

*Ministère de l'Enseignement Supérieur et de la Recherche  
Scientifique*

**Université Mohamed Boudiaf - M'sila**

**Faculté de Technologie  
Département d'Electronique**

**Option : Electronique des systèmes embarques**

*Cycle : Master Electronique*

*Support de Cours*

***Conception des Systèmes à Microprocesseur***

*Par :*  
***Dr .Garah Messaoud***

***Année universitaire : 2021/2022***

## **PRESENTATION**

Ce module de compétence générale fait appel aux préalables suivants : Logique combinatoire et séquentielle, Architecture des systèmes à microprocesseurs 8 bits.

### **Description du Module**

Ce document a été rédigé pour servir de support du cours **Conception des systèmes à microprocesseur** du Master *électronique des systèmes embarqués* et s'adresse principalement aux étudiants de la première année.

L'objectif de ce cours consiste à faire connaître les principes de fonctionnement et l'architecture d'un système à microprocesseur 16 bits, il permet, d'une part, aux étudiants de savoir manipuler le jeu d'instructions et les directives d'assemblage d'un microprocesseur, maîtriser la programmation en langage assembleur et comprendre les mécanismes d'interruption, et d'autre part de faire la conception et la réalisation des montages à base de microprocesseur 16 bits et des circuits d'interfaces : systèmes d'acquisition/ transmission de données, pilotage de convertisseurs.

Sur la base de ces objectifs, ce cours est divisé en Cinq chapitres :

Le premier chapitre est destiné aux notions de base sur les microprocesseurs : organisation interne des Microprocesseurs, organisation des informations (données, instructions, adresses) et bus, différents types de processeurs (microprocesseur standard, microcontrôleur, DSP, API, etc.). Architectures (Von Neumann, Harvard), CISC, RISC.

Le deuxième chapitre présente l'étude des Système à microprocesseur. Interfaçage avec le monde extérieur, Mémoires (Différents types, Conception d'un plan mémoire, Décodage d'adresses). Circuits d'entrées-sorties (Différents types et usages). Les interruptions (Causes, Interruptions matérielles, logicielles, Traitement des interruptions). Pile et ses utilisations.

Pour le troisième chapitre l'étude d'un microprocesseur 16 bits sera le sujet principal, débutant par la présentation de l'architecture interne et le brochage, ce chapitre va aborder

par la suite le traitement des instructions, la file d'attente. Les différents registres, ainsi que la gestion de la mémoire.

Le quatrième chapitre illustre les techniques de programmation. Dans ce chapitre l'étudiant va apprendre les modes d'adressage, l'étude du jeu d'instructions ainsi, une introduction au traitement programmé (Algorithme, Organigramme, Structure d'un programme) sera expliquée afin de maîtriser la programmation en langage assembleur.

L'objectif de cinquième chapitre est la gestion des circuits d'entrées-sorties avec des exemples d'utilisation de ces interfaces dans un système à microprocesseur.

La deuxième partie de ce support de cours est consacrée au guide de travaux dirigés et pratiques

---

## SOMMAIRE

---

Présentation.....	2
Description du Module .....	2

### *Chapitre I : Notions de base sur les microprocesseurs*

I.1. Historique.....	1
I.2. Architecture de base d'un ordinateur.....	1
I.2.1. Modèle de Von Neumann .....	1
I.2.2. Modèle de Harvard .....	2
I.2.3. Définitions .....	2
I.2.3.1. L'unité centrale.....	3
I.2.3.2. La mémoire principale.....	3
I.2.3.3. Les interfaces d'entrées/sorties.....	3
I.2.3.4. Les bus.....	3
I.3. Notion d'architecture RISC et CISC.....	3
I.3.1. Architecture CISC.....	4
I.3.2. Architecture RISC.....	4
I.3.3. Améliorations de l'architecture de base.....	5
I.4. Processeurs spéciaux.....	5
I.4.1 Les microcontrôleurs .....	5
I.4.2 Le processeur de signal. ....	5

### *Chapitre II : Système à microprocesseur*

II.1. Les mémoire .....	7
II.1.1. Organisation d'une mémoire.....	7
II.1.2. Caractéristiques d'une mémoire .....	8
II.1.3. Les mémoires vives (RAM).....	9
II.1.3.1. Les RAM statiques .....	9
II.1.3.2 Les RAM dynamiques.....	9
II.1.4. Les mémoires mortes (ROM).....	9
II.1.4.1. ROM.....	9
II.1.4.2. PROM.....	10
II.1.4.3. UPROM.....	10
II.1.4.4. EEPROM.....	10
II.1.4.5. FLASH EPROM.....	10
II.2. Décodage d'adresses. ....	11
II.3. Circuits d'Entrées/Sorties. ....	12
II.3.1 Définition. ....	12
II.3.2 Différentes types d'interfaces E/S.....	13
II.3.3 Schéma synoptique d'un circuit d'E/S .....	13
II.3.4 Adressage des ports d'E/S .....	13
II.4. Gestion des ports d'E/S par le 8086.....	13

---

---

II.5. Accès aux registres d'un circuit d'interface .....	15
II.6. Le circuit d'interface série .....	15
II.7. Le circuit d'interface parallèle 8255.....	16
II.8. Gestion d'interruptions.....	16
II.8.1. Définition d'une interruption.....	16
II.8.2 Types et propriétés des interruptions.....	17
II.8.2.1. Les interruptions matérielles.....	17
II.8.2.2 Les interruptions logicielles .....	18
II.8.3 La table des vecteurs d'interruption.....	18
II.9. Structure de la pile .....	19
II.9.1. Accès à la pile .....	20
II.9.2. Utilisations de la pile.....	22
III.9.2.1. Stockage temporaire .....	22
III.9.2.2. Les appels aux sous-programmes et la pile.....	22
III.9.2.3. Passage des paramètres dans la pile .....	22

### *Chapitre III : Etude d'un microprocesseur 16 bits*

III.1. Introduction.....	23
III.2. Architecture externe du 8086.....	23
III.3. Caractéristiques du 8086 .....	24
III.4. Architecture interne du 8086.....	24
III.5. Fonctions de l'UIB (Unité d'Interface de Bus) .....	25
III.6. Fonctions de l'UE (Unité d'Exécution).....	25
III.7. Les registres du 8086 .....	25
III.7.1. Les registres généraux .....	26
III.7.2. Les registres segments.....	27
III.7.3. Le registre IP : (Le compteur de programme) .....	28
III.7.4.. Le registre d'état (Flag) .....	28
III.8. Gestion de la mémoire.....	30
III.9. Fonctionnement d'un microprocesseur.....	32

### *Chapitre IV : La programmation en langage assembleur*

IV.1. Introduction .....	35
IV.2. syntaxe d'une instruction...: .....	35
IV.3. Les groupes d'instructions .....	36
IV.3.1. Les instructions de transfert de données.....	36
IV.3.2. Les instructions arithmétiques .....	40
IV.3.3. Les instructions logiques .....	44
IV.3.4. Les instructions de décalage et de rotation .....	45

---

---

IV.3.5. Les instructions de pile .....	47
IV.3.5.1 L'instruction d'empilement (push) .....	47
IV.3.5.2 L'instruction de d'empilement (pop) .....	48
IV.3.5.3 Autres instruction de la pile .....	49
IV.3.6 Les instructions de branchement.....	49
IV.3.7 Les instructions de boucles .....	54
IV.4. Déclaration de variables en assembleur .....	61
IV.5 Manipulation de chaînes de données .....	62

## Chapitre V : Les interfaces d'entrées/sorties

V.1. Types de liaisons .....	66
V.1.1. Liaison parallèle .....	66
V.1.2. Liaison série.....	66
V.2. Interface d'entrées/sorties .....	68
V.2.1. Définition.....	68
V.2.2. L'interface parallèle 8255.....	68
V.2.3. l'interface série, l'UART 8250 .....	74
V.3.4. Timer (Temporisateur), Intel 8254.....	77
V.3.5. Le contrôleur programmable d'interruptions 8259.....	79

## **BIBLIOGRAPHIE** 66

Problème : Les séries des Travaux dirigés (Les TDs) 67

Annexe : Les séries des Travaux pratiques (Les TPs) 81

---

## **Chapitre I :**

### **Notions de base sur les microprocesseurs**

#### **I.1 Historique**

- 1947 Invention du transistor
- 1958 TEXAS INSTRUMENTS produit le 1er circuit intégré (CI)
- 1961 Mise au point des technologies bipolaires TTL et ECL
- 1964 Intégration à petite échelle (SSI de 1 à 10 transistors)
- 1965 Intégration à moyenne échelle (MSI de 10 à 500 transistors)
- 1970 Mise au point de la technologie MOS
- 1971 Intégration à grande échelle (LSI de 500 à 20 000 transistors)
- 1985 Intégration à très grande échelle (VLSI plus de 20000 transistors)

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. 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 fabricant. Fort de ses 6000 transistors, le 8080, doté d'une horloge à 2MHz effectue 640000 instructions par seconde. En 1974, MOTOROLA, autre fondeur de silicium, décide de lancer le 6800 qui constituera lui aussi le début d'une grande famille.

#### **I.2 Architecture de base d'un ordinateur**

##### **I.2.1 Modèle de Von Neumann**

Pour traiter une information, un microprocesseur seul ne suffit pas, il faut l'insérer au sein d'un système minimum de traitement programmé de l'information. John Von Neumann est à l'origine d'un modèle de machine universelle de traitement programmé de l'information (1946).

Cette architecture (figure 1) sert de base à la plupart des systèmes à microprocesseur actuel. Elle est composée des éléments suivants :

- Une unité centrale
- Une mémoire principale

- des interfaces d'entrées/sorties.

Les différents organes du système sont reliés par des voies de communication appelées bus.

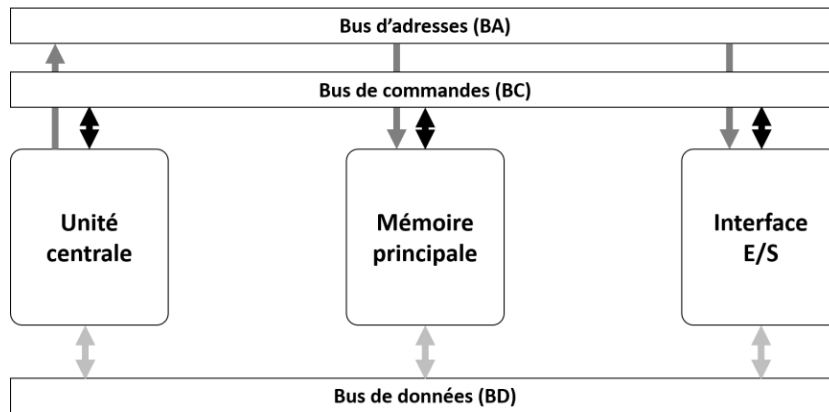


Figure I.1 : Modèle de Von Neumann.

### I.2.2 Modèle de Harvard

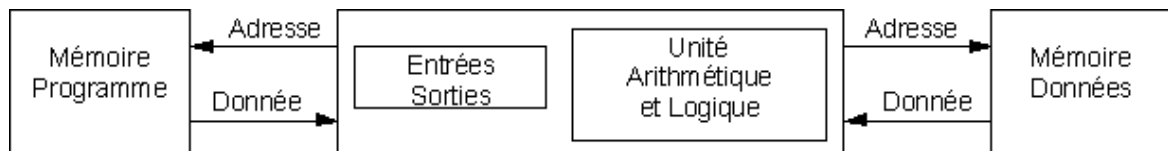


Figure I.2: Modèle de Harvard.

### I.2.3 Définitions

Dans cette partie, nous décrivons rapidement l'architecture de base d'un ordinateur et les principes de son fonctionnement.

**Un ordinateur** est une machine de traitement de l'information. Il est capable d'acquérir de l'information, de la stocker, de la transformer en effectuant des traitements quelconques, puis de la restituer sous une autre forme.

Le mot **informatique** vient de la contraction des mots information et automatique. On peut définir l'informatique comme étant le traitement automatique de l'information.

Nous appelons **information** tout ensemble de données. On distingue généralement différents types d'informations : textes, nombres, sons, images, etc., mais aussi les instructions composant un programme. Dans un ordinateur, toute information est manipulée sous forme binaire (ou numérique).



### I.2.3.1 L'unité centrale

Elle est composée par le microprocesseur qui est chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Toutes les activités du microprocesseur sont cadencées par une horloge.

On caractérise le microprocesseur par :

- Sa fréquence d'horloge : en MHz ou GHz
- Le nombre d'instructions par secondes qu'il est capable d'exécuter : en MIPS
- La taille des données qu'il est capable de traiter : en bits.

### I.2.3.2 La mémoire principale

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce programme. Physiquement, elle se décompose souvent en :

- Une mémoire morte (*ROM = Read Only Memory*) chargée de stocker le programme. C'est une mémoire à lecture seule.
- 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 dedans, ces données sont perdues à la mise hors tension.

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

### I.2.3.3 Les interfaces d'entrées/sorties

Elles permettent d'assurer la communication entre le microprocesseur et les périphériques. (Capteur, clavier, moniteur ou afficheur, imprimante, modem, etc...).

### I.2.3.4 Les bus

Un bus est un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

- ***Un bus de données :*** bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.

- **un bus d'adresses** : unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir  $2^n$  emplacements, avec  $n$  = nombre de lignes du bus d'adresses.

**Un bus de commande** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses

### I.3 Notion d'architecture RISC et CISC

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

- L'architecture CISC (*Complex Instruction Set Computer*).
- L'architecture RISC (*Reduced Instruction Set Computer*).

#### I.3.1 L'architecture CISC

C'est une architecture avec un grand nombre d'instructions où le microprocesseur doit exécuter des tâches complexes par instruction unique. Pour une tâche donnée, une machine CISC exécute ainsi un petit nombre d'instructions mais chacune nécessite un plus grand nombre de cycles d'horloge. Le code machine de ces instructions varie d'une instruction à l'autre et nécessite donc un décodeur complexe (microcode).

#### I.1.3.2 L'architecture RISC

C'est une architecture dans laquelle les instructions sont en nombre réduit (chargement, branchement, appel sous-programme). Les architectures RISC peuvent donc être réalisées à partir de séquenceur câblé. Leur réalisation libère de la surface permettant d'augmenter le nombre de registres ou d'unités de traitement par exemple. Chacune de ces instructions s'exécutent ainsi en un cycle d'horloge. Bien souvent, ces instructions ne disposent que d'un seul mode d'adressage. Les accès à la mémoire s'effectuent seulement à partir de deux instructions (*Load et Store*). Par contre, les instructions complexes doivent être réalisées à partir de séquences basées sur les instructions élémentaires, ce qui nécessite un compilateur très évolué dans le cas de programmation en langage de haut niveau.

#### I.1.3.3 Améliorations de l'architecture de base

L'ensemble des améliorations des microprocesseurs visent à diminuer le temps d'exécution du programme.

La première idée qui vient à l'esprit est d'augmenter tout simplement la fréquence de l'horloge du microprocesseur. Mais l'accélération des fréquences provoque un surcroît de consommation ce qui entraîne une élévation de température. On est alors amené à équiper les processeurs de systèmes de refroidissement ou à diminuer la tension d'alimentation.

Une autre possibilité d'augmenter la puissance de traitement d'un microprocesseur est de diminuer le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction. Dans le cas d'une programmation en langage de haut niveau, cette amélioration peut se faire en optimisant le compilateur. Il faut qu'il soit capable de sélectionner les séquences d'instructions minimisant le nombre moyen de cycles par instructions. Une autre solution est d'utiliser une architecture de microprocesseur qui réduise le nombre de cycles par instruction.

### **I.4 Processeurs spéciaux**

#### **I.4.1 Les microcontrôleurs**

Ce sont des systèmes minimums sur une seule puce. Ils contiennent un CPU, de la RAM, de la ROM et des ports d'Entrée/Sorties (parallèles, séries, I2C, etc..). Ils comportent aussi des fonctions spécifiques comme des compteurs programmables pour effectuer des mesures de durées, des CAN voir des CNA pour s'insérer au sein de chaînes d'acquisition, des interfaces pour réseaux de terrain, etc ...

Il est adapté pour répondre au mieux au besoin des applications embarquées (appareil Électroménagers, chaîne d'acquisition, lecteur carte à puce, etc...). Il est par contre généralement moins puissant en termes de rapidité, de taille de données traitables ou de taille de mémoire adressable qu'un microprocesseur.

#### **I.4.2 Le processeur de signal**

Le processeur de signal est beaucoup plus spécialisé. Alors qu'un microprocesseur n'est pas conçu pour une application spécifique, le processeur DSP (Digital Signal Processor) est optimisé pour effectuer du traitement numérique du signal (calcul de FFT, convolution, filtrage numérique, etc...).

Les domaines d'application des D.S.P étaient à l'origine les télécommunications et le secteur

Militaire. Aujourd'hui, les applications se sont diversifiées vers le multimédia (lecteur CD, MP3, etc..) l'électronique grand public (télévision numérique, téléphone portable, etc...), l'automatique, l'instrumentation, l'électronique automobile, etc.

## Chapitre II

### Systèmes à microprocesseur

#### II.1 Les mémoires

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.

##### II.1.1 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.

Adresse	Case mémoire
7 = 111	
6 = 110	
5 = 101	
4 = 100	
3 = 011	
2 = 010	
1 = 001	
0 = 000	0001 1010

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 suivante où l'on peut distinguer :



Les entrées d'adresses

Les entrées de données

Les sorties de données

Les entrées de commandes :

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

Une opération de lecture ou d'écriture de la mémoire suit toujours le même cycle :

1. sélection de l'adresse
2. choix de l'opération à effectuer (R/W)
3. sélection de la mémoire (CS = 0)
4. lecture ou écriture la donnée

### II.1.2 Caractéristiques d'une mémoire

**La capacité** : c'est le nombre total de bits que contient la mémoire. Elle s'exprime aussi souvent en octet.

**Le format des 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.

**Le temps d'accès** : c'est le temps qui s'écoule 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.

**Le temps de cycle :** il représente l'intervalle minimum qui doit séparer deux demandes Successives de lecture ou d'écriture.

**Le 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.

### **II.1.3 Les 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 Acces Memory : mémoire à accès aléatoire) :

- Les RAM statiques
- Les RAM dynamiques

#### **II.1.3.1 Les RAM statiques**

Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors.

#### **II.1.3.2 Les RAM dynamiques**

Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur. 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.

### **II.1.4 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, et contrairement aux RAM, ne peuvent être que lue.

#### **II.1.4.1 La ROM**

Elle est programmée par le fabricant et son contenu ne peut plus être ni modifiée, ni effacée par l'utilisateur.

**Avantage :**

- Non volatile
- Mémoire rapide

**Inconvénients :**

- Écriture impossible
- Modification impossible (toute erreur est fatale)
- Délai de fabrication (3 à 6 semaines)
- Obligation de grandes quantités en raison du coût élevé qu'entraîne la production du masque et le processus de fabrication.

### II.1.4.2 La PROM

C'est une ROM qui peut être programmée une seule fois par l'utilisateur (Programmable ROM). La programmation est réalisée à partir d'un programmeur spécifique.

**Avantage :**

- Claquage en quelques minutes
- Coût relativement faible

**Inconvénients :**

- Modification impossible (toute erreur est fatale)

### II.1.4.3 L'EPROM ou l'UV-EPROM

Pour faciliter la mise au point d'un programme ou tout simplement permettre une erreur de programmation, il est intéressant de pouvoir reprogrammer une PROM. La technique de claquage utilisée dans celles-ci ne le permet évidemment pas. L'EPROM (Erasable Programmable ROM) est une PROM qui peut être effacée.

**Avantage :**

- Reprogrammable et non Volatile

**Inconvénients :**

- Impossible de sélectionner une seule cellule à effacer
- l'écriture est beaucoup plus lente que sur une RAM (environ 1000x).

### II.1.4.4 L'EEPROM



L'EEPROM (Electrically EPROM) est une mémoire programmable et effaçable électriquement. Elle répond ainsi à l'inconvénient principal de l'EPROM.

**Avantage :**

- Comportement d'une RAM non Volatile.
- Programmation et effacement mot par mot possible.

**Inconvénients :**

- Très lente pour une utilisation en RAM.
- Coût de réalisation.

### II.1.4.5 La FLASH EPROM

La mémoire Flash s'apparente à la technologie de l'EEPROM. Elle est programmable et effaçable électriquement comme les EEPROM.

**Avantage :**

- Comportement d'une RAM non Volatile.
- Programmation et effacement mot par mot possible.

**Inconvénients :**

- Très lente pour une utilisation en RAM.
- Coût de réalisation.

## II.2 Décodage d'adresses

La multiplication des périphériques autour du microprocesseur oblige la présence d'un décodeur d'adresse chargé d'aiguiller les données présentes sur le bus de données. En effet, le microprocesseur peut communiquer avec les différentes mémoires et les différents boîtiers d'interface. Ceux-ci sont tous reliés sur le même bus de données et afin d'éviter des conflits, un seul composant doit être sélectionné à la fois.

Lorsqu'on réalise un système à microprocesseur, on attribue donc à chaque périphérique une zone d'adresse et une fonction « décodage d'adresse » est donc nécessaire afin de fournir les signaux de sélection de chacun des composants.

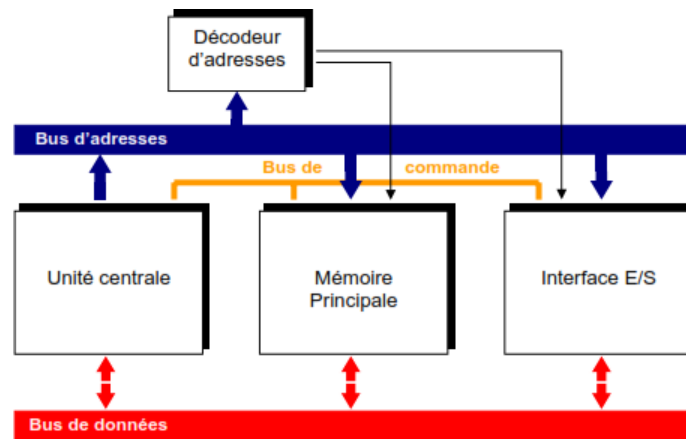


Fig. II.1 décodage d'adresses

**Remarque :** lorsqu'un composant n'est pas sélectionné, ses sorties sont mises à l'état « haute

Impédance » afin de ne pas perturber les données circulant sur le bus. (elle présente une impédance de sortie très élevée = circuit ouvert).

### II.3 Circuits d'Entrées/Sorties

#### II.3.1 Définition

Une interface d'entrées/sorties est un circuit intégré permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante, modem, disques, processus industriel, ... Les interfaces d'E/S sont connectées au microprocesseur à travers les bus d'adresses, de données et de commande.

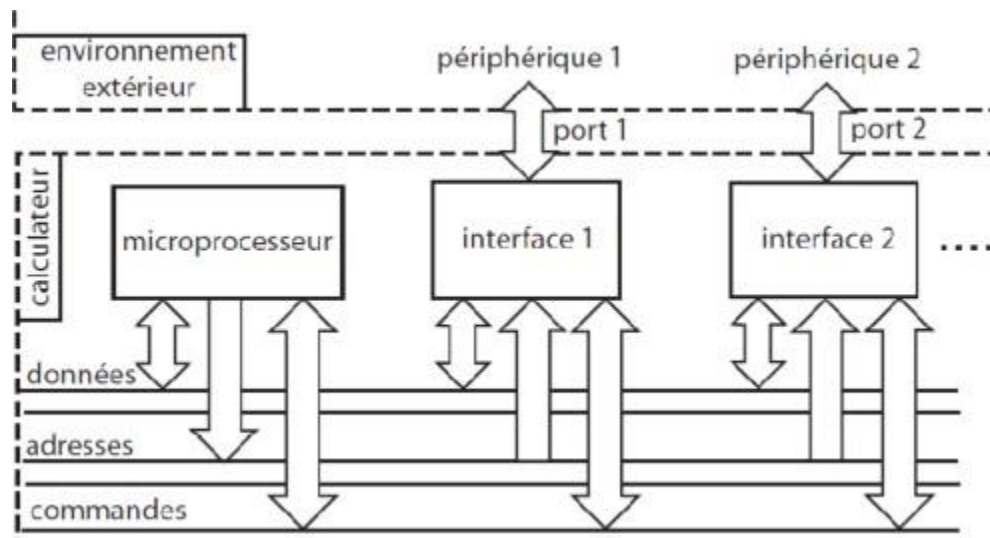


Fig. II.2 Circuits d'entrées/sorties

Les points d'accès aux interfaces sont appelés ports.

