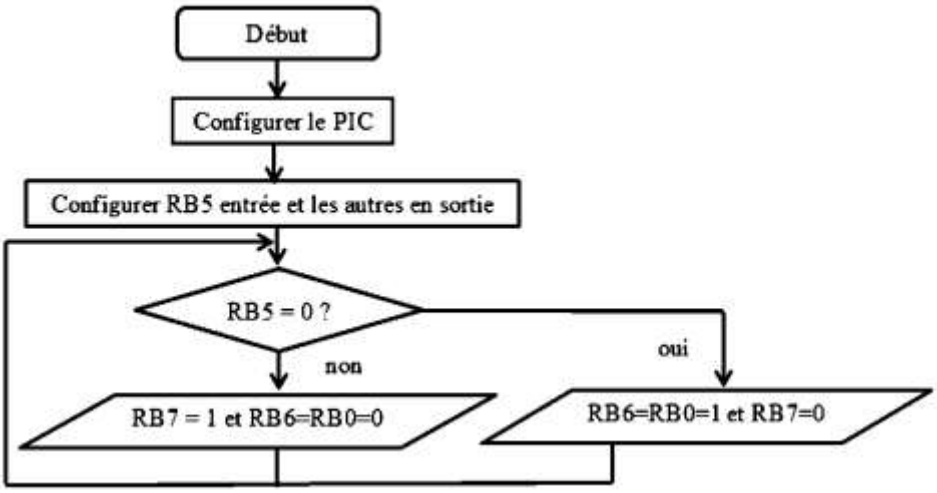
$b_7b_6b_5b_4b_3b_2b_1b_0 \oplus 0001000 = b_7b_6b_5\bar{b}_4b_3b_2b_1b_0$

Pour faire fonctionner la led D5 comme les autres leds, il suffit d'inverser le bit 4 (b4) du PORTB avant de l'envoyer sur le PORTA.

```
    LIST      p=16F84A              ; Définition de processeur
    #include <p16F84A.inc>          ; Définitions de variables

    __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

;-------------------------------- Démarrage sur reset ----------------------------------------------

    org 0x000             ; Adresse de départ après reset
    goto    init          ; Adresse 0: initialiser

;-------------------------------- Initialisation ---------------------------------------------------
init
    bsf     STATUS,RP0    ; sélectionner banque 1
    clrf    TRISA         ; PORTA en sortie (banque1)
    movlw   0xFF          ;
    movwf   TRISB         ; PORTB en entrée(banque1)
    bcf     OPTION_REG,7; activer les résistances PULL-UP du PORTB
    bcf     STATUS,RP0    ; repasser banque 0

;-------------------------------- Programme principal ----------------------------------------------
debut
    movf    PORTB,w       ; (PORTB) --> (w)= b7b6b5b4b3b2b1b0
    xorlw   0x10          ; (w)= b7b6b5b4b3b2b1b0 xor 00010000
    movwf   PORTA         ; (w) --> (PORTA)= b7b6b5b4b3b2b1b0
    goto    debut         ; boucler à l'infini
    end
```

Exercice 45 :

1. Organigramme



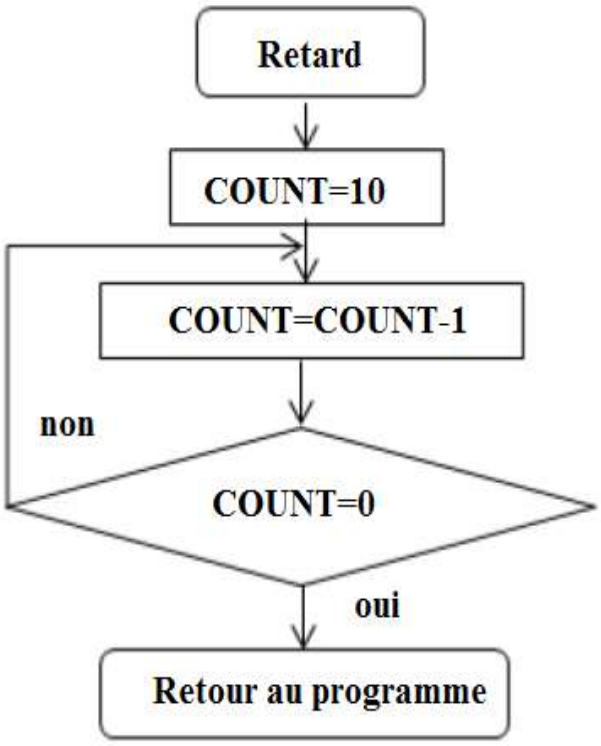
2. Programme en assembleur

```
    LIST      p=16F84A              ; Définition de processeur
    #include <p16F84A.inc>         ; Définitions de variables

    __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

;------------------ Démarrage sur RESET --------------------
    org 0x000             ; Adresse de départ après reset
    goto    init          ; Adresse 0: initialiser

;------------------ Initialisation -------------------------
init
    bsf     STATUS,RP0    ; sélectionner banque 1
    movlw   0x20          ;
    movwf   TRISB         ; RB5 entrée et les autres sorties
    bcf     STATUS,RP0    ; repasser banque 0

;------------------ Programme principal --------------------
debut
    btfsc   PORTB,5       ; RB5 = 0?
    goto    arr           ; sinon aller à arr
    movlw   0x41          ; si oui RB6=1, RB0=1 et RB7=0
    goto    ma
arr
    movlw   0x80          ; RB6=0, RB0=0 et RB7=1
ma
    movwf   PORTB         ; (w) --> (PORTB
    goto    debut         ; recommencer
    end
```