### II.3.2 Différentes types d'interfaces E/S

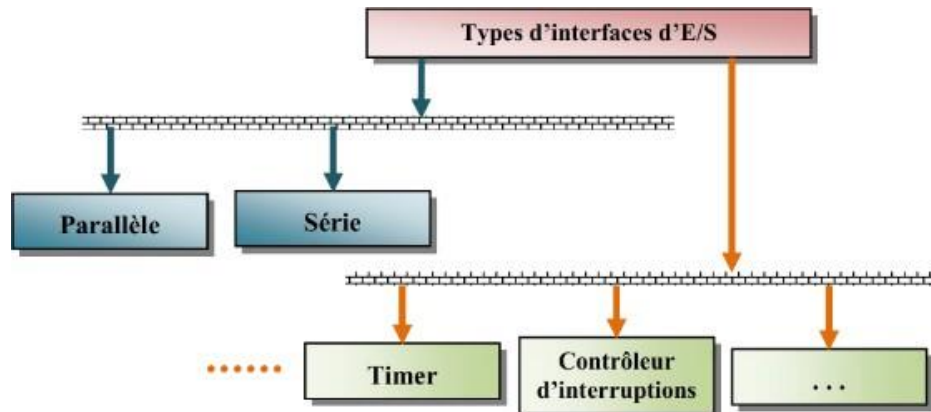
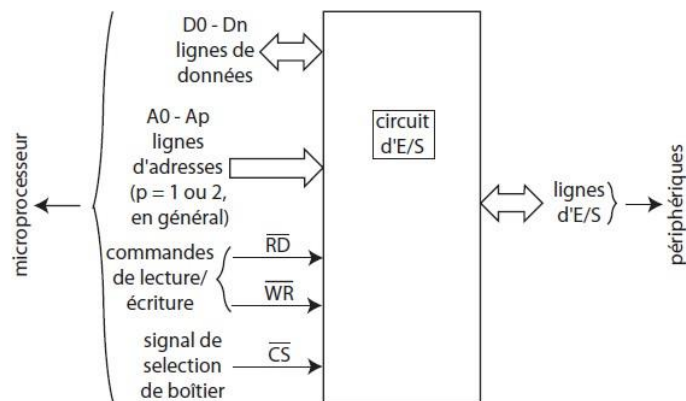


Fig. II.3 Différentes types d'interfaces E/S

### II.3.3 Schéma synoptique d'un circuit d'E/S :



II.3.4 Circuit d'E/S

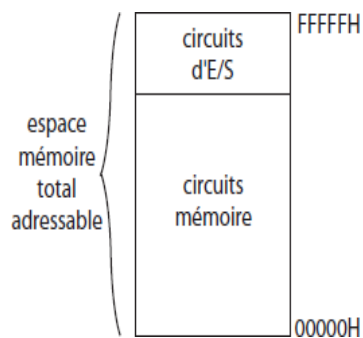
### II.3.5 Adressage des ports d'E/S

Un circuit d'E/S possède des registres pour gérer les échanges avec les périphériques :

- registres de configuration ;
- registres de données.

A chaque registre est assigné une adresse : le microprocesseur accède à un port d'E/S en spécifiant l'adresse de l'un de ses registres. Le microprocesseur peut voir les adresses des ports d'E/S de deux manières :

- **adressage cartographique** : les adresses des ports d'E/S appartiennent au même espace mémoire que les circuits mémoires (on dit que les E/S sont **mappées en mémoire**) :



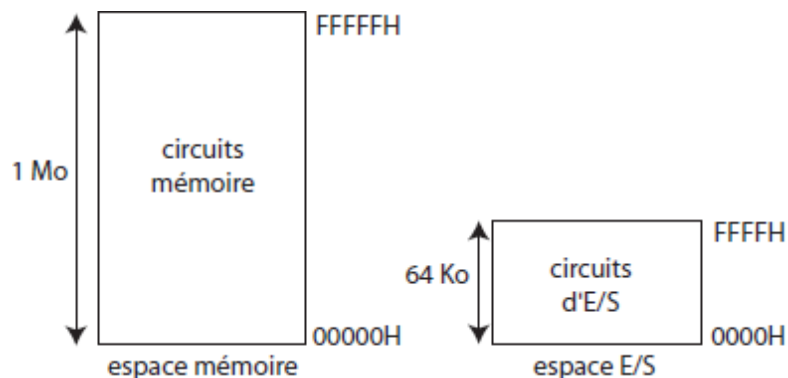
### Conséquences :

- l'espace d'adressage des mémoires diminue ;
- l'adressage des ports d'E/S se fait avec une adresse de même longueur (même nombre de bits) que pour les cases mémoires ;
- toutes les instructions employées avec des cases mémoires peuvent être appliquées aux ports d'E/S : les mêmes instructions permettent de lire et écrire dans la mémoire et les ports d'E/S, tous les modes d'adressage étant valables pour les E/S.

- **Adressage indépendant** : le microprocesseur considère deux espaces distincts :

- l'espace d'adressage des mémoires ;
- l'espace d'adressage des ports d'E/S.

C'est le cas du microprocesseur 8086 :



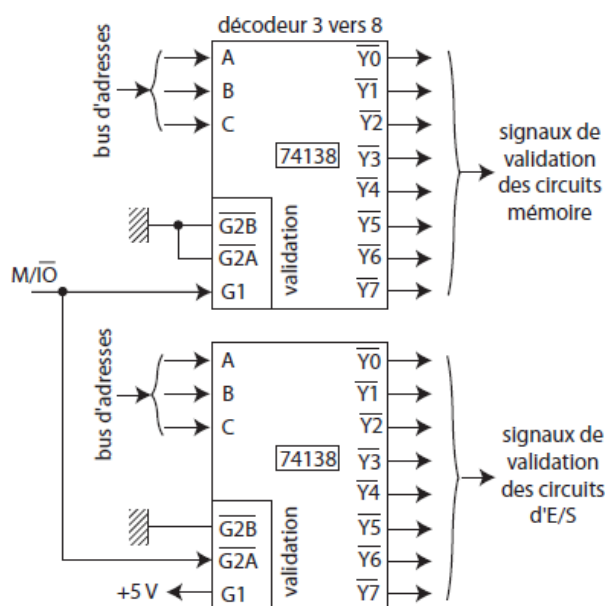
## II.4 Gestion des ports d'E/S par le 8086

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/I $\bar{O}$**  :

- pour un accès à la mémoire, M/I $\bar{O}$  = 1 ;
- pour un accès aux ports d'E/S, M/I $\bar{O}$  = 0.

Ce signal est utilisé pour valider le décodage d'adresse dans les deux espaces :



## II.5 Accès aux registres d'un circuit d'interface :

Chaque registre interne est associé à une adresse d'E/S. Pour y accéder, il existe deux instructions **IN** et **OUT** :

**IN AL,DX** ; met dans AL le contenu de l'adresse d'E/S pointé par DX

**OUT DX,AL** ; transfère le contenu de AL vers l'adresse d'E/S pointée par DX

Il faut évidemment que DX soit chargé au préalable par l'adresse d'entrée/sortie :

**MOV DX,adr. d'E/S.**

AL et DX sont les seuls registres autorisés dans les opérations d'E/S.

Exemples :

- lecture d'un port d'E/S sur 8 bits à l'adresse 300H :

```
mov dx,300H
```

```
in al,dx
```

- Écriture de la valeur 1234H dans le port d'E/S sur 16 bits à l'adresse 49H :

```
mov ax,1234H
```

```
out 49H,ax
```

### II.6 Le circuit d'interface série

Tout ordinateur est équipé d'un ou deux ports série permettant le transfert des données vers d'autres systèmes (modem par exemple). La gestion des ports série est prise en charge par un circuit appelé l'UART (Universal Asynchronous Receiver Transmitter).

L'UART 8250 possède plusieurs registres internes permettant de configurer la communication, d'envoyer ou recevoir des données.

Le premier de ces registres s'appelle RBR (Receiver Buffer) ou registre de réception. Il se situe à l'adresse d'E/S : 3F8h. Cette adresse est appelée **adresse de base** car les adresses de tous les autres registres de l'UART sont calculées par rapport à cette adresse en ajoutant un décalage. Ainsi, le registre : LCR (Line Control Register) ou registre de contrôle de la transmission se trouve à l'adresse  $3F8h + 3 = 3FBh$ . Le dernier registre est situé à 3FFh

L'UART occupe donc l'espace d'E/S : 3F8h – 3FFh.

### II.7 Le circuit d'interface parallèle 8255 (filière IEEA uniquement)

Le 8255 est un circuit d'interface programmable de 24 bits d'entrée/sortie organisés sous forme de 2 ports de 8 bits : A et B, et de deux ports de 4 bits Csup et Cinf (constituant le port C).

Chaque port est programmable en entrée et en sortie.

Le 8255 est programmé à travers son registre de commande.

Le 8255 est vu par le microprocesseur comme étant 4 octets dans l'espace des entrées/sorties

### II.8 Gestion d'interruptions

#### II.8.1 Définition d'une interruption :

Une interruption est une demande non prévisible adressée au microprocesseur. Elle provoque l'arrêt du programme en cours de traitement pour exécuter en priorité un sous-programme spécifique appelé sous-programme d'interruption ou traitant d'interruption.

A la fin de ce sous-programme, le microprocesseur reprend le programme principal là où il a été interrompu.

### **II.8.2 Types et propriétés des interruptions**

Il existe deux catégories d'interruptions : les interruptions matérielles et logicielles.

#### **II.8.2.1 Les interruptions matérielles**

Ce sont des interruptions d'origine externe. Elles sont déclenchées par application d'un état haut sur INTR (Interrupt Request) ou NMI (No Masquable Interrupt).

NMI est dite interruption non masquable. Elle ne peut être inhibée par programme. Elle est associée en général à des événements catastrophiques (erreur système, erreur mémoire, ...)

INTR est moins prioritaire que NMI et peut être masquée par programme.

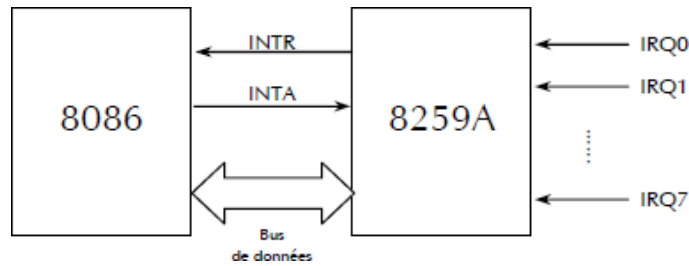
#### ***-Processus de l'interruption NMI***

Lorsque le microprocesseur reçoit une demande d'interruption sur NMI.

1. il termine l'instruction en cours d'exécution,
2. empile le registre d'état et les registres CS et IP,
3. met le drapeau IF (Interrupt Flag) à 0 (inhibition des interruptions),
4. cherche l'adresse du sous-programme à exécuter dans la table des vecteurs d'interruption,
5. exécute le sous-programme ...
6. remet le drapeau IF à 1 (autorisation des interruptions),
7. dépile IP, CS et le registre d'état et poursuit le programme principal.

#### ***-Processus de l'interruption INTR***

INTR est souvent associée à un circuit spécialisé appelé contrôleur d'interruption 8259A :



Le 8259A reçoit 8 entrées IRQ0 à IRQ7 (Interrupt ReQuest). Dès que l'une de ces entrées passe à l'état haut, le 8259A émet une demande d'interruption sur INTR :

1. Le 8086 termine l'instruction en cours.
2. envoie au 8259A sur INTA un signal "accusé de réception".
3. le 8259A place sur le bus de données le n° de l'interruption excitée. Ce numéro est reçu alors par le 8086.
4. le 8086 cherche l'adresse du sous-programme associé à cette interruption dans la table des vecteurs d'interruption, la suite du processus est identique à celui de NMI ...

L'interruption INTR peut être masquée par l'instruction **CLI** (CLear Interrupt flag) qui met le drapeau IF à 0. Aucun appel sur INTR n'est alors pris en compte jusqu'à la mise à 1 d'IF par l'instruction **STI** (SeT Interrupt flag).

### II.8.2.2 Les interruptions logicielles

Elles sont déclenchées par l'instruction **INT n** où n désigne le numéro d'interruption. Les sous-programmes associés aux interruptions logicielles sont en général des routines du BIOS ou du DOS, elles permettent au programme d'accéder aux différents périphériques de l'ordinateur.

#### Exemple

Interruption 10h : interruption BIOS (gestion de l'affichage)

Interruption 21h : interruption DOS (gestion des périphériques, des fichiers, ...)

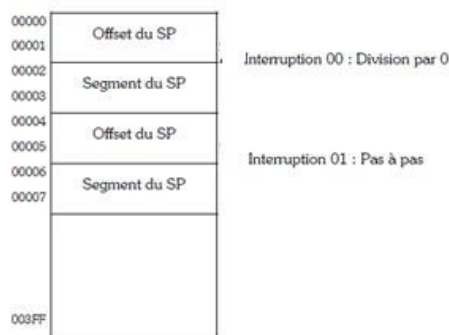
### II.8.3 La table des vecteurs d'interruption

Cette table contient l'adresse des sous-programmes (SP) de toutes les interruptions du microprocesseur (256 au total) et commence à partir de 00000. L'adresse d'un sous-programme d'interruption prend 4 octets : 2 octets pour son segment et 2 octets pour son offset :



La table occupe l'espace mémoire entre 00000h et 003FFh.

Pour une interruption logicielle n l'adresse du sous-programme associé se trouve à  $4n$



(offset du SP) et  $4n+2$  (segment du SP)

### Exemple

L'adresse du SP associé à l'interruption 21h se trouve à 00084h (offset du SP) et 00086h (segment du SP) :

En supposant que le contenu de ces 4 octets est

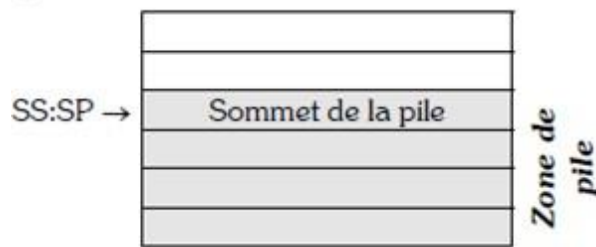
00084h	00085h	00086h	00087h
A0h	04h	45h	09h

## II.9 Structure de la pile

La pile est une zone mémoire gérée en LIFO (Last In, First Out), c'est à dire : le dernier élément entré sera le premier sorti. On peut assimiler une pile à un tube vertical dans lequel on mettra des pièces de monnaie. La seule pièce qu'on peut retirer est la dernière pièce empilée, celle qui existe au sommet.

La pile n'est en réalité qu'une simple zone mémoire dans laquelle sont stockées des données parvenant soit du microprocesseur soit du programme. Avec les langages évolués (Pascal, C, ...) sa présence est quasiment obligatoire.

Le sommet de la pile représente la dernière donnée empilée. Afin de le repérer dans la mémoire, le registre SP (avec SS comme registre de segment) y pointe en permanence. Affecter (imprudemment) une autre valeur à SP au cours du programme signifie la perte de l'adresse du sommet de la pile donc de toutes les données qui y existaient, ce qui se solde, dans la plupart des cas, par un blocage du système.



### II.9.1 Accès à la pile

Vu la structure de la pile, les deux opérations permises sont :

- Empiler une donnée (instruction PUSH)
- Dépiler une donnée (instruction POP)

Pour le 8086, les données empilées et dépilées sont toujours sur 16 bits.

#### - L'empilement

L'empilement d'une donnée est réalisé par l'instruction :

#### **PUSH données**

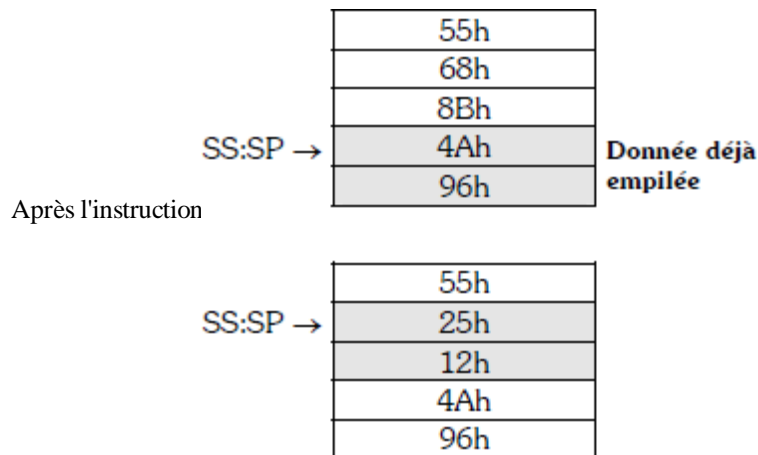
Où **donnée** est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect.

L'empilement s'effectue en deux étapes :

- **$SP \leftarrow SP - 2$**  : on fait reculer SP de deux cases (donnée sur 16 bits), pour qu'il pointe sur le nouveau sommet susceptible de recevoir la donnée à empiler.
- **$[SP] \leftarrow \text{donnée}$**  : le sommet de la pile reçoit la donnée. *Exemple*

Soit  $AX = 1225h$  et soit l'instruction : **PUSH AX**

- Etat de la pile avant l'empilement :



**-Le dépilement :** Le dépilement est l'opération qui permet de retirer le sommet de la pile et le stocker dans une destination. Il est réalisé par l'instruction :

### POP destination

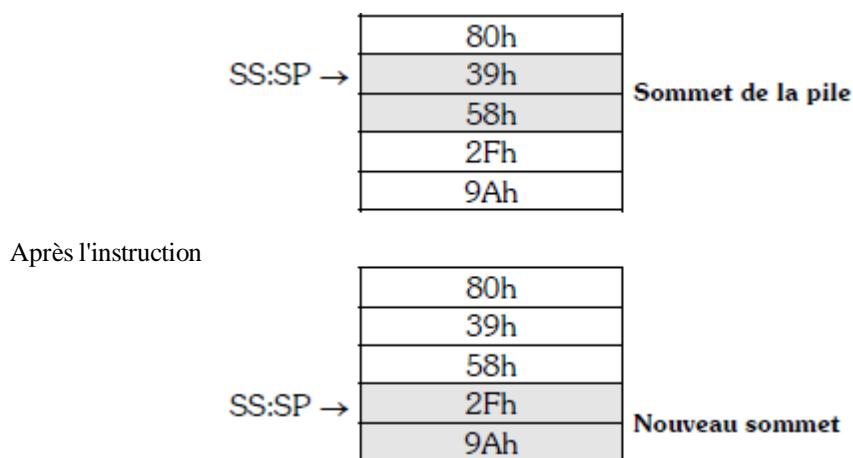
**Destination** est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect. Le dépilement engendre deux opérations :

- **destination** ← **[SP]** : le sommet de la pile est copié dans **destination**.
- **SP** ← **SP + 2** : la donnée du sommet étant retirée, on fait avancer SP pour pointer sur la donnée suivante.

### *Exemple*

Considérons l'instruction : **POP DX**

- Etat de la pile avant l'instruction :



Le registre DX est chargé par la valeur **5839h**.

### **Remarque**

L'instruction POP ne détruit pas la donnée existante au sommet de la pile, elle fait tout simplement avancer SP pour pointer sur la donnée suivante (le nouveau sommet).

### **II.9.2 Utilisations de la pile**

L'une des utilisations les plus classiques de la pile est le stockage temporaire des données. Ceci se manifeste notamment dans les cas suivants :

#### **1 - Le manque de registres**

Parfois, on a besoin de faire une opération faisant intervenir un registre spécifique alors qu'il contient une donnée importante. On peut soit sauvegarder son contenu dans un registre libre soit, si tous les registres sont utilisés, sauvegarder temporairement son contenu dans la pile et le récupérer après l'opération

#### **2- La sauvegarde des registres dans un sous-programme**

##### **II.9.2.2 Les appels aux sous-programmes et la pile**

La pile joue un rôle fondamental dans les appels aux sous-programmes, car, rappelons-le, un sous-programme peut être appelé de plusieurs endroits du programme et le retour s'effectue à l'instruction se trouvant juste après l'appel. La question qui se pose alors est : Comment le microprocesseur se rappelle-t-il de l'adresse de retour ? C'est dans la pile, en effet, qu'il sauvegarde cette adresse.

##### **II.9.2.3 Passage des paramètres dans la pile**

C'est la méthode employée dans tous les langages évolués. En effet, le nombre de paramètres n'est limité ici que par la taille de la pile. Un grand avantage de cette méthode est qu'un sous-programme peut appeler un autre sans perdre ses propres paramètres, il peut même s'auto-rappeler (sous-programme récursif) sans altérer les paramètres reçus pour chaque appel (ceux-ci restent, évidemment, sauvegardés dans la pile).

## **Chapitre III :**

### **Etude d'un microprocesseur 16 bits**

#### **III.1 Introduction**

L'Intel 8086 (également appelé iAPX 86) est un microprocesseur CISC 16 bits fabriqué par Intel à partir de 1978. C'est le premier processeur de la famille x86, qui est devenue l'architecture de processeur la plus répandue dans le monde des ordinateurs personnels, stations de travail et serveurs informatiques.

Il fut lancé en mai 1978. Il est basé sur des registres 16 bits, et dispose d'un bus externe de données de 16 bits ( $D_0 - D_{15}$ ) et d'un bus d'adresse de 20 bits ( $A_0 - A_{19}$ ), qui lui permet d'adresser 1 Mo. Il contient 29 000 transistors gravés en 3  $\mu\text{m}$ . Sa puissance de calcul varie de 0,33 MIPS (lorsqu'il est cadencé à 4,77 MHz comme dans l'IBM PC) jusqu'à 0,75 MIPS pour la version 10 MHz.

Le processeur 8086 d'Intel est à la base des processeurs Pentium. Les processeurs successifs (de PC) se sont en effet construits petit à petit en ajoutant à chaque processeur des instructions et des fonctionnalités supplémentaires, mais en conservant à chaque fois les spécificités du processeur précédent. C'est cette façon d'adapter les processeurs à chaque étape qui permet à un ancien programme écrit pour un 8086 de fonctionner sur un nouvel ordinateur équipé d'un Pentium IV.

#### **III.2 Architecture externe du 8086**

Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) à 40 broches (*figure 1*).

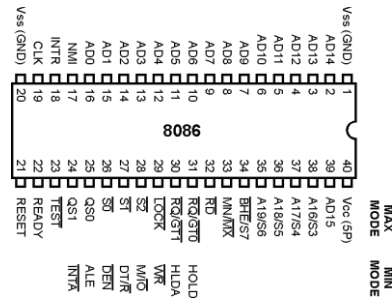


Figure III.1 : Le 8086.

### III.3 Caractéristiques du 8086

Le microprocesseur i8086 se caractérise par :

- Structure 16 bits.
- Capacité d'adressage de 1 Mo.
- 14 registres internes de 16 bits.
- 7 modes d'adressage.
- Opérations sur des bits, des octets, des mots et des chaînes de caractères.
- Arithmétique signée ou non signée.
- Arithmétique binaire ou BCD, sur 8 ou 16 bits.

### III.4 Architecture interne du 8086

Le 8086 se compose de deux unités de traitement séparées : l'unité d'interface de bus (**UIB**) et l'unité d'exécution (**UE**). Les deux unités fonctionnent simultanément, d'où une accélération du processus d'exécution d'un programme (*fonctionnement selon le principe du pipeline*).

L'unité d'exécution comporte l'**UAL**, les registres généraux (**AX, BX, CX, DX**), les registres d'adressage (**SP, BP, SI, DI**), le registre d'état (**Flags**) et le décodeur d'instructions.

L'unité d'interface de bus (**UIB**) comporte une file d'attente d'instructions gérée en **FIFO** (*First In First Out*), les registres de segments (**CS, DS, SS, ES**) et le pointeur d'instruction (**IP**).

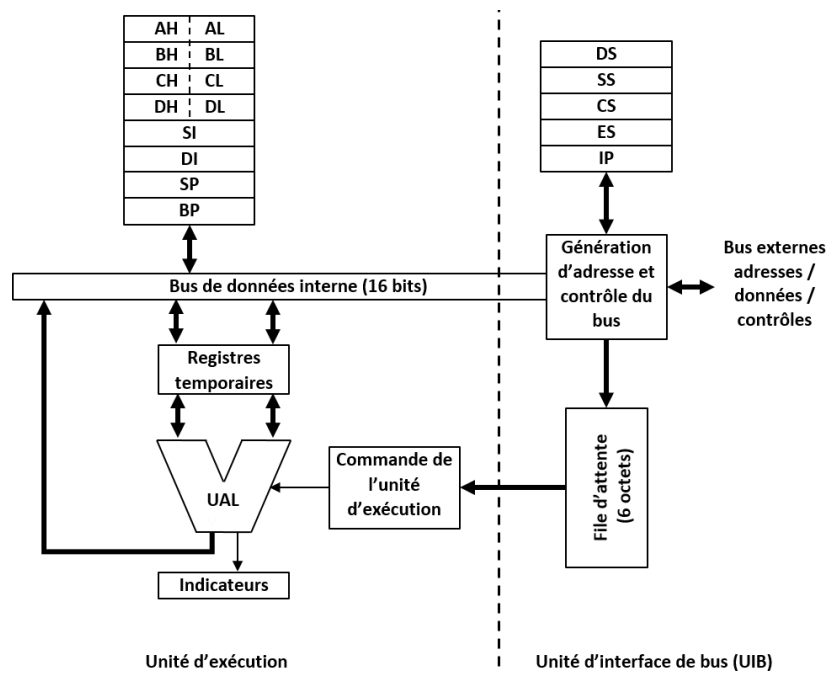


Figure III.2 : Architecture interne du 8086.

### III.5 Fonctions de l'UIB (Unité d'Interface de Bus)

- Elle cherche les instructions à exécuter dans la mémoire et les stocke dans la file d'attente *FIFO*.
- Elle calcule les adresses physiques sur 20 bits.
- Elle réalise le transfert des données avec la mémoire.

### III.6 Fonctions de l'UE (Unité d'Exécution)

- Elle extrait les codes des instructions à partir de la file d'attente et les exécute.
- Elle fournit les adresses des opérandes à l'UIB en nommant le segment concerné et en fournissant le déplacement dans ce segment.

### III.7 Les registres du 8086

Le jeu de registres contient l'ensemble des registres du microprocesseur. Un registre est une petite partie de mémoire intégrée au microprocesseur, dans le but de recevoir des informations spécifiques, notamment des adresses et des données stockées durant l'exécution d'un programme.

Il existe plusieurs types de registres. Certains d'entre eux sont affectés à des opérations d'ordre général et sont accessibles au programmeur à tout moment. Nous

disons alors qu'il s'agit de **registres généraux**. D'autres registres ont des rôles bien plus spécifiques et ne peuvent pas servir à un usage non spécialisé.

### III.7.1 Les registres généraux

Les registres généraux peuvent être utilisés dans toutes les opérations arithmétiques et logiques que le programmeur insère dans le code assembleur. Un registre complet présente une grandeur de 16 bits. Comme le montre la figure 3, chaque registre est en réalité divisé en deux registres distincts de 8 bits. De cette façon, nous pouvons utiliser une partie du registre si nous désirons y stocker une valeur n'excédant pas 8 bits. Si, au contraire, la valeur que nous désirons y ranger excède 8 bits, nous utiliserons le registre complet, c'est à dire 16 bits.

Le programmeur dispose de 8 registres internes de 16 bits qu'on peut diviser en deux groupes comme le montre la figure 3.



Figure III. 3 : Registres généraux du 8086.

#### a. groupe de données :

Ce groupe est formé par 4 registres de 16 bits (**AX, BX, CX, et DX**) chaque registre peut être divisé en deux registres de 8 bits (**AH, AL, BH, BL, CH, CL, DH et DL**).

**Le registre AX (Accumulateur) :** Toutes les opérations de transfert de données avec les entrées-sorties ainsi que le traitement des chaînes de caractères se font dans ce registre, de même les opérations arithmétiques et logiques. Les conversions en BCD du résultat d'une opération arithmétique (addition, soustraction, multiplication et division) se font dans ce registre.

**Le registre BX (registre de base) :** Il est utilisé pour l'adressage de données dans une zone mémoire différente de la zone code : en général il contient une adresse de décalage par rapport à une adresse de référence (*segment de données DS*). De plus il peut servir pour la conversion d'un code à un autre.

**Le registre CX (Le compteur) :** Lors de l'exécution d'une boucle on a souvent recours à



un compteur de boucles pour compter le nombre d'itérations, le registre CX a été fait pour servir comme compteur lors des instructions de boucle.

**Remarque :** Le registre *CL* sert en tant que compteur pour les opérations de décalage et de rotation, dans ce cas il va compter le nombre de décalages (rotation) de bits à droite ou à gauche.

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

### **b. groupe de pointeurs et index :**

Ces registres sont plus spécialement adaptés au traitement des éléments dans la mémoire. Ils sont en général munis de propriétés d'incrémentation et de décrémentation. Un cas particulier de pointeur est le pointeur de pile (***Stack Pointer : SP***). Ce registre permet de pointer la pile pour stocker des données ou des adresses selon le principe du ***LIFO*** (*Last In First Out*).

**L'index SI (source index) :** Il permet de pointer la mémoire il forme en général un décalage (un offset) par rapport à une base fixe (le registre *DS*), il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source.

**L'index DI (Destination index) :** Il permet aussi de pointer la mémoire, il présente un décalage par rapport à une base fixe (*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 pointent sur la zone pile (*une zone mémoire gérée en LIFO*), ils présentent un décalage par rapport à la base (le registre *SS*). Pour le registre *BP*, il a un rôle proche de celui de *BX*, mais il est généralement utilisé avec le segment de pile.

### **III.7.2 Les registres segment**

Le 8086 a quatre registres segments de 16 bits chacun : *CS* (*code segment*), *DS* (*Data segment*), *ES* (*Extra segment*) et *SS* (*Stack segment*), ces registres sont chargés de sélectionner les différents segments de la mémoire en pointant sur le début de chacun d'entre eux. Chaque segment de mémoire ne peut excéder les  $65536 = 2^{16}$  octets (64 Ko).

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

**Remarque :** Si la taille du programme dépasse les 65536 octets alors on peut diviser le code sur plusieurs segments (*chacun ne dépasse pas les 65536 octets*) et pour basculer d'une partie à une autre du programme, il suffit de changer la valeur du registre **CS** et de cette manière on résout le problème des programmes qui ont une taille supérieure à 65536 octets.

**Le registre DS (Data segment) :** Le registre segment de données pointe sur le segment des variables globales du programme, bien évidemment la taille ne peut excéder 64K octets (*si on a des données qui dépassent cette limite, on utilise la même astuce citée dans la remarque précédente mais dans ce cas on change la valeur de DS*).

**Le registre ES (Extra segment) :** Le registre de données supplémentaires **ES** est utilisé par le microprocesseur lorsque l'accès aux autres registres est devenu difficile ou impossible pour modifier des données, de même ce segment est utilisé pour le stockage des chaînes de caractères.

**Le segment SS (Stack segment) :** Le registre **SS** pointe sur la pile : la pile est une zone mémoire où on peut sauvegarder les registres ou les adresses ou les données pour les récupérer après l'exécution d'un sous-programme ou l'exécution d'un programme d'interruption. En général, il est conseillé de ne pas changer le contenu de ce registre car on risque de perdre des informations très importantes (exemple les passages d'arguments entre le programme principal et le sous-programme).

### III.7.3 Le registre IP : (Le compteur de programme)

Il contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter. Le registre **IP** est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

### III.7.4 Le registre d'état (Flag)

Le registre d'état sert à contenir l'état de certaines opérations effectuées par le processeur. Par exemple, quand le résultat d'une opération est trop grand pour être



**DF (Direction Flag : Auto Incrémentation/Décrémentation)** : utilisé par les instructions de traitement des chaînes de caractères pour auto incrémenter ou auto décrémenter le **SI** et le **DI**.

**IF (Interrupt Flag : Masque d'interruption)** : pour masquer les interruptions venant de l'extérieur ce bit est mis à 0, dans le cas contraire ( $IF = 1$ ) le microprocesseur reconnaît l'interruption de l'extérieur.

**TF (Trap Flag : Piège)** : pour que le microprocesseur exécute le programme pas à pas.

**Remarque :**

- Les instructions de branchements conditionnels utilisent les indicateurs (drapeaux), qui sont des bits spéciaux positionnés par l'UAL après certaines opérations.
- Chaque indicateur est manipulé individuellement par des instructions spécifiques.
- Les bits *IF*, *DF* et *TF* sont des indicateurs de contrôle qui permettent de modifier le comportement du microprocesseur. Ils sont positionnés par le programmeur.
- X : bit non utilisé.

### III.8 Gestion de la mémoire

L'espace mémoire adressable de  $1\text{ Mo} = 2^{20}$  octets (20 bits du bus d'adresse) du 8086 est divisé en quatre segments logiques allant jusqu'à 64 K octets chacun. L'accès à ces espaces est direct et simultané. Or, le compteur programme est de 16 bits donc la possibilité d'adressage est de  $2^{16} = 64\text{ Ko}$  (Ce qui ne couvre pas la totalité de la mémoire), alors on utilise deux registres pour indiquer une adresse au processeur, Chaque segment débute à l'endroit spécifié par le registre segment. Le déplacement (*offset*) à l'intérieur de chaque segment se fait par un registre de décalage qui permet de trouver une information à l'intérieur du segment.

Exemple la paire de registres **CS: IP** : pointe sur le code d'une instruction (*CS registre segment et IP déplacement*).

Un segment est donc une zone mémoire de 64 Ko (65 536 octets) définie par son adresse de départ (*sur 20 bits*) qui doit être un multiple de 16. Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bits de poids fort, les 4 bits de poids faible étant implicitement à 0.

Pour désigner une case mémoire parmi les  $2^{16} = 65\,536$  cases contenues dans un segment, il suffit d'une valeur sur 16 bits. Ainsi, une case mémoire est repérée par le 8086 au moyen de deux quantités (*registres*) sur 16 bits :

- L'adresse d'un segment ;
- Un déplacement ou offset (*appelé aussi **adresse effective***) dans ce segment.

Cette méthode de gestion de la mémoire est appelée ***segmentation de la mémoire***.

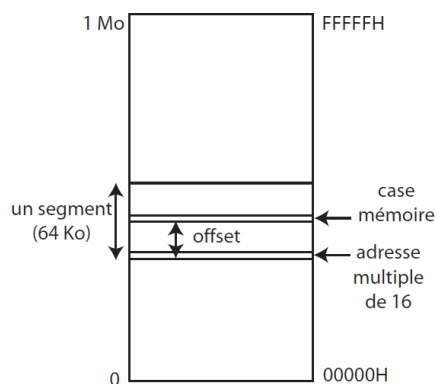


Figure III. 5 : Segmentation de la mémoire.

La donnée d'un couple (*segment, offset*) définit une ***adresse logique***, notée sous la forme ***segment : offset***.

L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 *digits hexadécimaux*) est appelée adresse physique car elle correspond à la valeur envoyée réellement sur le bus d'adresses  $A_0 - A_{19}$ .

La correspondance entre l'adresse logique et l'adresse physique est représentée sur la figure suivante :

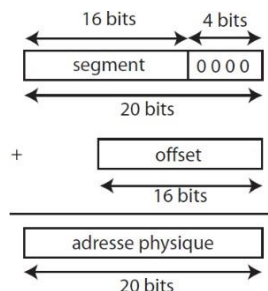


Figure III.6 : correspondance entre adresse logique et adresse physique.

Ainsi, l'adresse physique se calcule par l'expression :

$$\text{Adresse physique} = (16 \times \text{segment}) + \text{offset}$$

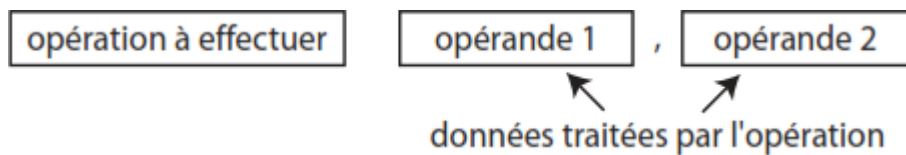
Car le fait d'injecter 4 zéros en poids faible de l'adresse du segment (*en binaire*) revient à effectuer un décalage de 4 positions (4 bits) vers la gauche, c'est à dire une multiplication par  $2^4 = 16$ .

A un instant donné, le 8086 a accès à 4 segments dont les adresses se trouvent dans les registres de segment *CS*, *DS*, *SS* et *ES*. Le segment de code contient les instructions du programme, le segment de données contient les données manipulées par le programme, le segment de pile contient la pile de sauvegarde et le segment supplémentaire peut aussi contenir des données.

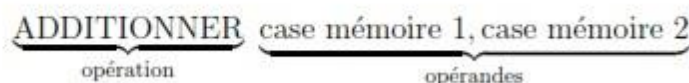
### III.9 Fonctionnement d'un microprocesseur

Un microprocesseur exécute un programme. Le programme est une suite d'instructions stockées dans la mémoire. Une instruction peut être codée sur un ou plusieurs octets.

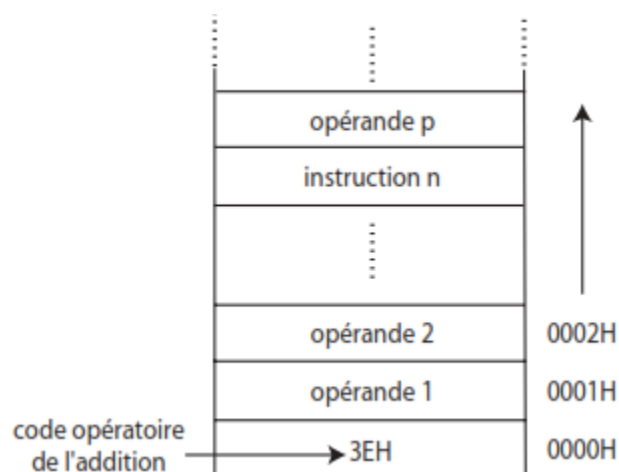
Format d'une instruction :



Exemple :

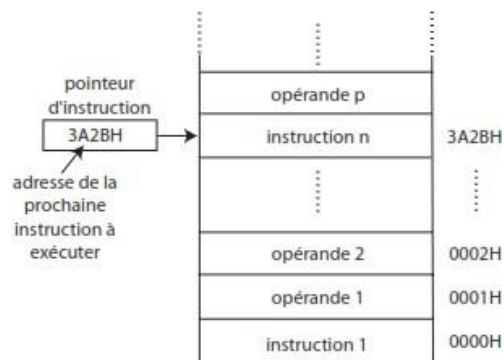


Rangement en mémoire:



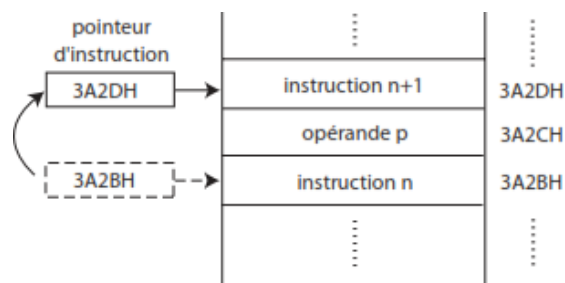
Pour exécuter les instructions dans l'ordre établi par le programme, le microprocesseur doit savoir à chaque instant l'adresse de la prochaine instruction à exécuter. Le microprocesseur utilise un registre contenant cette information. Ce registre est appelé pointeur d'instruction (IP : Instruction Pointer) ou compteur d'instructions ou compteur ordinal.

Exemple :



**Remarque :** la valeur initiale du pointeur d'instruction est fixée par le constructeur du microprocesseur. Elle vaut une valeur bien définie à chaque mise sous tension du microprocesseur ou bien lors d'une remise à zéro (reset). Pour savoir quel type d'opération doit être exécuté (addition, soustraction, ...), le microprocesseur lit le premier octet de l'instruction pointée par le pointeur d'instruction (code opératoire) et le range dans un registre appelé registre d'instruction. Le code opératoire est décodé par des circuits de décodage contenus dans le microprocesseur. Des signaux de commande pour l'UAL sont produits en fonction de l'opération demandée qui est alors exécutée.

Remarque : pour exécuter une instruction, l'UAL utilise des registres de travail, exemple : l'accumulateur, registre temporaire recevant des données intermédiaires. Pendant que l'instruction est décodée, le pointeur d'instruction est incrémenté de façon à pointer vers l'instruction suivante :



Puis le processus de lecture et de décodage des instructions recommence. A la suite de chaque instruction, un registre du microprocesseur est actualisé en fonction du dernier

résultat : c'est le registre d'état du microprocesseur. Chacun des bits du Registre d'état est un indicateur d'état ou flag (drapeau). Toutes ces étapes (lecture de l'instruction, d'encodage exécution) sont synchronisées par un séquenceur qui assure le bon déroulement des opérations.



## **Chapitre IV :**

### **La programmation en langage assembleur**

#### **IV.1 Introduction**

Une instruction est définie par son code opératoire, valeur numérique binaire difficile à manipuler par l'être humain. On utilise donc une notation symbolique pour représenter les instructions : les mnémoniques. Un programme constitué de mnémoniques est appelé programme en assembleur. Un programme en assembleur contient aussi des directives.

Pour programmer en assembleur, on doit utiliser, en plus des instructions assembleur, des directives. Une directive est une information que le programmeur fournit au compilateur, elle n'est pas transformée en une instruction en langage machine. Elle n'ajoute donc aucun octet au programme compilé. Donc les directives sont des déclarations qui vont guider l'assembleur.

Une directive est utilisée par exemple pour créer de l'espace mémoire pour des variables, pour définir des constantes, etc.

#### **IV.2 Syntaxe d'une instruction**

La syntaxe d'une instruction est comme suit :

`{Label :} Mnémonique {opérande} { ; commentaire}`

Le champ opérande est un champ optionnel selon l'instruction. Le champ commentaire c'est un champ sans signification syntaxique et sémantique pour l'assembleur. Le champ Label (étiquette) est destiné pour marquer une ligne qui sera la cible d'une instruction de saut ou de branchement. Une mnémonique est, généralement, composée uniquement de lettres (MOV, CMP, LOOP).

On pourra indifféremment écrire les mnémoniques avec des lettres minuscules ou majuscules.

#### **IV.3 Les groupes d'instructions**

- Les instructions arithmétiques.
- Les instructions logiques.
- Les instructions de décalage et de rotation.
- Les instructions de contrôle des indicateurs.
- Les instructions de comparaisons.
- Les instructions de branchements.
- Les instructions de boucles.
- Les instructions de traitement de chaînes de données

### IV.3.1 Les instructions de transfert de données.

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

Syntaxe : MOV destination, source

Opérandes autorisés : registre vers mémoire, registre vers registre, mémoire vers registre.

Le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire. Il existe différentes façons de spécifier l'emplacement d'un opérande dans une instruction : ce sont les

#### modes d'adressage

##### ✓ Mode immédiat

Un opérande est donné en immédiat s'il constitue lui-même la valeur. La donnée est spécifiée immédiatement après l'instruction, elle est donc située dans le segment de code.

Exemple: MOV SI, 1240h ; SI est chargé par la valeur immédiate 1240h (SI ← 1240h).

Remarque: Ce type de transfert est interdit avec les registres de segmentation (DS, CS, SS, ES).

##### ✓ Mode registre

L'opérande est désigné par le contenu d'un registre.

Exemple: MOV AX, DX

##### ✓ Mode direct

Un opérande est donné en mode direct s'il est désigné par son adresse effective dans la mémoire

Exemple 1: MOV [1200h],5Ah ;

La valeur 5Ah sera stockée à l'adresse effective 1200h. Pour calculer son adresse physique, il faut connaître la valeur stockée dans le registre de segmentation DS (Si DS = 720h), l'adresse physique sera :

Adresse physique =  $(720h \times 10h) + 1200h = 08400h$

Exemple 2:    **MOV [1200h], 4D59h**

4D59h est une donnée sur 16 bits, elle occupera les deux octets d'adresse effective 1200h et 1201h. Si DS = 720h, la répercussion de cette instruction sur la mémoire sera:

0720:1200----- 59

0720:1201 ----- 4D

Rappelons que 0720:1200 est la représentation (Segment : Offset) de l'adresse physique 08400h. Notez aussi la convention Little Endian dans le rangement en mémoire.

Remarque : Dans le cas de l'adressage direct de la mémoire, il faut indiquer le format de la donnée : octet ou mot (2 octets), car le microprocesseur 8086 peut manipuler des données sur 8 bits ou 16 bits. Pour cela, on doit utiliser un spécificateur de format :

**MOV byte ptr [1100H],65H**

Cette instruction transfère la valeur 65H (sur 1 octet) dans la case mémoire d'offset 1100H

**MOV word ptr [1100H],65H**

Cette instruction transfère la valeur 0065H (sur 2 octets) dans les cases mémoire d'offset 1100H et 1101H.

### ✓ **Mode indirect**

L'adresse effective est le contenu d'un registre de base (BX ou BP) ou index (DI ou SI) ou la combinaison des deux. Il existe trois types d'adressage indirect :

- Le mode indirect basé (registres de base),
- Le mode indirect indexé (registres index),
- Le mode indirect basé-indexé (registres de base et d'index).

Exemples :

MOV AX, [BX]

MOV AX, [SI]

MOV AX, [BX+DI]

Exemple 2:

Soit : BP = F415h ; SS = 0784h ; AX = 76E4h

MOV [BP] 7Ah, AL

Autre écriture : MOV [BP+7Ah], AL

La valeur 7Ah est un déplacement constant sur 8 bits. Ce déplacement est ajouté au contenu de BP pour former l'adresse effective.

- L'adresse effective :  $F415h + 7Ah = F48Fh$

- L'adresse physique :  $F48Fh + (10h \times 0784h) = 16CCFh$
- L'octet d'adresse 16CCFh recevra la valeur E4h (contenu d'AL).

Cet octet appartient évidemment à la pile (registre SS).

### ✓ **Mode indirect basé**

L'adresse effective de l'opérande est fournie par BX ou BP avec éventuellement un déplacement constant sur 8 ou 16 bits. Les registres de segment utilisés pour le calcul de l'adresse physique sont: DS pour BX et SS pour BP.

Ceci signifie que BX sera utilisé comme pointeur sur une donnée dans le segment de données (registre DS), Alors que BP sera employé pour accéder aux informations stockées dans la pile (registre SS).

Exemple 1: Soit BX = 1342h et DS = 1072h et soit l'instruction :

MOV [BX],0768h

### ✓ **Mode indirect indexé**

L'adresse effective est fournie par l'un des registres d'index SI ou DI avec éventuellement un déplacement constant sur 8 ou 16 bits. SI et DI sont des pointeurs utilisés dans la zone des données. Le registre de segment employé est DS.