EXERCICES PROPOSER AVEC CES SOLUTIONS

Exercice 46:

1) Organigramme



2) Sous-programme en assembleur

```
Retard      movlw  0x0A        (N = 10)
            movwf  COUNT
Delai       decfsz COUNT,f
            goto   Delai
            return
```

Etiquette	Instruction	Durée en cycle
Retard	Movlw 0x0A	1
	Movwf Count	1
Delai	Decfsz Count,1	3x10-1=29
	Goto Delai	
	return	2

☞ **Notion:**

Durée en cycle des instructions de sauts conditionnels

BTFSC f,b BTFSS f,b DECFSZ INCFSZ

1 cycle s'il n'y a pas de saut et 2 cycles s'il y a saut

Durée en cycle des instructions de sauts inconditionnels

GOTO k CALL k RETURN RETLW RETFIE

Toujours 2 cycles

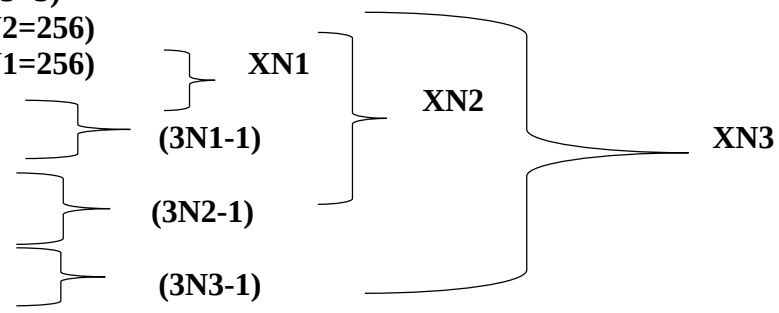
Durée en cycle = 1 + 1 + 2 9 + 2 = 33 cycles.

Un cycle dure : 4/8 Mhz = 0,5 µs **Durée : T = 33 cycles x 0,5 µs =16,5 µs**

4. l'instruction qui permet d'appeler un sous-programme est CALL.

Exercice 47

```
tempo movlw 0x05
      movwf cptr3    (N3=5)
b3    clrf  cptr2    (N2=256)
b2    clrf  cptr1    (N1=256)
b1    decfsz cptr1, f
      goto  b1
      decfsz cptr2, f
      goto  b2
      decfsz cptr3, f
      goto  b3
      return
```



1. Durée :

Durée d'un cycle : 4 / 10Mhz = 0,4 µs

T = [1 + 1 + 1xN1xN2xN3 + (3xN1 –1) xN2xN3 + (3xN2 –1) xN3 + 3xN3 – 1 + 2] x 0,4µs

T = [4 + N1xN2xN3 + 3xN1 xN2xN3 - N2xN3 + 3xN2xN3 - N3 + 3xN3 – 1] x 0,4µs

T = [3 + 4xN1 xN2xN3 + 2xN2xN3 + 2xN3] x 0,4µs

T = [3 + ((4xN1 + 2) x N2 + 2) x N3] x 0,4µs

T = [3 + ((4x256 + 2) x256 + 2) x 5] x 0,4µs = 1313293 x 0,4µs = 525317,2 µs

T ≈ 525,32ms

2. Durée :

Durée d'un cycle : 4 / 4Mhz = 1 µs

T = [3 + ((4x256 + 2) x256 + 2) x 203] x 1µs = 53319577 x 1µs = 53319577µs

T = 53319,577ms = 53,319577s

Exercice 48 :