Exemple 1 :

Soit SI = 24ABh ; DI = C140h ; DS = A1BAh

MOV [SI+1000h],142Bh

L'adresse effective :  $24ABh + 1000h = 34ABh$

L'adresse physique :  $34ABh + A1BA0h = A504Bh$

Exemple 2 :

Soit le programme suivant :

MOV AX,A1BAh

MOV DS,AX

MOV DI,C140h

MOV AX,1412h

MOV [DI],AX

Dans la dernière instruction où l'adressage indirect indexé est employé : L'adresse effective : C140h, L'adresse physique :  $A1BA0h + C140h = ADCE0h$

### ✓ Mode indirect basé-indexé

Dans ce mode, les modes indexé et basé sont combinés. L'adresse logique est donnée par BX/BP et SI/DI. Quatre cas sont possibles :

- Avec le registre BX (et le registre de segmentation DS) :

BX + SI + déplacement éventuel sur 8 ou 16 bits

BX + DI + déplacement éventuel sur 8 ou 16 bits

- Avec le registre BP (et le registre de segmentation SS) :

BP + SI + déplacement éventuel sur 8 ou 16 bits

BP + DI + déplacement éventuel sur 8 ou 16 bits

Ce type d'adressage est très utile dans le traitement des chaînes de caractères ou dans le cas de traitement simultané de tableaux. En effet, en faisant pointer BX sur le début d'une chaîne de caractères, SI peut être utilisé comme index pour accéder aux différents caractères.

#### Exemple :

Soit : SS = 1240h ; BP = 42A8h ; SI = 4010h

MOV [BP+SI],12h

Autre écriture : MOV [BP][SI],12h

L'adresse effective : BP + SI = 82B8h, L'adresse physique :  $10h \times SS + 82B8h = 1A6B8h$

L'octet d'adresse 1A6B8h sera chargé par la valeur 12h.

Exemple 2 : Ecrire un programme en assembleur 8086 qui permet de : Charger la valeur 100H dans le registre AX. Charger la valeur 205H dans le registre BX. Permuter les valeurs des registres AX et BX.

Solution 1:

```
mov ax,100h
mov bx,205H
mov cx,ax x
mov ax,bx
mov bx,cx
```

Solution 2:

```
mov ax,100h
mov bx,205H
chg ax,bx
```

Remarque : L'instruction XCHG (Exchange) permet de permuter deux valeurs.

Syntaxe : XCHG opérande 1, opérande 2

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre.

### IV.3.2 Les instructions arithmétiques

Les instructions arithmétiques de base sont l'addition, la soustraction, la multiplication et la division qui incluent diverses variantes.

#### ✓ ADD (Addition sans retenue)

Syntaxe : ADD opérande 1, opérande 2

L'opération effectuée est :  $\text{opérande1} \leftarrow \text{opérande1} + \text{opérande2}$ .

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF,ZF, AF, PF, CF

#### Remarque :

- Les deux opérandes doivent être de même taille (8 bits ou 16 bits).
- Tous les registres sont utilisables sauf les registres de segment.
- L'addition mémoire, mémoire est interdite.

#### Exemples :

- ADD DX, 1000h ; addition sur 16 bits :  $\text{DX} \leftarrow \text{DX} + 1000\text{h}$ .
- ADD BH, F1h; addition sur 8 bits:  $\text{BH} \leftarrow \text{BH} + \text{F1h}$ .
- ADD AL,[1205h] ; dressage direct.
- ADD CL,[SI] ; adressage indexé

#### ✓ ADC (Addition avec retenue)

Syntaxe : ADC opérande 1, opérande 2

L'opération effectuée est :  $\text{opérande1} \leftarrow \text{opérande1} + \text{opérande2} + \text{CF}$

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF,ZF, AF, PF, CF

#### Exemple :

ADC DX, 300h ; ( $\text{DX} \leftarrow \text{DX} + 300\text{h} + \text{CF}$ ), Si  $\text{DX} = 500\text{h}$  et  $\text{CF} = 1$  alors DX prendra la valeur 801h.

#### ✓ INC (Incrémentation)

Syntaxe : INC opérande 1

L'opération effectuée est :  $\text{opérande1} \leftarrow \text{opérande1} + 1$

Opérandes autorisés : registre et mémoire

Indicateurs affectés : OF, SF, ZF, AF, PF

Exemple: INC DI ; (DI←DI + 1)

### ✓ SUB (soustraction sans emprunt)

Syntaxe : SUB opérande 1, opérande 2

L'opération effectuée est : opérande1 ← opérande1 - opérande2

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF, ZF, AF, PF, CF

Exemple: SUB CH, BL ; (CH←CH-BL).

### ✓ SBB (soustraction avec emprunt)

Syntaxe : SBB opérande 1, opérande 2

L'opération effectuée est : opérande1 ← opérande1 - opérande2-CF

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF, ZF, AF, PF, CF

Exemple : SBB CH, BL ; (CH←CH-BL-CF).

### ✓ DEC (Décrémentation)

Syntaxe : DEC opérande 1

L'opération effectuée est : opérande1 ← opérande1 -1

Opérandes autorisés : registre et mémoire

Indicateurs affectés : OF, SF, ZF, AF, PF

**Exemple:** DEC DI ; (DI←DI - 1)

### ✓ MUL (Multiplication)

Syntaxe : MUL opérande 1

- Si opérande est sur 8 bits : AX ← AL x opérande.

- Si opérande est sur 16 bits : (DX : AX) ← AX x opérande

Opérandes autorisés : registre et mémoire

Indicateurs affectés : OF, SF, ZF, AF, PF, CF

Dans le cas d'une multiplication sur 16 bits les indicateurs ne sont pas affectés.

Exemples : MUL DH ; (AX←AL x DH)

### ✓ **IMUL (Multiplication signée)**

L'instruction IMUL fonctionne de la même façon que l'instruction MUL, mais sur des nombres signés.

#### **Exemple 1 :**

```
MOV BX, 435
MOV AX, 2372
IMUL BX
```

A l'issue de l'exécution de ces 3 instructions, AX contient la valeur BE8C et DX la valeur F, soit la valeur hexadécimale FBE8C, c'est-à-dire 1031820, le produit de 435 par 2372.

#### **Exemple 2 :**

```
MOV BX, -435
MOV AX, 2372
IMUL BX
```

A l'issue de l'exécution de ces 3 instructions, AX contient la valeur 4174 et DX la valeur FFF0, soit la valeur -1031820, le produit de -435 par 2372.

### ✓ **DIV et IDIV (Multiplication)**

Syntaxe : DIV opérande , IDIV opérande

L'opération effectuée est :

-Si opérande est sur 8 bits : AL = Quotient et AH = reste de (AX / opérande).

-Si opérande est sur 16 bits : AX = Quotient et DX = reste de (DX:AX) / opérande).

Opérandes autorisés : registre et mémoire

Indicateurs affectés:   DIV    OF, SF,ZF, PF, CF

                          IDIV   OF, SF, ZF, AF, PF, CF

#### **Exemple 1 :**

```
MOV BX, FFFEh
MOV AX, 2372H
```



MOV DX, 0172H

DIV BX

A l'issue de l'exécution de ces 4 instructions, AX contient la valeur 0172H et DX la valeur 2656H. Tous les opérandes sont considérés comme nombres positifs.

### **Exemple 2:**

MOV BX, FFFEh

MOV AX, 2372H

MOV DX, 0172H

IDIV BX

A l'issue de l'exécution de ces 4 instructions, AX contient la valeur 2372H et DX la valeur 0172H. Un opérande est considéré comme un nombre négatif si le bit du poids le plus fort est 1 (bit signe).

### **✓ NEG (Complément à 2**

Syntaxe : NEG opérande

L'opération effectuée est : opérande  $\leftarrow$   $\overline{\text{opérande}} + 1$

Opérandes autorisés : registre et mémoire

Indicateurs affectés : OF, SF, ZF, AF, PF

Exemple : mov al, 36h,  
neg al

### **Exercice 1**

Soit les nombres a, b et c stockés en mémoire tel que : a est stocké à l'adresse 100H. b est stocké à l'adresse 101H. c est stocké à l'adresse 102H.

Ecrire un programme en assembleur 8086 qui permet d'évaluer l'expression suivante :

$$d = a^2 / (2b + c).$$

Et stocker le résultat en mémoire à l'adresse 200H.

### **Solution :**

MOV AL, 2

MUL [0101H]

MOV BL,[0102H]

MOV BH,0

ADD BX,AX

MOV AL,[0100H]

```
MOV AH,0
MUL [0100H]
MOV DX,0
DIV BX
MOV [0200H], AX
MOV [0202H], DX
HLT
```

### IV.3.3 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, Complément à 1;

#### ✓ AND (ET logique)

Syntaxe : AND opérande 1, opérande 2

L'opération effectuée est : opérande1 ← opérande1 ET opérande2.

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF,ZF, PF, CF

#### Exemple :

```
mov al,10010110B → 10010110
mov bl,11001101B → 11001101
and al, bl        → 10000100
```

Application : masquage de bits pour mettre à zéro certains bits dans un mot.

#### ✓ OR (OU logique)

Syntaxe : OR opérande 1, opérande 2

L'opération effectuée est : opérande1 ← opérande1 OU opérande2.

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés : OF, SF,ZF, PF, CF

#### Exemple :

```
mov al,10010110B → 10010110
mov bl,11001101B → 11001101
or al, bl         → 11011111
```

Application: mise à 1 d'un ou plusieurs bits dans un mot.

### ✓ XOR (OU Exclusive)

Syntaxe : XOR opérande 1, opérande 2

L'opération effectuée est : opérande1  $\leftarrow$  opérande1  $\oplus$  opérande2.

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat.

Indicateurs affectés: OF, SF, ZF, PF, CF

#### Exemple :

mov al,10010110B  $\rightarrow$  10010110

mov bl,11001101B  $\rightarrow$  11001101

Or al, bl  $\rightarrow$  01011011

### ✓ NOT (Complément à 1)

Syntaxe : NOT opérande

L'opération effectuée est : opérande  $\leftarrow$  opérande

Opérandes autorisés : registre et mémoire

#### Exemple :

mov al, 36h

neg al

mov al, 36h

Not al

### IV.3.4 Les instructions de décalage et de rotation

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). Dans les rotations, les bits déplacés dans un sens sont réinjectés de l'autre côté du mot.

### ✓ SHR (Décalage vers la droite)

Syntaxe : SHR opérande,n

Cette instruction décale l'opérande de n positions vers la droite

Opérandes autorisés : registre et 1, mémoire et 1, registre et cl, mémoire et cl,

Remarque : si le nombre de bits à décaler est supérieur à 1, ce nombre doit être placé dans le registre CL.

### Exemple :

```
mov al,11001011B
```

```
shr al,1
```

Remarque : un décalage à droite est équivalent à une division par 2

### ✓ **SHL (Décalage vers la gauche)**

Syntaxe : SHL opérande,n

Cette instruction décale l'opérande de n positions vers la gauche

Opérandes autorisés : registre et 1, mémoire et 1, registre et cl, mémoire et cl,

Remarque : si le nombre de bits à décaler est supérieur à 1, ce nombre doit être placé dans le registre CL.

### Exemple :

```
mov al,11001011B
```

```
shl al, 1
```

Remarque : un décalage à gauche est équivalent à une multiplication par 2

### ✓ **SAR (Décalage arithmétique vers la droite)**

Syntaxe : SAR opérande

Cette instruction décale l'opérande de n positions vers la gauche, ce décalage conserve le bit de signe bien que celui-ci soit décalé

Opérandes autorisés : registre, mémoire

Indicateurs affectés : OF, SF,ZF, PF, CF

### **Exemple :**

```
mov al,11001011B
```

```
sar al,1
```

Remarque : le décalage arithmétique à gauche (SAL) est identique au décalage logique à gauche (SHL).

### ✓ **ROR (Rotation à droite)**

Syntaxe : ROR opérande

Cette instruction décale l'opérande de n positions vers la droite, les bits qui sortent par la droite sont injectés par la gauche.

Opérandes autorisés : registre et 1, mémoire et 1, registre et cl, mémoire et cl

Indicateurs affectés : OF, CF

Remarques:

- 1-Si le nombre de bits à décaler est supérieur à 1, ce nombre doit être placé dans le registre CL.
- 2- La rotation à gauche (ROL), se réalise comme la rotation à droite, mais dans l'autre sens: Le bit qui sort à gauche est injecté à la droite.

### ✓ **RCR (Rotation à droite avec passage par l'indicateur de retenue)**

Syntaxe : RCR opérande

Cette instruction décale l'opérande de n positions vers la droite en passant par l'indicateur de retenue CF. Le bit sortant par la droite est copié dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la gauche

Opérandes autorisés : registre et 1, mémoire et 1, registre et cl, mémoire et cl

Indicateurs affectés : OF, CF

Remarque : L'instruction RCL fonctionne de la même façon, mais dans l'autre sens

### **IV.3.5 Les instructions de pile**

Une pile est une zone mémoire servant à stocker temporairement des valeurs. On ne peut stocker qu'une information à la fois et l'élément dépilé à un moment donné est celui qui a été empilé en dernier : c'est la structure LIFO (Last In, First Out). Les opérations ne se font que sur 16 bits. La pile commence au segment SS et elle finit dans le même segment mais à l'offset SP.

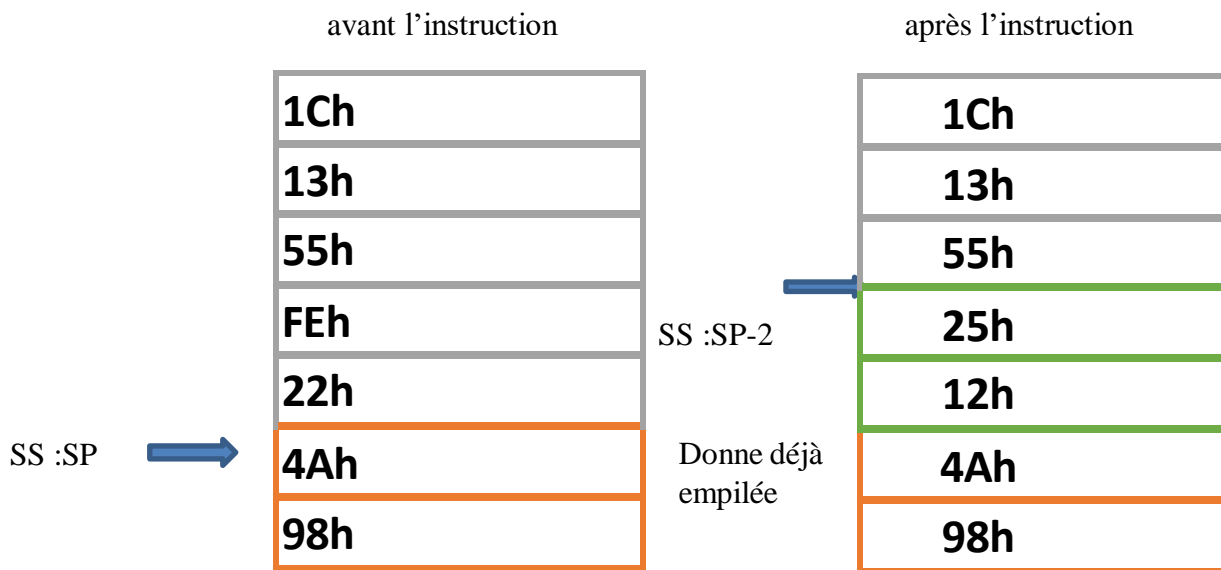
A noter que cette pile est remplie à l'envers : le premier élément est situé à une adresse plus haute que le dernier élément.

#### **IV.3.5.1 L'instruction d'empilement (push)**

L'empilement d'une donnée est réalisé par l'instruction : **PUSH donnée**, où donnée est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect. L'empilement s'effectue en deux étapes :

- 1- $SP \leftarrow SP - 2$  : on fait reculer SP de deux cases (donnée sur 16 bits), pour qu'il pointe sur le nouveau sommet susceptible de recevoir la donnée à empiler.
- 2- $[SP] \leftarrow \text{donnée}$  : le sommet de la pile reçoit la donnée.

Exemple : Soit AX = 1225h et soit l'instruction : PUSH AX



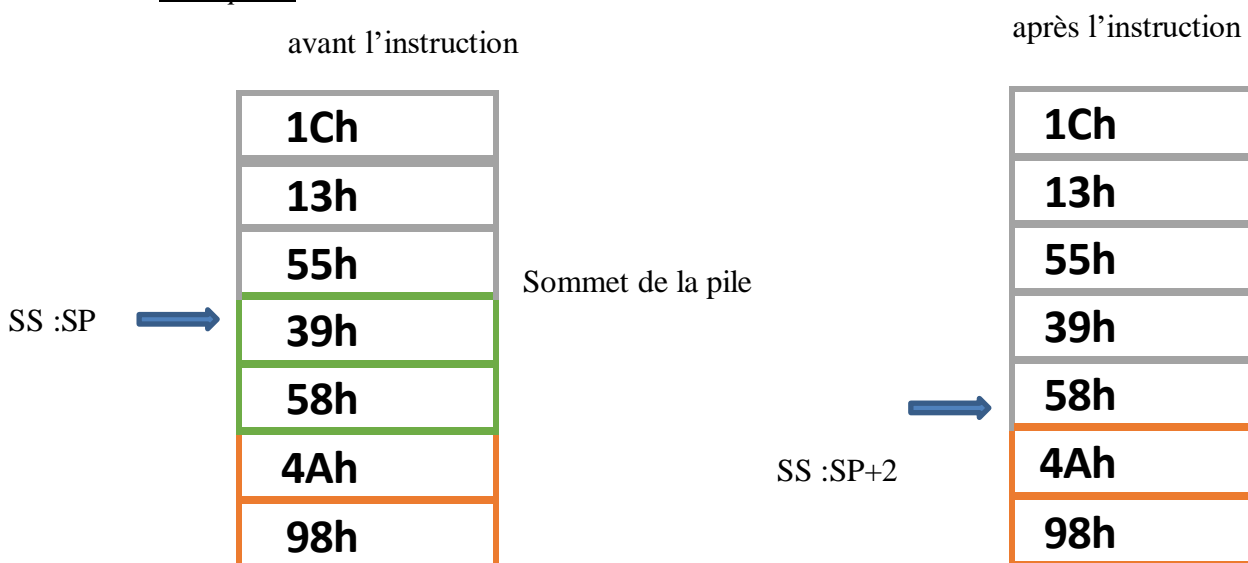
#### IV.3.5.2 L'instruction d'empilement (push)

Le dépilement est l'opération qui permet de retirer le sommet de la pile et le stocker dans une destination, il est réalisé par l'instruction : **POP destination**

La destination est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect. Le dépilement engendre deux opérations :

- destination  $\leftarrow$  [SP] : le sommet de la pile est copié dans destination.
- SP  $\leftarrow$  SP + 2 : la donnée du sommet étant retirée, on fait avancer SP pour pointer sur la donnée suivante.

Exemple : Considérons l'instruction : POP DX



Le registre DX est chargé par la valeur 5839h

### IV.3.5.3 Autres instruction de la pile

- **PUSHA (Push All):** Empile les 8 registres généraux. Les registres sont empilés dans l'ordre AX, CX, DX, BX, SP, BP, SI, DI. La valeur empilée de SP est celle avant le début de l'instruction.
- **POPA (Pop All):** Dépile les 16 octets du sommet de la pile vers les 8 registres généraux. Les registres sont dépilés dans l'ordre DI, SI, BP, SP, BX, DX, CX, AX. En fait, le registre SP pose un cas particulier, puisqu'il n'est pas mis à jour avec la valeur dépilée qui n'est pas prise en compte.
- **PUSHF (Push Flags):** Empile le registre des indicateurs.
- **POPF (Pop Flags):** Dépile vers le registre des indicateurs

### IV.3.6 Les instructions de branchement

Les instructions de branchement (ou saut) permettent de modifier l'ordre d'exécution des instructions du programme en fonction de certaines conditions.

Il existe 3 types de saut : saut inconditionnel, saut conditionnel, appel de sous-programmes

#### ✓ **CMP (Comparaison)**

Syntaxe : CMP opérande1, opérande2

Soustrait l'opérande 2 de l'opérande 1. Positionne les indicateurs en fonction du résultat qui n'est pas stocké.

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat

Indicateurs affectés: OF, SF, ZF, PF, and CF

#### ✓ **TEST (Comparaison logique)**

Syntaxe : TEST opérande1, opérande2

Teste les bits de l'opérande 1 spécifiés par l'opérande 2, cette instruction effectue un ET logique, bit à bit, entre les deux opérandes, positionne les indicateurs, mais le résultat n'est pas stocké

Opérandes autorisés : registre et mémoire ; registre et registre ; mémoire et registre ;  
Registre / mémoire et immédiat

Indicateurs affectés : OF, SF, ZF, PF, CF

### ✓ **JMP (saut inconditionnel)**

Syntaxe : JMP label

Cette instruction effectue un saut (jump) vers le label spécifié. Un label(ou étiquette) est une représentation symbolique de l'adresse d'une instruction en mémoire. L'instruction JMP ajoute au registre IP (pointeur d'instruction) le nombre d'octets (distance) qui sépare l'instruction de saut de sa destination. Pour un saut en arrière, la distance est négative (codée en complément à 2).

### ✓ **Les sauts conditionnels**

Syntaxe : JC condition label

Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite, sinon l'exécution se poursuit séquentiellement à l'instruction suivante. La condition du saut porte sur l'état de l'un (ou plusieurs) des indicateurs d'état (flags) du microprocesseur. Les indicateurs sont positionnés en fonction du résultat de la dernière opération. Il existe des sauts arithmétiques. Ils suivent en général l'instruction de comparaison.