Pour que chaque led clienté 2 seconde il faut que :

ctr3:25;

ctr2:1;

ctr1:1;

```
LIST p=16f84A ; Définition de processeur
INCLUDE "p16f84A.inc" ; Définitions de variables
__CONFIG _CP_OFF & _XT_OSC & _PWRTE_OFF & _WDT_OFF ; fixe les ;fusibles
cblock 000C;
ctr3:1;
ctr2:1;
ctr1:1;
endc;
org 00
```

```
goto init
init;
bsf STATUS,RP0 ; bank 1
MOVLW 00 ; pour configurer RB0 en entrée et les autres ;en sortie
MOVWF TRISB
bcf STATUS,RP0 ; bank 0
debut
bcf STATUS,C
MOVLW 80
CLRF PORTB
INCF PORTB
NAD RRF PORTB
call Dela
btfss PORTB,0
GOto NAD
CLRF PORTB
goto debut
Dela
movlw 25;
movwf ctr3;
t3 clrf ctr2;
t2 clrf ctr1;
t1 decfsz ctr1;
GOTO t1;
decfsz ctr2;
GOTO t2;
decfsz ctr3;
GOTO t3;
return
END
```

Déclaration Input/output

Programme principale

Procédure de délai de 2 seconde nommé **Dela**

Exercice 48 :

1) 16 : mide range F: Mémoire Flash A:20MHz

Soit le début de programme MPLAB suivant :

2) Signifier les Syntaxes suivants :

CP_OFF: Accès de lire de programme.

XT_OSC: Type d'oscillateur

WDT_OFF: Watch dog desactivé

T=4/20 10^6=0.2 µs

3) Durée d'un cycle : 4 / 20Mhz = 0.2 µs

N=18,19 ou 12H (voir exercice 47)

4)

```
bsf STATUS,RP0 ; bank 1
CLRF TRISB ; pour configurer RB0 en entrée et les autres ;en sortie
MOVLW 09
MOVWF TRISA
bcf STATUS,RP0 ; bank 0
```

5)

```
clrf PORTA
clrf PORTB
HH btfss PORTA,0
goto HH
NAD RLF PORTB
btfsc PORTA,3
GOTO ss
call Delay
btfss PORTB,4
GOto NAD
GOTO mop
mop rrf PORTB
CALL Delay
btfsc PORTA,3
GOTO ss
btfss PORTB,0
GOTO mop
GOTO NAD
ss clrf PORTB
```

☞ **Remarque :**

Pour résolution des exercices 48 et 49 on a confirmé avec logiciel porteuse bien sûr a l'intermédiaire Microship software

Bibliographie :

[1] J.Lassi Khaled, Microprocesseurs et Microcontrôleurs (microprocesseur) 8086, Université Virtuelle de Tunis

[2] M.Aumiaux , Les systèmes à microprocesseurs ,2^{éme} édition .Masson, 1982

[3] Intel The 8086 Family, Users Manual . 1979.

[4] Dr J.Y. Haggége, Microprocesseurs ,Cours Institut Supérieur des études Technologiques de Radés ,2003

[5] A. Oumnad, Microprocesseurs de la famille 8086.

[6] Dr. Okba Benelmir, Microprocesseurs et ses interfaces. Université de Biskra Algérie

[7] A.Hmiden, Microprocesseurs 8086. Institut Supérieur des Etudes Technologiques de Sousse Tunis

[8] J W. Coffroun, Programmation du 8086 – 8088, Edition SYBEX, 1983

[9] P. André, La liaison RS232, Dunod , Paris, 1998.

[10] A.B. Fontaine, Le microprocesseur 16 bits 8086/8088, Matériel logiciel et système d'exploitation Masson, Paris, 1988.

[11] B. Geoffrion. 8086 - 8088 , Programmation en langage assembleur, Editions Radio, Paris, 1986.

[12] H. Lilen. Microprocesseurs ,Du CISC au RISC, Dunod, Paris, 1995.

[13] G. H. MacEwen, Introduction to Computer Systems, McGraw-Hill, Tokyo, 1981.

[14] A. Mariatte, PC, modems et serveurs. P.S.I, Lagny, France, 1986.

[15] http://www.bigonoff.com/

[16] Bignouf Seconde partie, Révision 12 La Gamme Mid-Range ,Par L'etude Des16F87X (16F876-16F877)