<i>Mnémonique</i>	<i>Signification</i>	Mnémonique
<i>JC</i>	<i>Jump If Carry</i>	JC
<i>JNC</i>	<i>Jump If Not Carry</i>	JNC
<i>JZ</i>	<i>Jump If Zero</i>	JZ
<i>JNZ</i>	<i>Jump If Not Zero</i>	JNZ
<i>JP</i>	<i>Jump If Parity</i>	JP
<i>JNP</i>	<i>Jump If Not Parity</i>	JNP
<i>JPE</i>	<i>Jump If Parity Even</i>	JPE
<i>JPO</i>	<i>Jump If Parity Odd</i>	JPO
<i>JO</i>	<i>Jump If Overflow</i>	JO



<i>JNO</i>	<i>Jump If Not Overflow</i>	JNO
<i>JS</i>	<i>Jump If Sign</i>	JS
<i>JNS</i>	<i>Jump If Not Sign</i>	JNS
JCXZ	Jump If CX Register is Zero	(CF ou ZF) = 0
JNA	Jump If Not Above	CF =1 ou ZF = 1
JNB	Jump If Not Below	CF = 1
JNAE	Jump If Not Above nor Equal	CF = 1
JNBE	Jump If Not Below nor Equal	CF = 0 et ZF = 0
JE	Jump If Equal	ZF = 1
JNE	Jump If Not Equal	ZF = 0
JCXZ	Jump If CX Register is Zero	(CF ou ZF) = 0
JNA	Jump If Not Above	CF =1 ou ZF = 1
JNB	Jump If Not Below	CF = 1
JA	Jump If Above	CF = 0 et ZF = 0
JAЕ	Jump If Above or Equal	CF = 0
JB	Jump If Below	CF = 1
JBE	Jump If Below or Equal	CF = 0 ou ZF = 0
JG	Jump If Greater	ZF = 0 et SF = 0
JGE	Jump If Greater or Equal	SF = 0
JNG	Jump If Not Greater	((SF xor OF) ou ZF) = 1

## Chapitre VI :La programmation en langage assembleur

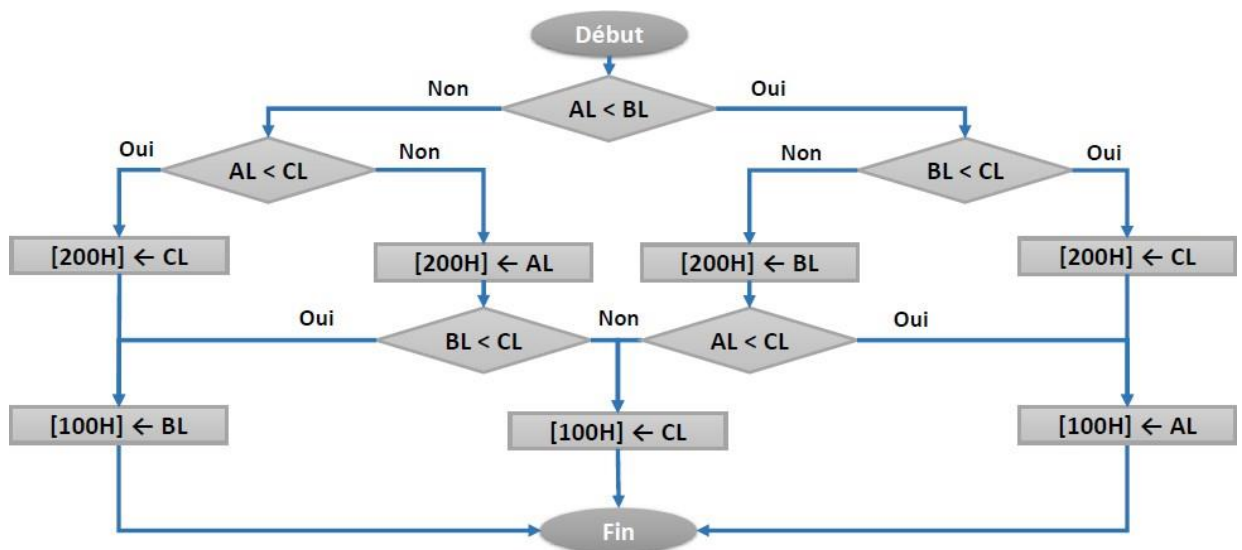
---

JNGE	Jump If Not Greater nor Equal	$(SF \text{ xor } OF) = 1$
JL	Jump If Less	$(SF \text{ xor } OF) = 1$
JLE	Jump If Less or Equal	$((SF \text{ xor } OF) \text{ ou } ZF) = 1$
JNL	Jump If Not Less	$SF = OF$
JNLE	Jump If Not Less nor Equal	$ZF = 0 \text{ et } SF = OF$
JA	Jump If Above	$CF = 0 \text{ et } ZF = 0$
JAE	Jump If Above or Equal	$CF = 0$
JB	Jump If Below	$CF = 1$
JBE	Jump If Below or Equal	$CF = 0 \text{ ou } ZF = 0$
JG	Jump If Greater	$ZF = 0 \text{ et } SF = 0$
JGE	Jump If Greater or Equal	$SF = 0$
JNG	Jump If Not Greater	$((SF \text{ xor } OF) \text{ ou } ZF) = 1$
JNGE	Jump If Not Greater nor Equal	$(SF \text{ xor } OF) = 1$
JL	Jump If Less	$(SF \text{ xor } OF) = 1$
JLE	Jump If Less or Equal	$((SF \text{ xor } OF) \text{ ou } ZF) = 1$
JNL	Jump If Not Less	$SF = OF$
JNLE	Jump If Not Less nor Equal	$ZF = 0 \text{ et } SF = OF$
JA	Jump If Above	$CF = 0 \text{ et } ZF = 0$
JAE	Jump If Above or Equal	$CF = 0$
JB	Jump If Below	$CF = 1$
JBE	Jump If Below or Equal	$CF = 0 \text{ ou } ZF = 0$

### Exercice 2:

Ecrire un programme en Assembleur 8086 qui permet de : Comparer les valeurs contenues dans les registres AL, BL et CL pour trouver le minimum et le maximum de ces trois valeurs. Stocker le minimum en mémoire à l'adresse 100H. Stocker le maximum en mémoire à l'adresse 200H.

### Solution :



### Programme :

```
CMP AL,BL
JB etq1
CMP AL,CL
JB etq2
MOV [200H],AL
CMP BL,CL
JB etq3
etq4: MOV [100H],CL
      JMP Fin
etq2: MOV [200H],CL
etq3: MOV [100H],BL
      JMP Fin
```

```
etq1:  CMP BL,CL
      JB etq5
      MOV [200H],BL
      CMP AL,CL
      JB etq6
      JMP etq4
etq5:  MOV [200H],CL
etq6:  MOV [100H],AL
Fin:   HLT
```

### IV.3.7 Les instructions de boucles

Il n'existe pas de structure générale en assembleur pour coder une boucle tant-que ou une boucle répéter. Cependant, à l'aide des instructions vues précédemment pour réaliser des tests, on peut coder n'importe quel type de boucle. On verra également que des instructions simplifient grandement le codage des boucles pour (boucle for).

Une boucle est essentiellement composée de deux éléments :

- 1- Une condition de sortie qui indique quand la boucle doit être interrompue, qu'il faut sortir de la boucle et continuer l'exécution des instructions en séquence.
- 2- Un corps de boucle qui spécifie l'action à réaliser pendant que la condition de sortie n'est pas vérifiée, à chaque itération de la boucle. La condition de sortie va donc être codée d'une manière ressemblant aux tests, un saut conditionnel testera cette condition et entraînera, quand la condition est vérifiée la sortie de la boucle. Le corps de la boucle devra pour sa part (boucler), c'est-à-dire, une fois exécuté, entraîner la réévaluation de la condition de sortie, et la prise de décision quant à la poursuite ou non des itérations.

#### ✓ *boucle tant-que*

Le squelette d'une, dans un langage de haut niveau, est le suivant :

```
TANT-QUE (condition) FAIRE
    action
FIN_TQ
```

Cela va se traduire en assembleur sous la forme suivante :

```
TQn: calcul de la condition
Jcc FTQn
```

action

...

JMP TQn

FTQn:

...

### ✓ *boucle répète*

Le squelette d'une r, dans un langage de haut niveau, est le suivant :

REPETER

action

JUSQUA (condition vraie)

Cela va se traduire en assembleur sous la forme suivante :

REPETERn: action

...

Calcul de la condition

Jcc REPETERn

### ✓ *boucle pour*

Une boucle pour est généralement codée à l'aide d'une instruction de la famille LOOP.

Syntaxe : LOOP label

LOOP fonctionne avec le registre CX qui joue le rôle de compteur de boucles. LOOP décrémente le compteur sans modifier aucun des indicateurs. Si le compteur est différent de 0, un saut à l'étiquette opérande de l'instruction LOOP est réalisé.

Le squelette d'une boucle pour s'écrit de la manière suivante :

POUR indice = 0 à bs FAIRE

Action

Fin pour

Cependant, en assembleur, seules existent des boucles ayant un indice qui est décrémente à chaque itération. Aussi, la boucle précédente doit initialement être transformée en une boucle du type suivant :

POUR indice = bs à 0, pas := -1 FAIRE

action

Fin pour

Cette boucle se traduit en assembleur de la manière suivante :

mov cx, n ; n est le nombre d'itérations

POURn: ...  
action  
loop POURn  
...

### Exemple :

Ecrire un programme en assembleur qui permet de transférer les 100 octets de données situés à l'adresse 3000H vers l'adresse 4000H.

MOV SI, 3000 H ; charger l'adresse de source dans SI  
MOV DI, 4000 H ; charger l'adresse de destination dans DI  
MOV CX, 64 H ; initialiser CX à 64h (100)

Label: MOV AH, [SI] ; charger l'octet source dans AH  
MOV [DI], AH ; charger AH (l'octet source) dans la destination  
INC SI ; incrémenter SI pour pointer le prochain octet source  
INC DI ; incrémenter DI pour pointer la prochaine destination  
DEC CX ; décrémenter le compteur de boucle  
JNZ Label ; sauter vers Label  
HLT ; fin du programme

- Le programme précédent peut être écrit de la manière suivante :

MOV SI, 3000 H ; charger l'adresse de source dans SI  
MOV DI, 4000 H ; charger l'adresse de destination dans DI  
MOV CX, 64 H ; initialiser CX à 64h (100)

Label: MOV AH, [SI] ; charger l'octet source dans AH  
MOV [DI], AH ; charger AH (l'octet source) dans la destination  
INC SI ; incrémenter SI pour pointer le prochain octet source  
INC DI ; incrémenter DI pour pointer la prochaine destination  
LOOP Label ; boucler tant que CX  $\neq$  0  
HLT ; fin du programme.

### ✓ *boucle LOOPE (loop while equal)*

Cette instruction a la même fonction que l'instruction LOOP. Elle décrémente le compteur de boucle CX de 1, et compare CX avec 0, si ce n'est pas égale, elle saute au label spécifié, mais, de plus, elle ne saute au label que si l'indicateur ZF est égal à 1, donc si la dernière opération effectuée a positionné cet indicateur.

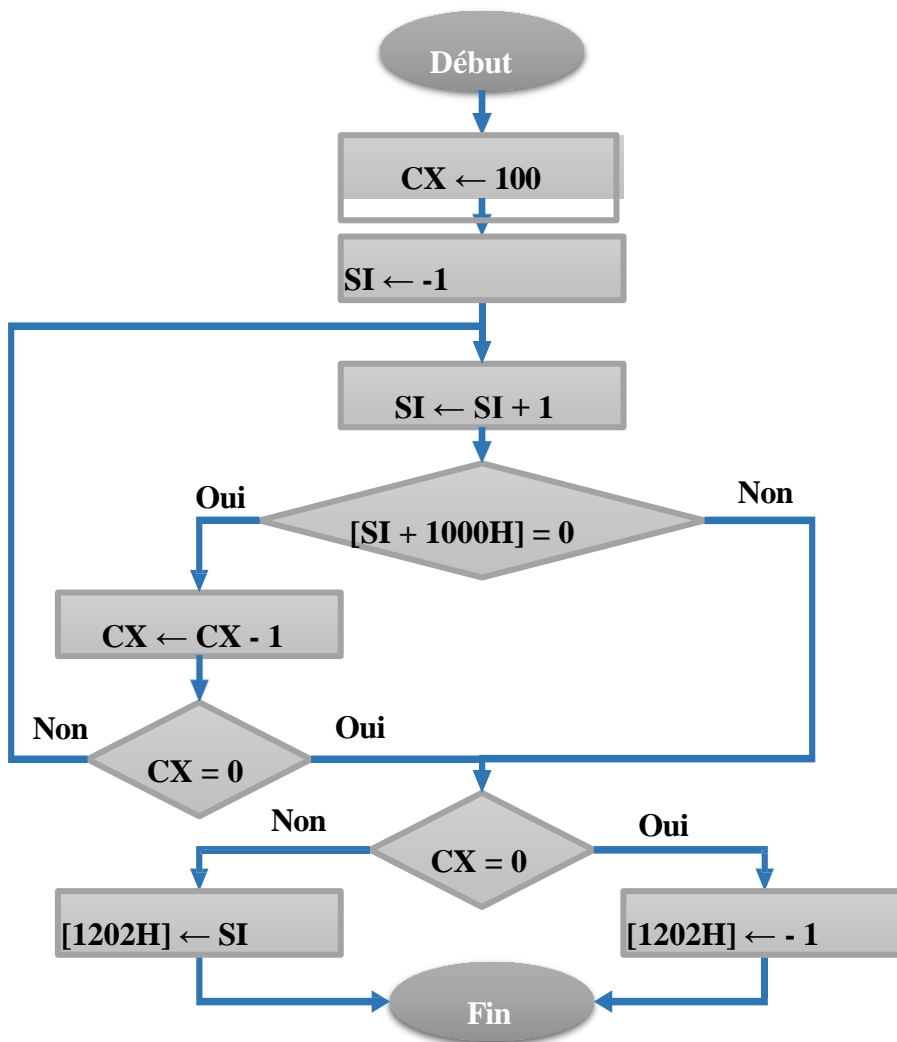
Remarque :

L'instruction LOOPE est équivalente à l'instruction LOOPZ.

### Exercice 3:

Recherche d'un premier élément non nul dans un tableau de 100 octets stocké en mémoire à l'adresse 1000H. Stocker l'indice de cet élément en mémoire à l'adresse 1202H, si aucun élément n'est trouvé, stocker -1 à cette adresse.

### L'organigramme



### Le programme :

```
MOV CX,100
```

```
MOV SI,-1
```

```
Boucle : INC SI
```

```
CMP [1000H + SI],0
LOOPE Boucle CMP CX,0
JNE Trouvé
MOV [1202H],-1
JMP Fin
```

Trouvé: MOV [1202H], SI

Fin: HLT

### ✓ *Boucle LOOPNE (loop while Not equal)*

Cette instruction décrémente CX et effectue un saut à l'adresse si CX  $\neq$  0 et ZF = 0.

#### Remarque :

L'instruction LOOPNE est équivalente à l'instruction LOOPNZ.

#### **Exemple :**

Recherche d'un premier élément nul dans un tableau de 100 octets stocké en mémoire à l'adresse 1000H. Stocker l'indice de cet élément en mémoire à l'adresse 1202H. Si aucun élément n'est trouvé, stocker -1 à cette adresse

#### Le programme :

```
MOV CX, 100
MOV SI, -1
```

Boucle: INC SI

```
CMP [1000H + SI],0
```

```
LOOPNE Boucle
```

```
CMP CX, 0
```

```
JNE Trouvé
```

```
MOV [1202H],-1
```

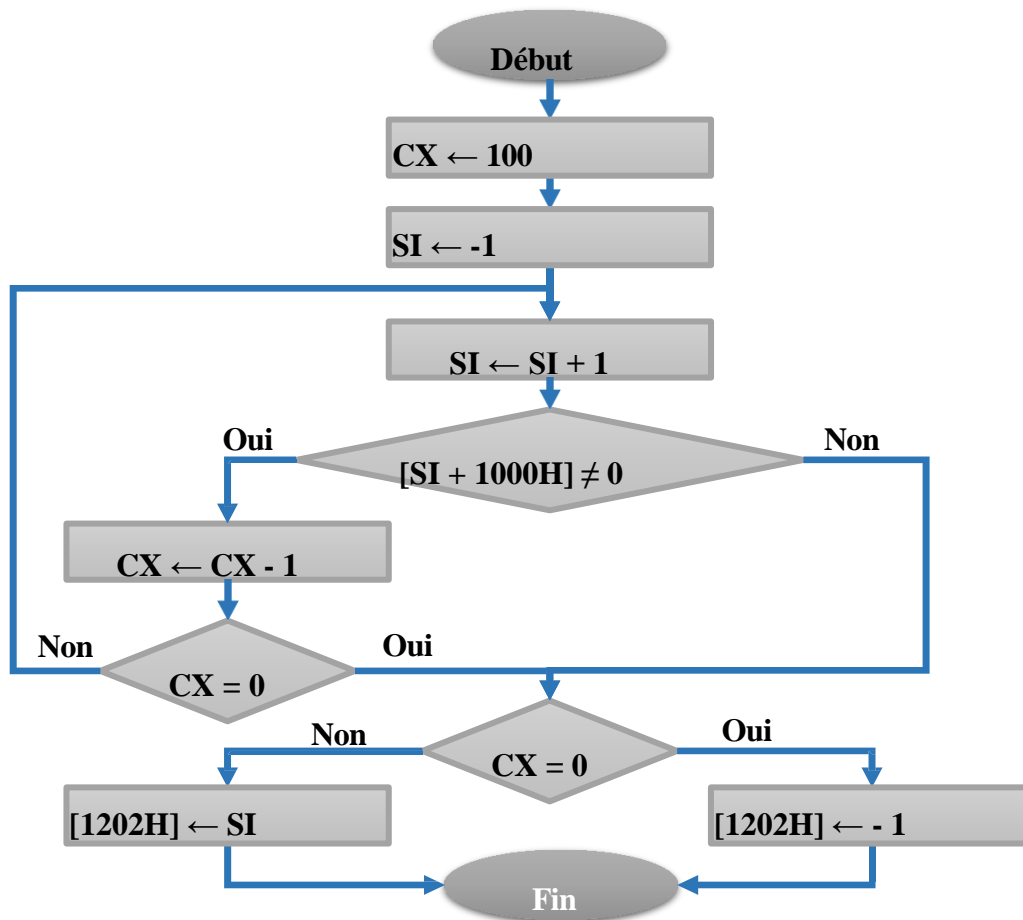
```
JMP Fin
```

Trouvé: MOV [1202H], SI

Fin: HLT

#### L'organigramme

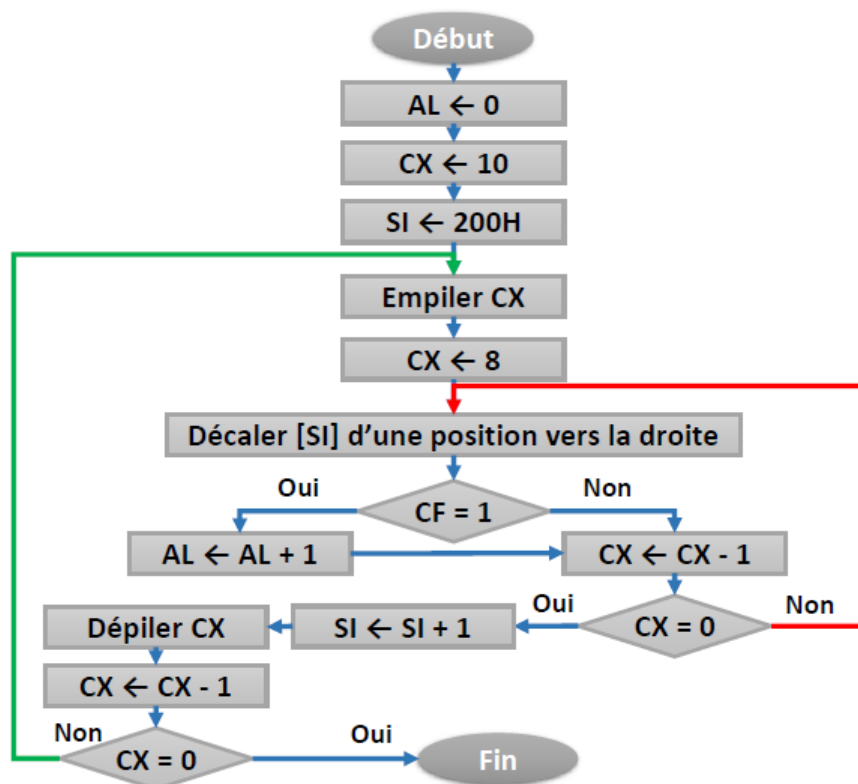




#### **Exercice 4 :**

Ecrire un programme en Assembleur 8086 qui permet de compter le nombre de bits 1 dans un tableau de 10 octets, se trouvant en mémoire à l'adresse 200H. Le résultat sera stocké dans un registre.

#### **Solution :**



```

MOV AL, 0
MOV CX, 10
MOV SI, 200H
B1:  PUSH CX
      MOV CX, 8
B2:  SHR  [SI], 1
      JNC NEXT
      INC AL
NEXT: LOOP B2
      INC SI
      POP CX
      LOOP B1
      HLT
  
```

### IV.4 Déclaration de variables en assembleur

On déclare les variables à l'aide de directives. L'assembleur attribue à chaque variable une adresse. Dans le programme, on repère les variables grâce à leurs noms, les noms des variables (comme les étiquettes) sont composés d'une suite de 31 caractères au maximum, commençant obligatoirement par une lettre. Le nom peut comporter des majuscules, des minuscules, des chiffres, plus les caractères @, ? et \_.

Lors de la déclaration d'une variable, on peut lui affecter une valeur initiale, les directives DB (Define Byte) et DW (Define Word) permettent de déclarer des variables de respectivement 1 ou 2 octets.

#### Exemple :

truc DW 0F0AH ; 2 octets initialisés à 0F0A en hexadécimal.

masque DB 01110000b ; 1 octet initialisé à 01110000 en binaire.

Les variables s'utilisent dans le programme en les désignant par leurs noms, après la déclaration précédente, on peut écrire par exemple :

```
MOV AX, truc
```

```
AND AL, masque
```

```
MOV truc, AX
```

L'assembleur se charge de remplacer les noms de variables par les adresses correspondantes. Il est aussi possible de déclarer des tableaux, c'est à dire des suites d'octets ou de mots consécutifs. Pour cela, on utilise plusieurs valeurs initiales :

machin db 10, 0FH ; 2 fois 1 octet

chose db -2, 'ALORS' ; un nombre et une chaîne de caractères.

Si l'on veut écrire un caractère X à la place de l'O de ALORS, on pourra écrire :

```
MOV AL, 'X'
```

```
MOV chose + 3, AL
```

Lorsque l'on veut déclarer un tableau de n cases, toutes initialisées à la même valeur, on utilise la directive DUP :

tab DB 100 dup (15) ; 100 octets valant 15

zzz DW 10 dup (?) ; 10 mots de 16 bits non initialisés.

La directive EQU associe une valeur à un symbole qui pourra être ensuite utilisé dans le programme à la place de la constante qu'elle définit.

nom EQU constante

Par exemple, la commande : Trois EQU 3 définit une constante qui s'appelle Trois, dont la valeur est 3. Une fois cette déclaration effectuée, toute occurrence de l'identificateur Trois sera remplacée par la valeur indiquée.

Nous décrivons ici quelques instructions ne prenant aucun opérande et agissant chacune sur un bit particulier du registre d'indicateurs FLAGS.

- CLC (Clear Carry Flag) mise à zéro de l'indicateur de retenue CF.
- CLD (Clear Direction Flag) mise à zéro de l'indicateur de direction DF.
- CLI (Clear Interrupt Flag) mise à zéro du masque d'interruption IF.
- CMC (Complement Carry Flag) complémente l'indicateur CF.
- STC (Set Carry Flag) mise à 1 de l'indicateur CF.
- STD (Set Direction Flag) mise à 1 de l'indicateur DF.
- STI (Set Interrupt Flag) mise à 1 du masque d'interruption IF.

### IV.5 Manipulation de chaînes de données

Des instructions spécialement sont conçues pour le traitement de séquences de données,

Les données peuvent être des caractères (des octets) ou des mots.

Il y a cinq types d'instruction :

LODS : chargement d'un élément de chaîne depuis la mémoire dans un registre.

STOS : écriture d'un élément de chaîne en mémoire.

MOVS : transfert d'un élément entre deux chaînes.

CMPS : comparaison entre deux éléments de chaîne.

SCAS : comparaison entre une valeur et un élément de chaîne.

#### ✓ L'instruction REP (Repeat)

Cette instruction permet d'itérer sur toute une chaîne les opérations élémentaires vues précédemment.

- Cette instruction préfixe en fait l'opération à itérer.
- Elle donne tout son intérêt aux instructions de manipulation des chaînes de données.
- L'opérande de cette instruction est une instruction.
- Le processeur effectue cette instruction, et décrémente le registre

CX, jusqu'à ce que celui-ci soit nul.

- Il faut donc placer préalablement 'le nombre de fois à répéter l'instruction dans le registre CX'.

Il existe plusieurs variantes de l'instruction REP.

### ✓ **LODS (Load String Data )**

Syntaxe : LODSB ou LODSW

1- L'opérande est un octet : LODSB (Load String Byte)

LODSB: charge AL avec l'octet pointé par DS:SI, puis met à jour le contenu de SI en fonction de l'indicateur DF (DF = 0: SI est incrémenté, DF = 1 : SI est décrémenté).

2- L'opérande est un mot de 16 bits : LODSW (Load String Word)

LODSW: charge AX avec le mot pointé par DS:SI, puis met à jour le contenu de SI en fonction de l'indicateur DF (DF = 0 : SI est doublement incrémenté, DF =1: SI est doublement décrémenté).

### ✓ **STOS (Store String Data )**

Syntaxe : STOSB ou STOSW

1- L'opérande est un octet : STOSB (Store String Byte)

STOSB: charge l'octet pointé par ES:DI avec le contenu de AL, puis met à jour le contenu de DI en fonction de l'indicateur DF.

2- L'opérande est un mot de 16 bits : STOSW (Store String Word).

STOSW: charge le mot de 16 bits pointé par ES:DI avec le contenu de AX, puis met à jour le contenu de DI en fonction de l'indicateur DF.

### ✓ **MOVS (Move String Data)**

Syntaxe : MOVSB ou MOVSW

1- L'opérande est un octet : MOVSB (Move String Byte)

MOVSB: copie un octet de l'adresse DS:SI à l'adresse ES:DI, SI et DI sont ensuite incrémentés de 1 (si DF=0) ou décrémentés de 1 (si DF=1).

2- L'opérande est un mot de 16 bits : MOVSW (Move String Word)

MOVSW: copie un mot de l'adresse DS:SI à l'adresse ES:DI, SI et DI sont ensuite incrémentés de 2 (si DF=0) ou décrémentés de 2 (si DF=1).

### ✓ **CMPS (Compare String Data)**

Syntaxe : CMPS

1- L'opérande est un octet : CMPSB (Compare String Byte)

CMPSB compare deux chaînes formées d'une suite d'octets.

### 2-L'opérande est un mot de 16 bits : CMPSW (Compare String Word)

CMPSW compare deux chaînes formées d'une suite de mots. DS:SI doivent contenir l'adresse de la chaîne source, et ES:DI celle de la chaîne destination. Pour chaque octet des chaînes à comparer, une soustraction est effectuée dont le résultat est utilisé pour mettre à jour les indicateurs, puis les registres SI et DI sont incrémentés ou décrémentés selon la valeur de DF.

#### ✓ **SCAS (Scan String Data)**

Syntaxe : SCAS

#### 1-L'opérande est un octet : SCASB (Scan String Byte)

SCASB recherche dans une chaîne l'octet spécifié par le contenu de AL.

#### 2-L'opérande est un mot de 16 bits : SCASW (Scan String Word)

SCASW recherche dans une chaîne le mot spécifié par le contenu de AX. ES:DI contient l'adresse de la chaîne analysée. Pour chaque octet ou mot de la chaîne une soustraction est effectuée avec le caractère recherché, dont le résultat est utilisé pour mettre à jour les indicateurs, puis DI est incrémenté ou décrémenté suivant la valeur DF.

#### ✓ **REP (Repeat String)**

Syntaxe : REP MOVS ou REP STOS

Préfixe de répétition de chaîne, l'instruction préfixée est répétée le nombre de fois spécifié par le contenu du registre CX. Pour chaque élément de la chaîne, l'instruction est exécutée et le contenu de CX est décrémenté. Lorsque le contenu de CX atteint 0, l'exécution continue à l'instruction suivante. En principe REP est utilisé avec l'une des instructions MOVS ou STOS.

#### **Exemple**

Ecrire un programme en assembleur 8086 qui copie la chaîne Msg1 dans Msg2 (avec Msg1 = Hello).

```
Msg1 DB 'Hello'
Msg2 DB ?
MOV SI, OFFSET Msg1
MOV DI, OFFSET Msg2
MOV CX, 5
REP MOVSB
HLT
```

### ✓ **REPE (Repeat String while Equal) , REPZ (Repeat String while Zero)**

Syntaxe : REPE CMPS / REPZ CMPS ou REPE SCAS / REPZ SCAS

Répéter tant que  $ZF = 1$ , l'instruction préfixée est répétée tant que la condition est vraie et au maximum le nombre de fois spécifié par le contenu du registre CX. Pour chaque élément de la chaîne, l'instruction est exécutée et le contenu de CX est décrémenté. Lorsque le contenu de CX atteint 0, ou lorsque la condition est fausse, l'exécution continue à l'instruction suivante.

REPE / REPZ ne peut être utilisée qu'avec l'une des instructions SCAS ou CMPS : ce sont les seules instructions de chaîne modifiant l'indicateur ZF.

### ✓ **REPNE (Repeat String while Not Equal) REPNZ (Repeat String while Not Zero)**

Syntaxe : REPNE CMPS / REPNZ CMPS ou REPNE SCAS / REPNZ SCAS

Répéter tant que  $ZF = 0$ , l'instruction préfixée est répétée tant que la condition est vraie et au maximum le nombre de fois spécifié par le contenu du registre CX. Pour chaque élément de la chaîne, l'instruction est exécutée et le contenu de CX est décrémenté. Lorsque le contenu de CX atteint 0, ou lorsque la condition est fausse, l'exécution continue à l'instruction suivante.

REPNE / REPNZ ne peut être utilisée qu'avec l'une des instructions SCAS ou CMPS : ce sont les seules instructions de chaîne modifiant l'indicateur ZF.

## Chapitre V

### Les interfaces d'entrées/sorties

#### V.1. Types de liaisons

Les systèmes à microprocesseur utilisent deux types de liaisons différentes pour se connecter à des périphériques : Liaison parallèle et liaison série. On caractérise un type de liaison par sa vitesse de transmission ou débit (en bit/s).

##### V.1.1 Liaison parallèle

Dans ce type de liaison, tous les bits d'un mot sont transmis simultanément. Ce type de transmission permet des transferts rapides mais reste limitée à de faibles distances de transmission à cause du nombre important de lignes nécessaires (coût et encombrement) et des problèmes d'interférence électromagnétique entre chaque ligne (fiabilité). La transmission est cadencée par une horloge

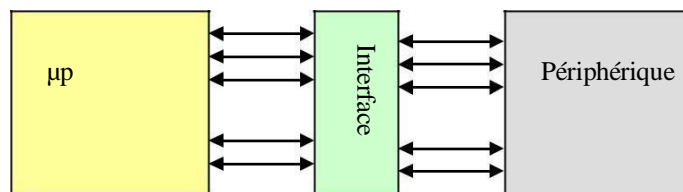


Figure V.1 liaison parallèle

##### V.1.2 Liaison série

Dans ce type de liaison, les bits constitutifs d'un mot sont transmis les uns après les autres sur un seul fil. Les distances de transmission peuvent donc être plus beaucoup plus importantes mais la vitesse de transmission est plus faible. Sur des distances supérieures à quelques dizaines de mètres, on utilisera des modems aux extrémités de la liaison.

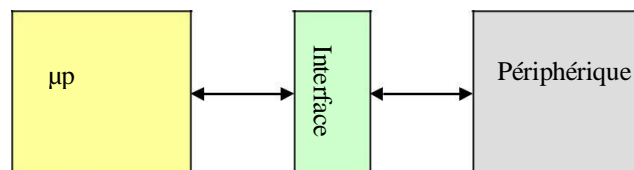


Figure V.2 liaison série



La transmission de données en série peut se concevoir de deux façons différentes :

**En mode synchrone**, l'émetteur et le récepteur possède une horloge synchronisée qui cadence la transmission. Le flot de données peut être ininterrompu.

**En mode asynchrone**, la transmission s'effectue au rythme de la présence des données.

Les caractères envoyés sont encadrés par un signal start et un signal stop.

### ✓ **Principe de base d'une liaison série asynchrone :**

Afin que les éléments communicants puissent se comprendre, il est nécessaire d'établir un protocole de transmission. Ce protocole devra être le même pour chaque élément.

#### ➤ Paramètres rentrant en jeu :

- Longueur des mots transmis : 7 bits (code ASCII) ou 8 bits
- vitesse de transmission : les vitesses varient de 110 bit/s à 128000 bit/s et détermine les fréquences d'horloge de l'émetteur et du récepteur.

**parité** : le mot transmis peut être suivi ou non d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux types de parité : paire ou impaire. Si on fixe une parité paire, le nombre total de bits à 1 transmis (bit de parité inclus) doit être paire. C'est l'inverse pour une parité impaire.

**bit de start** : la ligne au repos est à l'état 1 (permet de tester une coupure de la ligne). Le passage à l'état bas de la ligne va indiquer qu'un transfert va commencer. Cela permet de synchroniser l'horloge de réception.

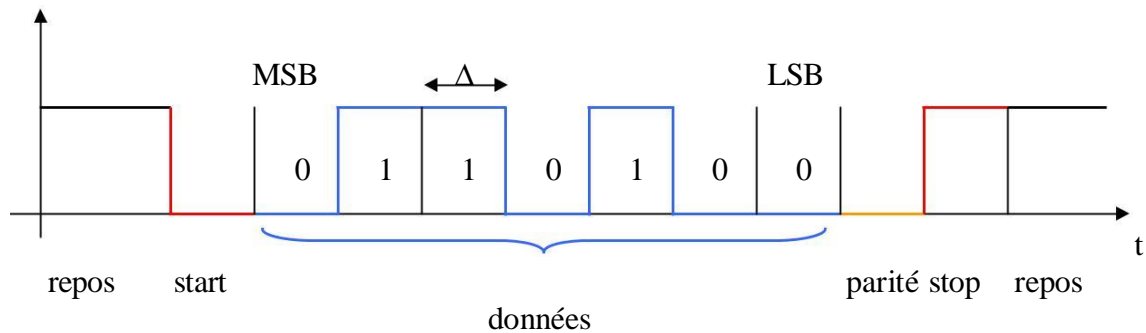
**bit de stop** : après la transmission, la ligne est positionnée à un niveau 1 pendant un certain nombre de bit afin de spécifier la fin du transfert. En principe, on transmet un, un et demi ou 2 bits de stop.

#### ➤ Déroulement d'une transmission :

Les paramètres du protocole de transmission doivent toujours être fixés avant la transmission. En l'absence de transmission, la liaison est au repos au niveau haut pour détecter une éventuelle coupure sur le support de transmission. Une transmission s'effectue de la manière suivante :

L'émetteur positionne la ligne à l'état bas : c'est le bit de start. Les bits sont transmis les uns après les autres, en commençant par le bit de poids fort. Le bit de parité est éventuellement transmis. L'émetteur positionne la ligne à l'état haut : c'est le bit de stop.

Exemple : transmission d'un mot de 7 bits (0110100)<sub>2</sub> – Parité impaire – 1 bit de Stop



## V.2 Interface d'entrées/sorties

### V.2.1 Définition

Une interface d'entrées/sorties est un circuit permettant au CPU de communiquer avec l'environnement extérieur (périphérique) : clavier, écran, imprimante, processus industriel etc...

Les interfaces d'E/S sont connectées au microprocesseur à travers les bus d'adresses de données et de commandes. Un circuit d'E/S possède des registres pour gérer les échanges avec les périphériques :

- Registre de configuration
- Registre de données

A chaque registre est assignée une adresse : le microprocesseur accède à un port d'E/S en spécifiant l'adresse de l'un de ses registres. Intel a développé plusieurs circuits intégrés de contrôle de périphérique conçus pour soutenir la famille de processeurs 80x86 tels que:

Le 8255A Programmable Peripheral Interface (PPI),

Le 8259 Programmable Interrupt Controller (PIC),

Le 8253/54 Programmable Interval Timer

(PIT), Le 8237 Programmable DMA  
Controller.

### V.2.2 L'interface parallèle 8255

Le rôle d'une interface parallèle est de transférer des données du microprocesseur 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.

## Chapitre V :Les interfaces d'entrées/sorties

Le 8255 fournit 24 lignes d'E/S qui peuvent être organisées en trois ports d'E/S (A, B, et C) de 8 bits chacun. De plus le 8255 peut fonctionner selon 3 modes: mode 0, mode1 ou mode2.

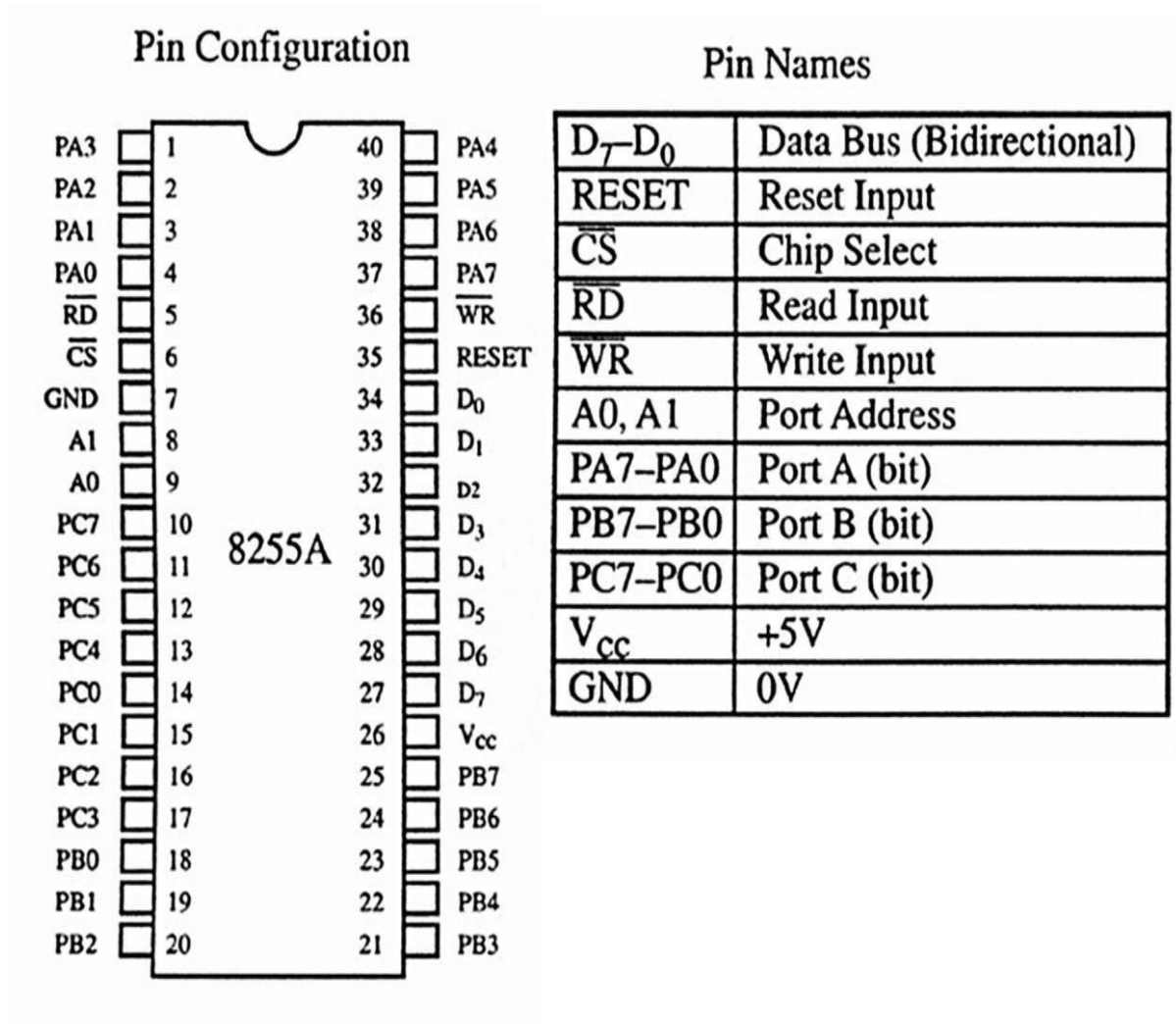
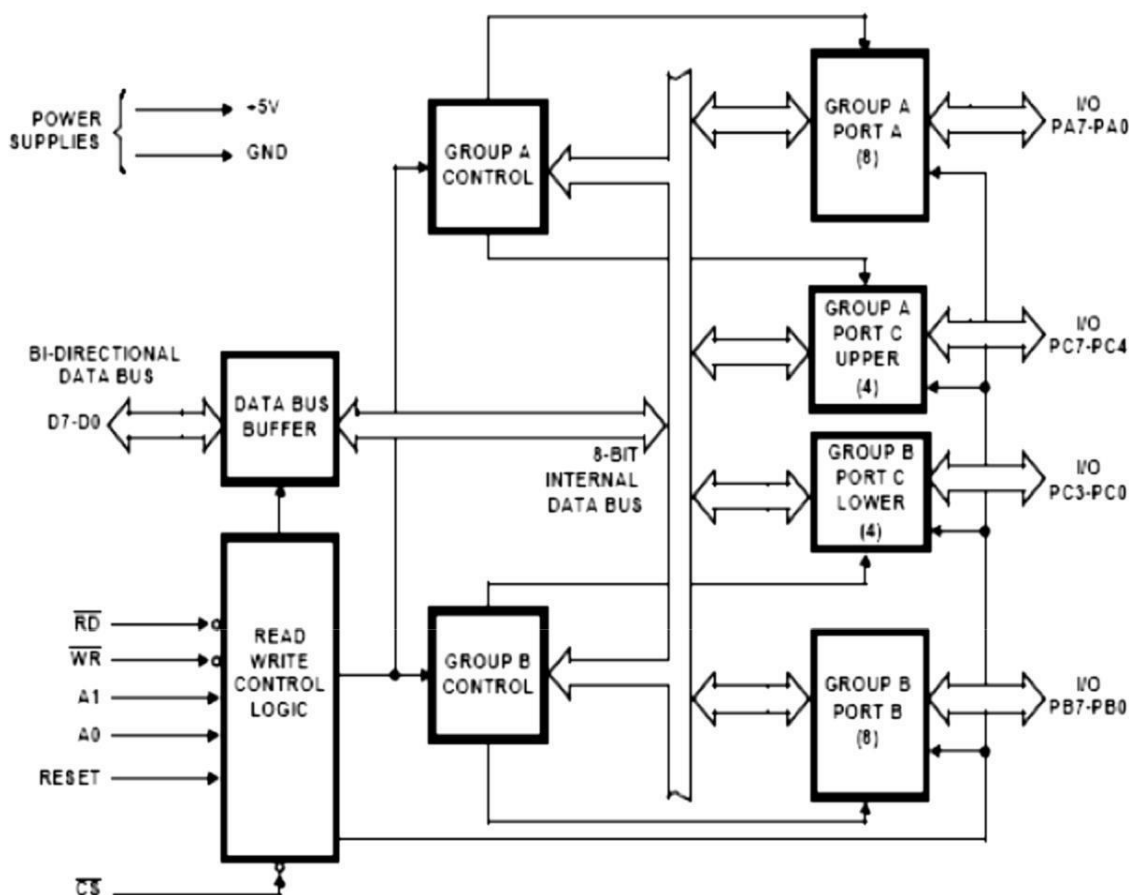


Figure V.3 : brochage du 8255

$A_1$	$A_0$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	0	1	0	<i>Input operation (READ)</i>
0	1	0	1	0	Port A → data bus
1	0	0	1	0	Port B → data bus
					Port C → data bus
					<i>Output operation (WRITE)</i>
0	0	1	0	0	Data bus → port A
0	1	1	0	0	Data bus → port B
1	0	1	0	0	Data bus → port C
1	1	1	0	0	Data bus → control
					<i>Disable function</i>
X	X	X	X	1	Data bus tristate
1	1	0	1	0	Illegal condition
X	X	1	1	0	Data bus tristate

Tableau V.1 : Sélection des ports de 8255



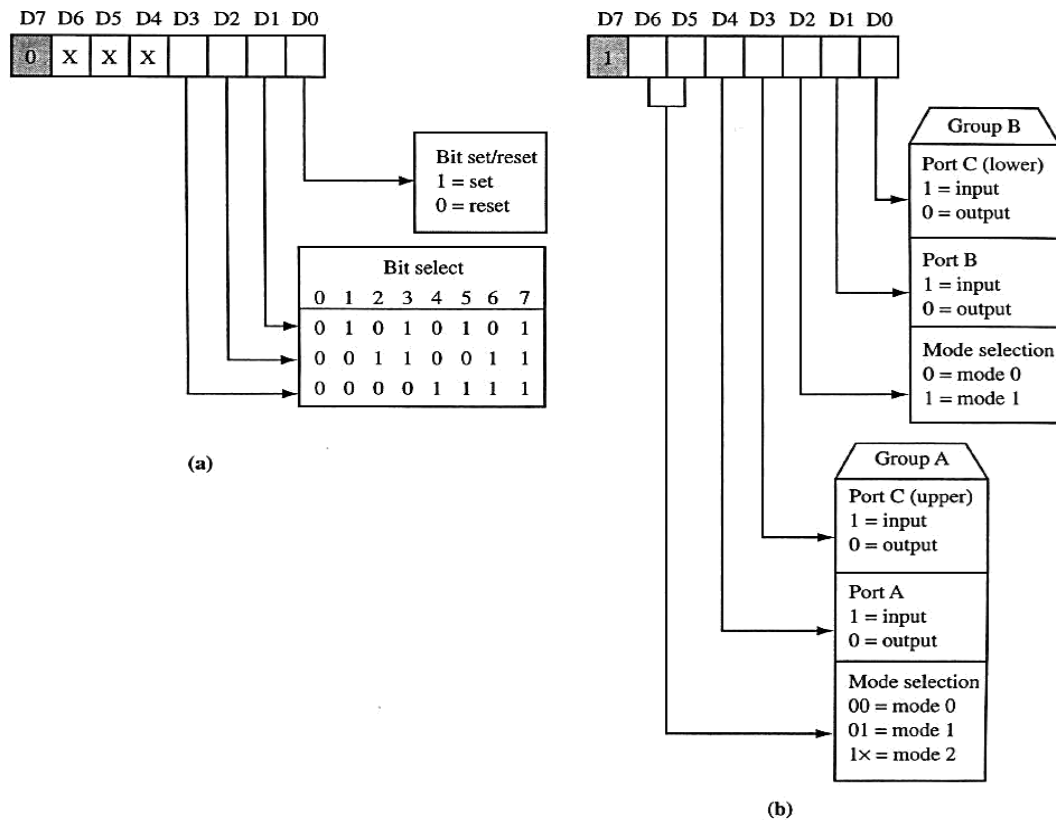


Figure V.4 le fonctionnement du PPI 8255

La figure précédente montre: le fonctionnement du PPI 8255

- (a) Lorsque le bit 7 = 0, mise à « 1 » ou « 0 » d'une ligne du PORTC individuellement.
- (b) Lorsque le bit 7 = 1, l'un des modes 0, 1, ou 2 peuvent être programmés.

**Mode 0 (E/ S de base):** Ports A et B fonctionnent comme entrées ou sorties. Le port C est divisé en deux groupes de 4 bits qui peuvent être configurés comme entrées ou sorties.

**Mode 1 :** Il est utilisé pour le dialogue avec des périphériques nécessitant un asservissement (contrôle). Ports A et B fonctionnent comme des entrées ou sorties comme en mode 0. Port C est utilisé pour le contrôle.

**Mode 2:** le port A est bidirectionnel (entrée et sortie).

Port C est utilisé comme signaux du contrôle. Port B n'est pas utilisé.

**Remarque :** Ces modes peuvent aussi être mélangés. Par exemple, le port A peut être programmé pour fonctionner en mode 2, tandis que le port B fonctionne en mode 0.

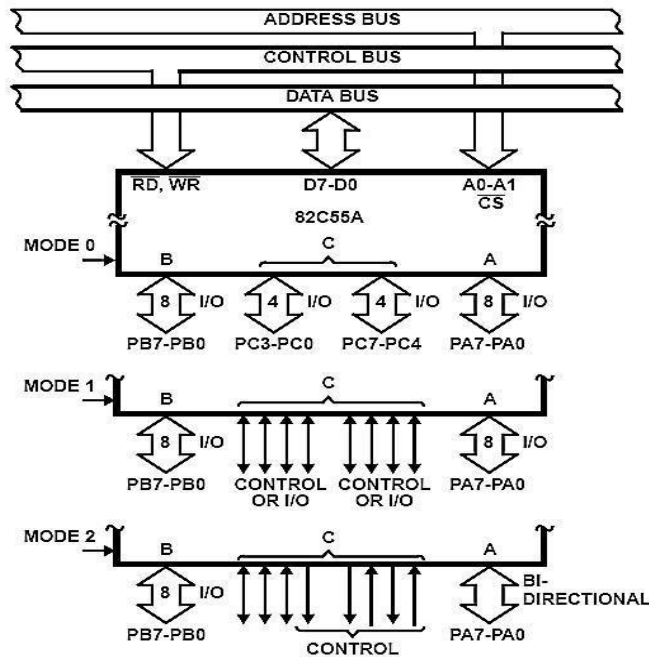


Figure V.5 le fonctionnement du PPI 8255

## -Interfaçage de 8086 avec le 8255

On peut connecter le bus de données du 8255 sur les lignes de données de poids faible du 8086 (D0 - D7) ou sur celles de poids fort (D8 - D15). Donc l'un des deux signaux A0 ou BHE doit être utilisé pour sélectionner le 8255 alors les adresses des registres du 8255 se trouvent à des adresses paires (validation par A0) ou impaires (validation par BHE).

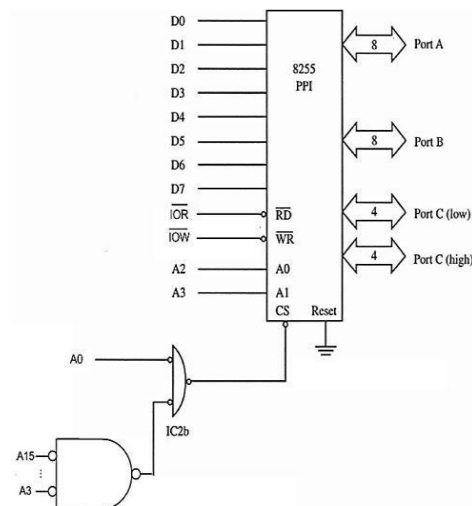
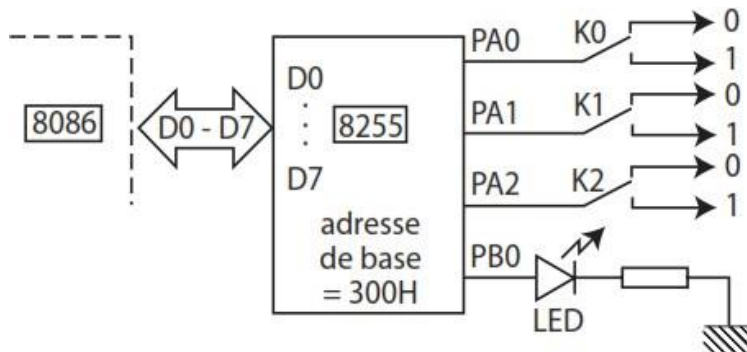


Figure V.6. Interfaçage de 8086 avec le 8255

## Exemple :

A15-A12 A11-A8 A7-A4	A2 A1	A0
0000 0000 0000	0 0	0
0000 0000 0000	0 1	0
0000 0000 0000	1 0	0
0000 0000 0000	1 1	0
Chip Select (CS)	Sélection de port	adresses paires

## Exemple de programmation : soit le montage suivant :



On veut que la led s'allume lorsqu'on a la combinaison : K0 = 1 et K1 = 0 et K2 = 1.

Solution

*portA equ 300H ; adresses des registres du 8255*

*portB equ 302H*

*portC equ 304H*

*controle equ 306H*

*mov dx,controle ; initialisation du port A en entrée*

*mov al,90H ; et du port B en sortie (mode 0) :*

*out dx,al ; controle = 10010000B = 90H*

*boucle : mov dx,portA ; lecture du port A*

*in al,dx*

*and al,00000111B ; masquage PA0, PA1 et PA2*

*cmp al,00000101B ; test PA0 = 1, PA1 = 0 et PA2 = 1*

*jne faux ; non -> aller au label "faux" ...*

*mov al,00000001B ; oui -> mettre PB0 `a 1*

*jmp suite ; et continuer au label "suite"*

*faux : mov al,00000000B ; ... mettre PB0 `a 0*

*suite : mov dx,portB ; écriture du port B*

*out dx,al*

*jmp boucle ; retourner lire le port*

### V.2.3 l'interface série, l'UART 8250

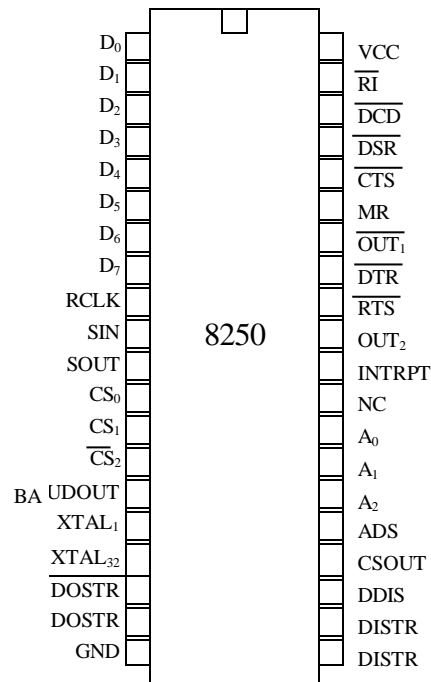


Figure V.7: Brochage du l'UART 8250

Le 8250 possède 11 registres. Comme il n'y a que 3 bits d'adresses ( $A_0$ ,  $A_1$  et  $A_2$ ), plusieurs registres doivent se partager la même adresse :

DLAB	A2	A1	A0	Registre
0	0	0	0	RBR : Receiver Buffer Register, registre de réception (accessible seulement en lecture)
0	0	0	0	THR : Transmitter Holding Register, registre d'émission (accessible seulement en écriture)
1	0	0	0	DLL : Divisor Latch LSB, octet de poids faible du diviseur d'horloge
1	0	0	1	DLM : Divisor Latch MSB, octet de poids fort du diviseur d'horloge
0	0	0	1	IER : Interrupt Enable Register, registre d'autorisation des interruptions
X	0	1	0	IIR : Interrupt Identification Register, registre d'identification des interruptions
X	0	1	1	LCR : Line Control Register, registre de contrôle de ligne
X	1	0	0	MCR : Modem Control Register, registre de contrôle modem
X	1	0	1	LSR : Line Status Register, registre d'état de la ligne
X	1	1	0	MSR : Modem Status Register, registre d'état du modem
X	1	1	1	SCR : Scratch Register, registre à usage général

Tableau V.2 : Sélection des registres du 8250



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

### ✓ Line Control Register (LCR)

Bits 0 et 1 : longueur du mot transmis,	bit 1	bit 0	
	0	0	→ 5 bits
	0	1	→ 6 bits
	1	0	→ 7 bits
	1	1	→ 8 bits
Bit 2 : nombre de bits de stop,	0	→ 1 bit de stop,	
	1	→ 1.5 bits de stop si 5 bits, 2 bits de stop sinon;	
Bit 3 : autorisation de parité,	0	→ pas de parité,	
	1	→ parité générée et vérifiée ;	
Bit 4 : sélection de parité,	0	→ parité impaire	
	1	→ parité paire	
Bit5: /			
Bit 6: /			
Bit 7 : DLAB (Divisor Latch Access bit),	0	→ accès aux registres d'émission, de réception et IER,	
	1	→ accès au diviseur d'horloge.	

### ✓ Line Status Register (LSR)

bit 0 : 1	→ donnée reçue ;
bit 1 : 1	→ erreur d'écrasement de caractère ;
bit 2 : 1	→ erreur de parité ;
bit 3 : 1	→ erreur de cadrage (bit de stop non valide) ;
bit 4 : 1	→ détection d'un état logique 0 sur RxD pendant une durée supérieure à la durée d'un mot ;
bit 5 : 1	→ registre de transmission vide ;
bit 6 : 1	→ registre à décalage vide ;
bit 7 :	non utilisé, toujours à 0

### ✓ Diviseur d'horloge (DLM,DLL)

La vitesse de transmission est fixée par la valeur du diviseur d'horloge

$$\text{vitesse (bits/s)} = \text{fréquence d'horloge} / 16x \text{ (DLM, DLL)}$$

Exemple de calcul :

Vitesse de transmission désirée = 1200 bit/s, fréquence d'horloge = 1.8432 MHz.

Détermination de la valeur du diviseur d'horloge :

$$\text{diviseur} = \text{fréquence d' horloge} / 16 \times \text{vitesse} = 1.8432 \times 10^6 / 16 \times 1200 = 96 \text{ alors } DLM = 0 \text{ et } DLL = 96$$

### • Ports USB (Universal Serial Bus)

Le nouveau Bus USB de connexion série, promu par les principaux constructeurs informatiques et éditeurs de logiciels (Compaq, DEC, IBM, Intel, Microsoft, NEC, Nortel) et dont les premières spécifications doivent permettre le raccordement sur une prise unique de plusieurs équipements divers (imprimante, téléphone, modem, fax, clavier, souris, scanners, écrans...). On désire ainsi éviter la multiplication actuelle des connecteurs sur les PC. Ce bus permet la transmission de données en série. Cette nouvelle technique se doit d'être rapide, bidirectionnelle, synchrone, de faible coût et l'attachement d'un nouveau périphérique doit être dynamique. De plus l'alimentation des équipements est possible.

Le débit brut (émission plus réception) peut aller jusqu'à 12 Mbit/s au maximum. Le débit brut doit être partagé entre tous les appareils connectés au bus. Le port USB possède 4 broches à savoir : une paire d'alimentation (VCC et GND) et une paire torsadée de données inversés (-DATA et +DATA). Au repos +DATA et -DATA sont à l'état haut et bas respectivement

### V.2.4 Timer (Temporisateur), Intel 8254 [9]

Le Timer 8254 est un contrôleur d'entrées/sortie. Il fait le couplage avec une horloge pour mettre à disposition de la machine un mécanisme de mesure de temps. Le schéma de la figure V.8 donne une représentation minimale de connexion en se limitant au bus de données et aux signaux permettant de sélectionner les registres du contrôleur. Les bits A0 et A1 permettent d'accéder à l'un des quatre registres internes : un registre par compteur (Compteur 0 : A1=0 et A0=0, Compteur 1 : A1=0 et A0=1, Compteur 2 : A1=1 et A0=0) et registre de commande (A1=1 et A0=1).

Le CS est le résultat du décodage d'adresses des autres bits du bus d'adresses

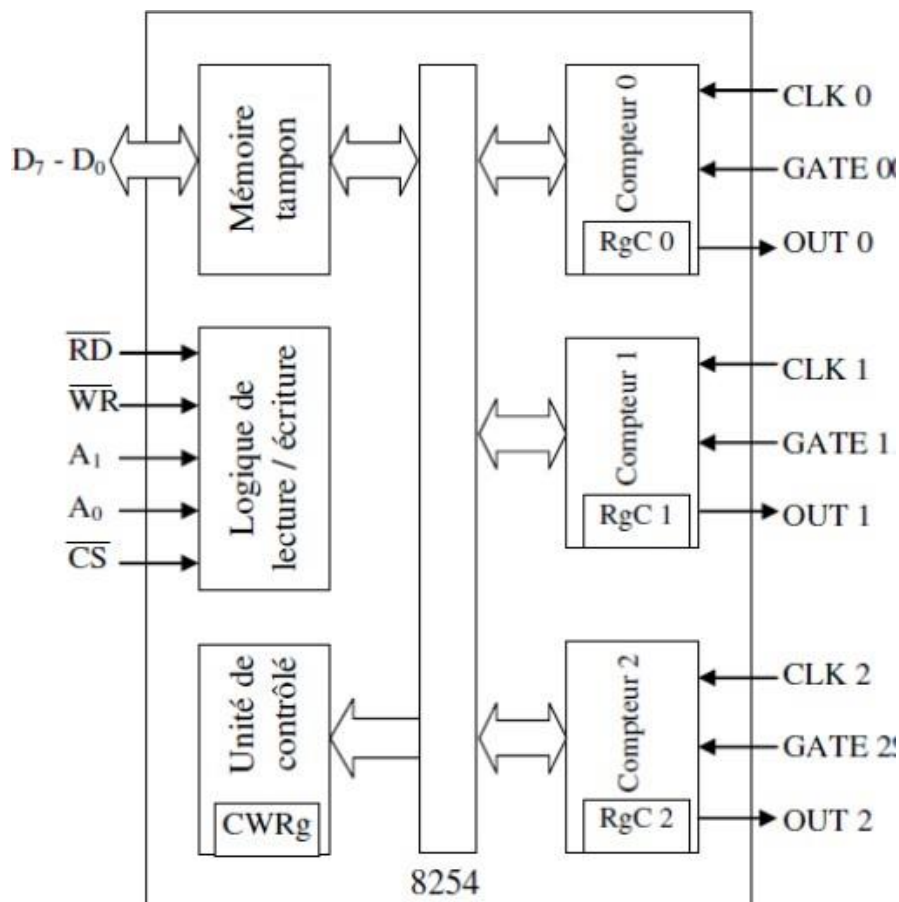


Figure V.7: Brochage du Timer 8254

Le circuit comporte trois unités fonctionnelles identiques, appelés compteurs, axées sur le comptage du temps. Un compteur comporte deux entrées matérielles (CLK), Gate et une entrée logicielle, le compteur RgC. l'entrée CLK reçoit des impulsions qui peuvent être périodiques ou non.

GATE est une entrée matérielle de validation du comptage. Le registre 16 bits RgC est accessible par programmation et peut être pré positionné à une certaine valeur. Chaque impulsion sur l'entrée CLK décrémente le compteur. La sortie OUT est initialement à 0 passe à 1 lorsque RgC passe à la valeur 0. A partir de l'initialisation du registre compteur, le 8254 propose 6 modes de fonctionnement. C'est la programmation du registre CWRg (Control Word Register) qui permet de configurer le mode de fonctionnement d'une unité fonctionnelle avec les valeurs des bits M2 à M0. SC1 et SC0 servent à définir dans un mot de commande l'unité de comptage (Sélection du Compteur 0 : SC1=0 et SC0=0, Compteur 1 : SC1=0 et SC0=1, Compteur 2 : SC1=1 et SC0=0) faisant l'objet du paramétrage par le mot de commande. Les bits RW0 et RW1 définissent le mot de transfert des données. Le bit BCD permet de réaliser un comptage BCD.

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	
SC1	SC0	RW1	RW0	M2	M1	M0	BCD	
							0	: Comptage binaire
							1	: Comptage BCD
				0	0	0		: Mode 0
				0	0	1		: Mode 1
				X	1	0		: Mode 2
				X	1	1		: Mode 3
				1	0	0		: Mode 4
				1	0	1		: Mode 5
		0	0					: Commande du latch Compteur
		0	1					: R/W LSB 8 bits
		1	0					: R/W MSB 8 bits
		1	1					: R/W LSB ensuite MSB
0	0							: Sélection du Compteur 0
0	1							: Sélection du Compteur 1
1	0							: Sélection du Compteur 2
1	1							: Commande de lecture

Tableau V.3 : Mot de contrôle CWRg du 8254

### Exemple du mode 0

Ce mode est adapté au comptage d'évènements. Après avoir programmé l'unité de comptage par le biais du registre de commande et initialisé le compteur. Lorsque GATE passe à 1. Le compteur est alors décrémenté à chaque évènement signalé par une impulsion sur l'entrée CLK. La sortie OUT passe de sa valeur initiale 0 à la valeur 1 lorsque le comptage arrive à 0. OUT reste alors au niveau haut jusqu'au chargement d'une nouvelle valeur dans le registre compteur. La sortie OUT peut servir de signal d'interruption.

### V.2.5 Le contrôleur programmable d'interruptions 8259

Le microprocesseur 8086 ne dispose que de deux lignes de demandes d'interruptions matérielles (NMI et INTR). Pour pouvoir connecter plusieurs périphériques utilisant des interruptions, on peut utiliser le contrôleur programmable d'interruptions.

Un contrôleur d'interruption programmable (PIC) fonctionne comme un directeur général dans un environnement de système commandé par interruption.

Il accepte les demandes provenant de l'équipement périphérique, détermine laquelle des demandes entrantes est de la plus haute importance (priorité), vérifie si la demande entrante a une valeur de priorité plus élevée que le niveau en cours d'exécution et émet une interruption vers le processeur sur la base de cette détermination.

#### Exemple :

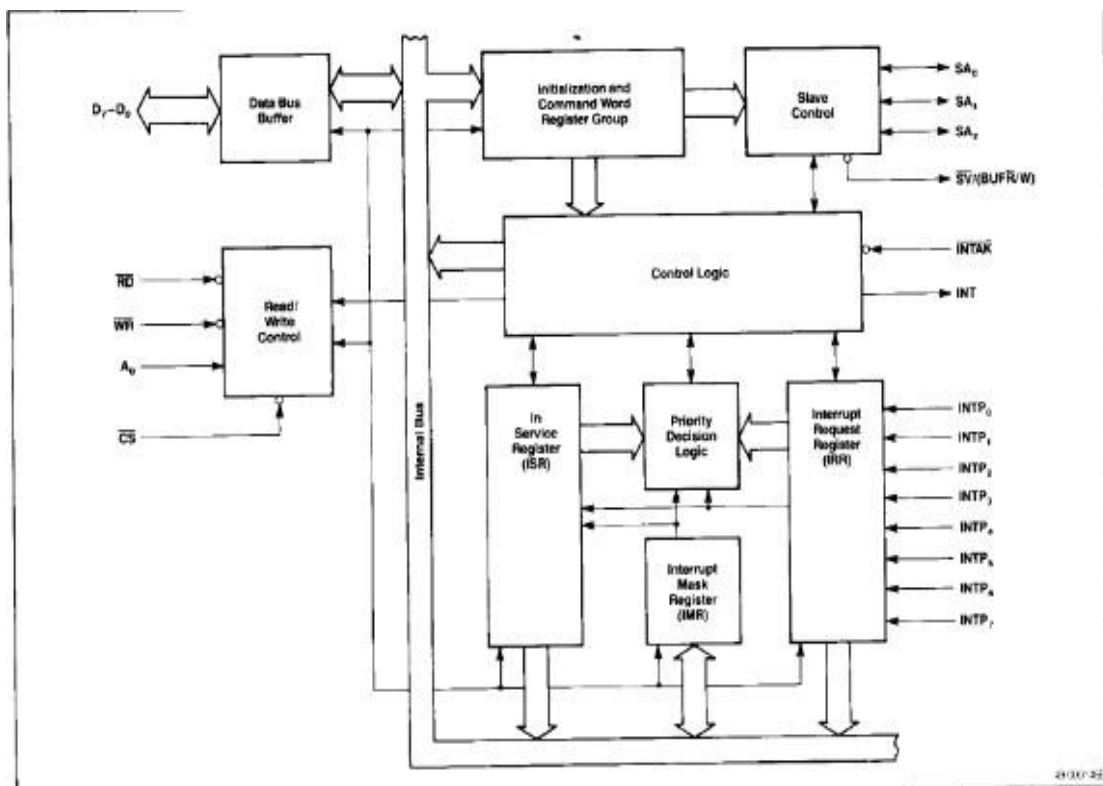
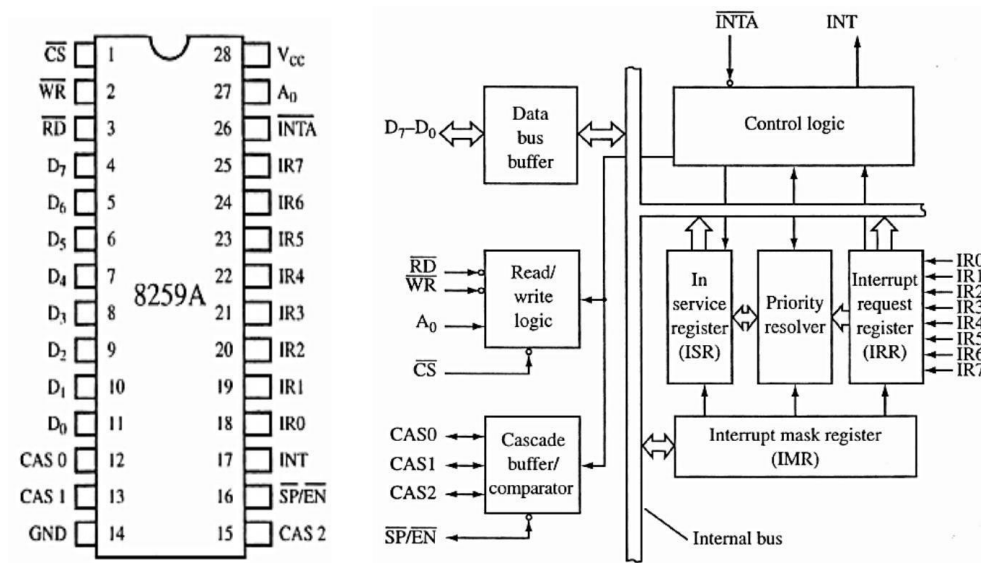
#### ✓ le rôle de PIC 8259 :

- recevoir des demandes d'interruptions des périphériques ;
- résoudre les priorités des interruptions ;
- générer le signal INTR pour le 8086 ;
- émettre le numéro de l'interruption sur le bus de données.

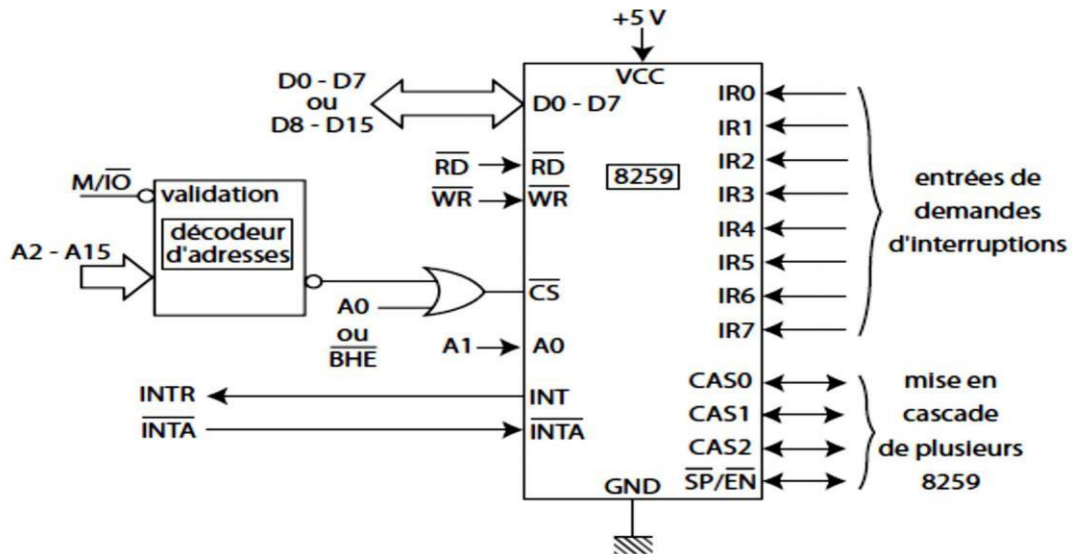
Le 8259A peut gérer jusqu'à 8 demandes d'interruptions matérielles. Il est programmé par le logiciel du système comme un périphérique d'E / S. Une sélection de modes de priorité est disponible pour le programmeur de sorte que la manière dont les demandes sont traitées par le 8259A peut être configurée pour correspondre à ses besoins du système.

Les modes de priorité peuvent être modifiés ou reconfigurés dynamiquement à tout moment pendant le programme principal. Cela signifie que la structure d'interruption complète peut être définie selon les besoins.

### V.2.5.1 Brochage du 8259 :



**Exemple :** interfaçage de 8259 avec le 8086 :



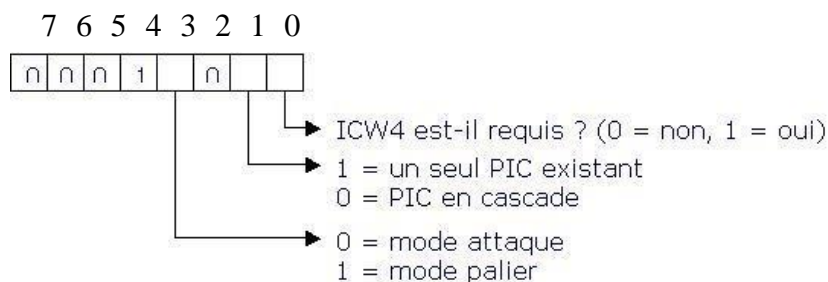
**Remarque :** si le nombre de demandes d'interruptions est supérieur à 8, on peut placer plusieurs 8259 en cascade.

Maintenant, nous allons voir la programmation des PICs. Il y a deux types de commandes que l'on peut adresser au PIC : les OCW et ICW (Operational Command Words, Initialization Command Words).

A la différence des OCW, vous devez envoyer les ICW dans un ordre bien précis.

## ICW 1 :

Pour commencer, vous devez d'abord envoyer **ICW1** sur le premier port du PIC, soit 20H (maître) ou 0A0H (esclave).



Le premier bit concerne ICW4. Quand vous vous lancez dans l'initialisation d'un des PICs, vous devez obligatoirement envoyer ICW1, ICW2 et ensuite ICW3. Cependant, en mettant ce bit à un, vous indiquez au PIC que vous enverrez ICW4 juste après ICW3 ; sinon, pour ne pas l'envoyer, il suffit de le mettre à 0. Le deuxième bit permet d'indiquer au PIC s'il y a un ou deux PICs.. Le quatrième bit (3) montre au PIC la manière dont les lignes d'interruption IR0 à IR7 sont activées. Les sources d'interruption demandent une

interruption via ces lignes. L'opération se déroule en revanche en mode attaque dans un environnement PC (bit 3 = 0). Cela veut dire qu'une source d'interruption met momentanément sa ligne IR sur *high* pour la passer ensuite sur *low* pour informer le PIC de sa demande. En mode palier, la source d'interruption laisse sa ligne sur *high* ce qui provoque l'exécution ininterrompue de l'interruption concernée jusqu'à ce que la source remette la ligne sur *low*. En général, on utilise le mode attaque (bit = 0)

### **ICW2.**

qui fait suite à ICW1 indique au PIC le numéro d'interruption de base. Soit x un numéro d'IRQ entre 0 et 7. x, il faut savoir que vous ne pouvez pas donner n'importe quel numéro d'interruption, mais seulement un numéro où les 3 premiers bits sont à 0.

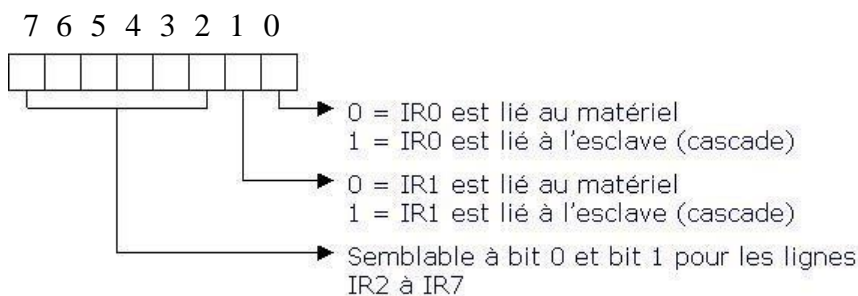


Il faut envoyer cette valeur sur le port 21H ou 0A1H, et ce sera de même pour ICW3 et ICW4 (si vous devez l'envoyer).

### **ICW3**

qui suit ICW2 ne doit être envoyé au PIC que s'il est relié en cascade avec l'autre PIC. Il attend cette commande que si le bit 1 de ICW1 se trouvait à 1, à l'envoi de ICW3, il y a une distinction à savoir si vous êtes en train de programmer le maître ou l'esclave. L'octet se construit différemment.

#### Pour le maître :





Cette commande permet d'indiquer au maître par quelle ligne d'interruption l'esclave est-il relié au maître. Comme dans les AT, ils sont reliés par l'intermédiaire de l'IRQ2, le BIOS envoie la valeur 00000100b soit 4H ( sur le port 21H)

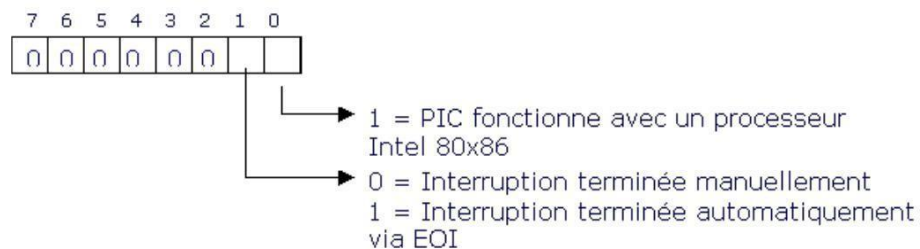
Pour l'esclave :



Cette commande permet d'indiquer à l'esclave par quelle ligne d'interruption il est relié au maître. Dans les AT, ils sont reliés par l'intermédiaire de l'IRQ2, donc le BIOS envoie 2 comme valeur (au port 0A1H).

### ICW4

Informe donc le PIC s'il fonctionne dans un environnement Intel, et s'il doit enregistrer automatiquement la fin d'une Interruption ou bien s'il doit demander l'aide d'un logiciel. En général, c'est la méthode manuelle qui est appliquée dans les PC, soit bit1 = 0.

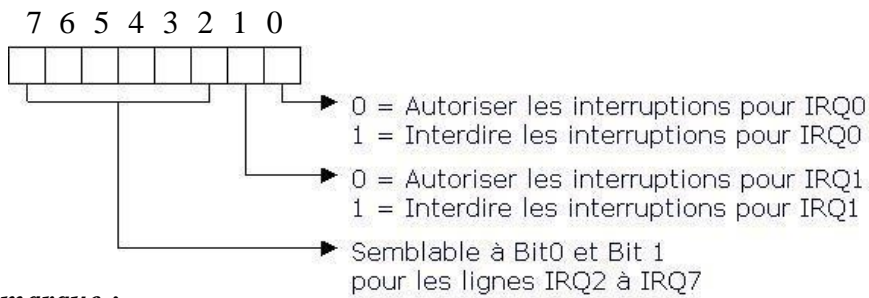


### OCW :

Contrairement à l'initialisation du PIC, vous pouvez envoyer chaque ordre, indifféremment, dans n'importe quel ordre.

### OCW1

L'envoi de OCW1 se fait sur le second port (20H ou 0A1H). Il permet d'activer ou de désactiver les lignes d'interruptions. En général, ce registre se trouve à 0, autorisant toutes les interruptions. A la différence des autres registres, vous pouvez lire son état en faisant une lecture sur le second port (21H ou 0A1H)



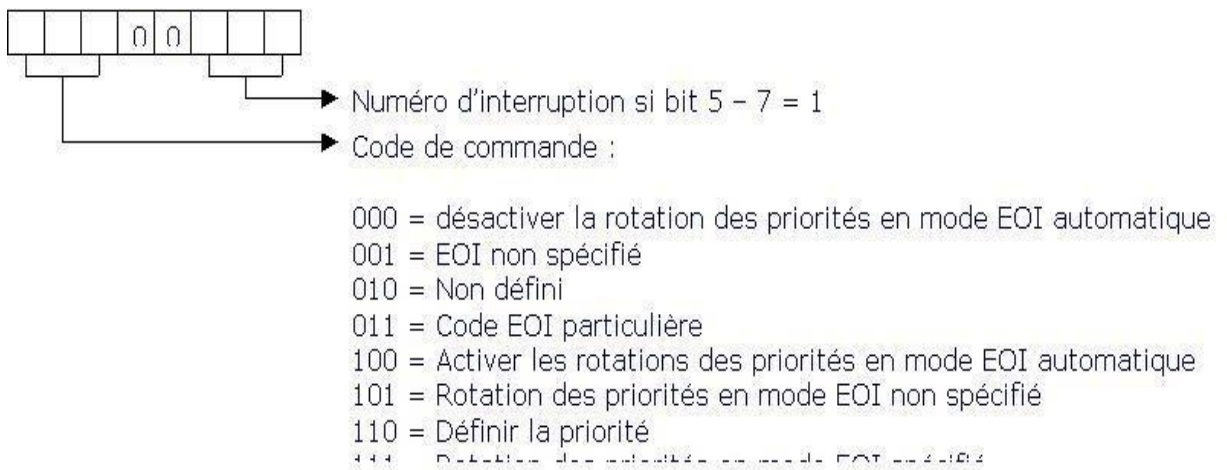
**Remarque :**

La fonction « STI » met la valeur 11111111b dans ce registre, interdisant toute interruption de type hardware. Par contre, la fonction « CLI » met la valeur 0 dans ce registre pour activer toutes les interruptions.

**OCW2**

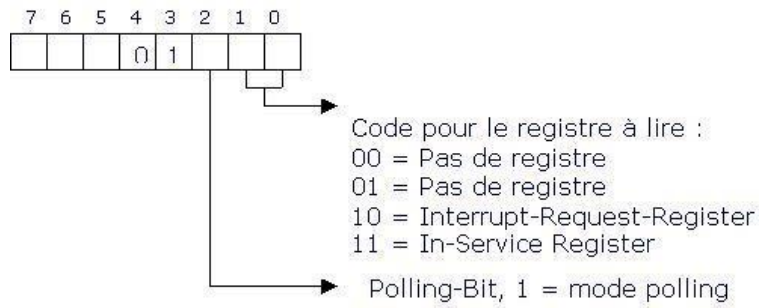
Comme les PICs ont été programmés en mode EOI manuel, OCW2 permet de finir une interruption, soit envoyer ce fameux code EOI. Pour ce faire, il vous suffit d'envoyer la valeur 00100000b soit 20h sur le premier port : 20H ou 0A0H. Les autres fonctions ne sont pas très utiles, et n'ont pas trop d'utilité dans un ordinateur.

7 6 5 4 3 2 1 0



**OCW3**

OCW3 permet à un programme de lire les registres internes nommés IRR et ISR du PIC. La commande doit être envoyée sur le premier port du PIC, tout comme OCW2 : les bits 3 et 4 permettant de les distinguer. Après l'accès en écriture sur le premier port, le registre souhaité peut être demandé à l'étape suivante via un accès en lecture sur le port. Dans ce contexte, un programme peut servir à déterminer le gestionnaire d'interruption en cours d'exécution et celui qui attend son tour.

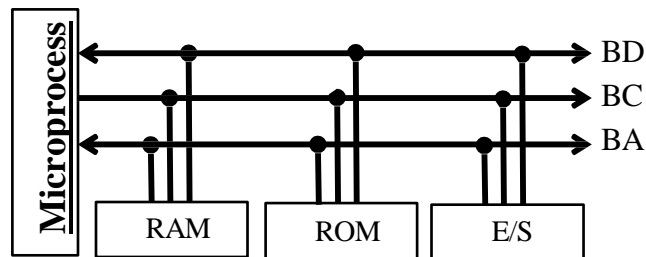


**Série de TD N° : 1**

**Généralités sur les Systèmes à base de microprocesseur**

**Exercice 1 :**

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



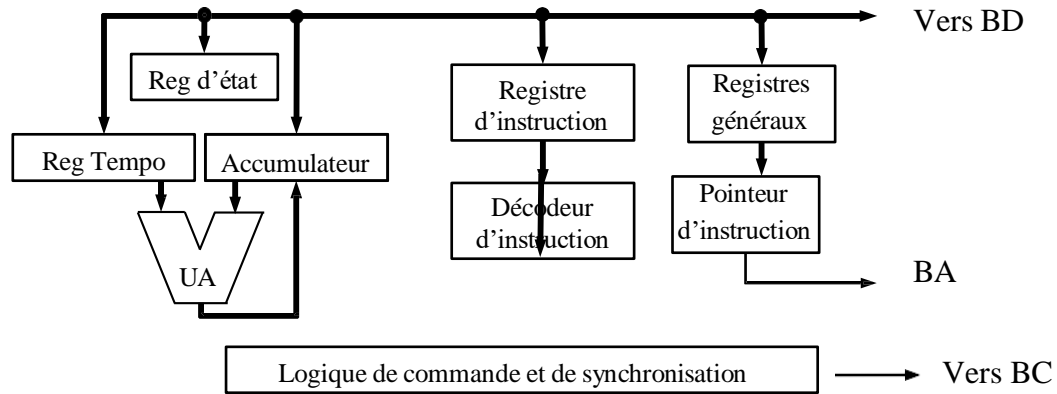
**Figure 1.**

**Exercice 2 :**

1. Combien de lignes d'adresses sont nécessaires pour adresser : 8 Ko, 64 Ko, 12 Ko et 100 Ko.
2. Le nombre de lignes d'adresses dépend-t-il de la taille du mot du système.
3. Dans la plupart des ordinateurs, les adresses des cases mémoires sont exprimées en hexadécimal.
  - a. Combien de cases contient une zone dont l'adresse de base (début) est 300H et son adresse haute (fin) est F9FH ?
  - b. Une zone mémoire occupe 16 Ko, si son adresse de base est 0000H, quelle est l'adresse haute de cette zone ?

**Exercice 3 :**

**On considère le schéma synoptique ci-dessous :**



**Figure 2.**

1. Quel est le rôle de chaque unité ?
2. Donner les différentes étapes par lesquelles passe l'exécution d'une instruction.

**Série de TD N° : 2**

**Architecture interne du microprocesseur Intel 8086**

Répondre aux questions suivantes :

1. Représenter sur un schéma synoptique la structure interne du microprocesseur Intel 8086.
2. Quelle est la capacité d'adressage mémoire du 8086 ?
3. Pour un 8086, donnez le nom, la taille et expliquez le rôle des registres suivants : AX, CX, DX, IP, DS.
4. Expliquer le rôle de chaque bit du registre d'état (flag) du 8086.
5. Donner la définition, la taille et la fonction de la file d'attente.
6. Quelle est la fonction de l'unité d'exécution (UE).
7. Quelle est la fonction de l'unité d'interface de bus (UIB) ?
8. Définir : l'adresse effective, l'adresse logique et l'adresse physique.
9. Comment se calcule l'adresse physique.
10. Que contient chacun des segments mémoires du 8086 ?

**Série de TD N° : 3**

**Programmation du Microprocesseur 8086 en Assembleur**

1. Dans l'extrait de programme suivant, préciser pour chacune des instructions le mode d'adressage :

Instruction	Mode d'adressage
MOV AL, [000BH]	
ADD AL, C4H	
MOV [BX], 00H	
SUB AL, [BX + SI]	

**Ecrire des programmes en assembleur 8086 qui permettent de :**

2. Evaluer l'expression suivante et stocker le résultat en mémoire à l'adresse 0300H :

$$Y = 2x^2 + 3x - 1.$$

Tel que x est un nombre positif (1 octet) stocké en mémoire à l'adresse 0100H.

3. Compter le nombre d'éléments nuls dans un tableau de 100 octets stockés en mémoire à partir de l'adresse 200H. Le résultat sera stocké dans la case mémoire d'adresse 400H.

4. compter le nombre de 1 dans un mot binaire de 16 bits stocké dans l'accumulateur. Le résultat sera stocké dans la case mémoire d'adresse 1200H.

5. Trier par ordre croissant (du plus petit au plus grand) les 50 éléments d'un tableau stocké en mémoire à l'adresse 300H.

6. Calculer la somme des nombre entiers pairs (de 0 à 100) et stocker le résultat dans le registre BX.

Calculer la somme des nombre entiers impairs (de 0 à 100) et stocker le résultat dans le registre DX.

Le programme doit utiliser une seule boucle.

7. Calculer la factorielle d'un nombre entier n de 8 bits (stocké en mémoire à l'adresse 0210H) noté n!. Le résultat doit être stocké en mémoire à l'adresse 0220H.

- Quelle peut être la taille du résultat ?
- Quelle condition doit satisfaire le nombre n pour qu'il n'y ait pas dépassement de cette taille ?
- Il faut prendre en considération le cas  $0! = 1$

## **TP01 : prise en main du kit mts-86c**

### **1. Principaux composants**

Le kit **MTS-86C** est composé d'un microprocesseur 16 bits (8086 d'Intel) auquel sont connectés tous les constituants principaux de l'informatique industrielle :

- ✓ RAM (64Ko) 2 x 62256
- ✓ EPROM Moniteur (64Ko) 2 x 27256
- ✓ EPROM Utilisateur (64Ko) 2 x 27256
- ✓ Interface parallèle 3 x 8255
- ✓ Interface série 2 x 8251
- ✓ Contrôleur de clavier 8279
- ✓ Contrôleur d'interruption 8259
- ✓ Timer programmable 8253
- ✓ ADC (8 bits – 8 entrées) ADC809
- ✓ DAC (8 bits) DAC0808

### **2. Cartographie de la mémoire du kit**

L'espace mémoire utilisé est divisé en trois parties (Figure 1). Une zone pour le programme moniteur et exemples d'application, la seconde pour le programme utilisateur et les vecteurs d'interruptions et la troisième, une zone extensible, elle peut être du type ROM ou RAM.

FFFFFFH	Monitor program	ROM
F8000H	Exercise program	
F0000H	User memory	
E0000H	OPEN	RAM
10000H	User program	
00400H	Interrupt Vector Table	
00000H		

**Figure 1 : Cartographie de la mémoire du kit MTS-86C**



Dans notre laboratoire, nous allons étudier le 8086 Kit MTS-86C. Un Kit avec installation I / O et de même construction et interfaçage que les dispositifs 8255, 8251, 8259, 8253, 8254, etc.

### **3. Objectifs :**

L'objectif de cette manipulation est de rappeler l'étudiant les notions de la programmation assembleur selon les instructions de 8086. On procédera par la suite à introduire la procédure de l'utilisation du KIT MTS-86C en utilisant des programmes assembleur simples à la fin de la manipulation l'étudiant sera capable de :

- ✓ Ecrire un programme assembleur simple
- ✓ Etablir la connexion entre le PC et le Kit
- ✓ Compiler le programme éditer et créer le fichier hexadécimal
- ✓ Transférer le fichier vers le Kit à travers une connexion série et le logiciel HyperTerminal de Windows.

### **4. Matériels utilisés :**

- ✓ Le Kit MTS-86C
- ✓ Un PC doté d'un port de communication série.

### **5. logiciels :**

- ✓ Microsoft Windows
- ✓ Hyper Terminal

### **6. Connexion du MTS-86C avec PC et le chargement des programmes:**

1. Pour connecter MTS-86C à un PC, connecter le port parallèle fourni à MTS-86C. Connecter l'extrémité de type COM1 au port COM1 du PC. Mettre le module MTS-86C on marche.
2. Aller au START>program files>accessoires>communications>Hyper Terminal
3. Sélectionner COM1 comme port.
4. Cliquer ok tab. Sélectionner bit rate au 19200.
5. sélectionner contrôle de débit (Xon/Xoff)
6. Appuyer sur n'importe touche entre (A-F) pour faire la communication entre MTS-86C et PC.

### **7. Ecrire un code et créer un fichier hexadécimal :**

1. Le 8086 ne peut pas exécuter un fichier texte simple. il peut exécuter un fichier hexadécimal donc un fichier texte doit se convertir en **hex** , Cela se fait dans les étapes suivantes.
2. Ecrire ton code dans un fichier **WordPad** avec les instructions nécessaires par exemple :

```
CODE    SEGMENT
        ASSUME CS :CODE ,DS : CODE ,ES :CODE ,SS :CODE
        ORG    1000H
        MOV    AX,123AH
        MOV    BX,1254H
        ADD    AX,BX
        HLT
CODE    ENDS
        END
```

3. Enregistrer le fichier sous le nom **P1.asm** par exemple dans un répertoire nommé **UP** dans la partition C.
4. Aller au **cmd** de l'invite de commande et accéder au répertoire **UP** et taper : **v P1** (pour la commande v soit active il faut copier dans le répertoire **UP** quelques fichiers exécutables nécessaires de CD ROM du Kit)
5. Tu peux voir l'exécution en cours.
6. Donner le nom de fichier **bin** comme **P1.bin** sinon taper **P1.asm** puis **P1.bin**
7. Taper entrée quand l'adresse est demandée.
8. Ton fichier **hex** a été créé tu peux vérifier la conversion dans le répertoire **UP** vous allez trouver **P1.hex** .

#### **6. Chargement du fichier (hex) dans MTS-86C:**

1. Appuyer sur **L** dans la fenêtre de l'HyperTerminal.
2. Cliquez (to Transfer) dans le coin supérieur de l'onglet droit. Sélectionner (send text file).
3. Sélectionner ton fichier **hex** .
4. Taper **g (GO)** et taper entrée.
5. Ton programme sera exécuté.
6. Taper **R** pour voir les résultats dans les registres.

## **TP 02 : Initiation à la programmation en langage machine** **8086**

### **1- Objectifs :**

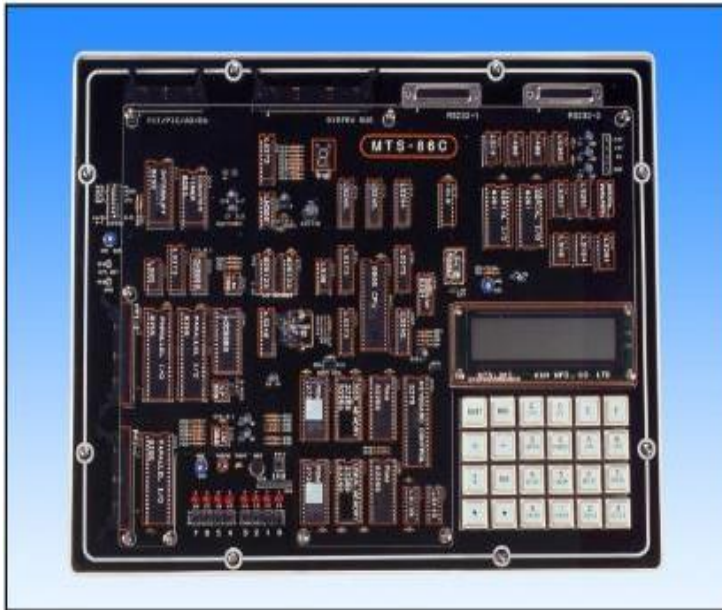
L'objectif de cette manipulation est de rappeler l'étudiant les notions de la programmation. On procèdera par la suite à introduire la procédure d'utilisation du Kit MTS-86C en utilisant seulement les touches du clavier pour saisir des programmes simples. A la fin de la manipulation l'étudiant sera capable de :

- ✓ Ecrire un programme en langage machine
- ✓ Utiliser le clavier du Kit MTS-86C
- ✓ Comparer l'utilisation du langage machine et du langage assembleur.

### **2- Matériels utilisés :**

- ✓ Saisir un programme en langage machine

Le Kit MTS-86C.



### **3- Le clavier du kit MTS-86C**

#### **3.1- Les touches de fonctions**

TOUCHES	DESCRIPTION
RESET	La touche RESET vous permet d'initialiser la carte MTS-86C. Quand la touche est pressée, le 8086 envoie un message sur le LCD et le monitor est prêt.
NMI	La touche NMI génère une interruption non masquable de type 2
+	La touche + vous permet d'additionner 2 valeurs hexadécimales
-	La touche – vous permet de substituer une valeur hexadécimale d'une autre.
:	Cette touche vous permet de séparer une adresse entrée sur deux Parties ; la valeur de segment ou de l'offset
REG	Cette touche vous permet d'utiliser les contenus des registres du 8086 comme une adresse ou une donnée en entrée
,	Cette touche sépare les entrées saisies à partir des touches du clavier et incrémente l'adresse mémoire à chaque entrée.
.	Lorsque cette touche est appuyée la commande courante est exécutée. Notez que lorsque la touche GO est appuyée, le 8086 lance l'exécution à partir de l'adresse spécifiée.

### 3.2- Les touches hexadécimales

Touche Hexadécimale	Commande		Registre	
	Acronyme	Nom	Acronyme	Nom
0 EB/AX	EB	Examine par Octet	AX	Accumulateur
2 GO/CX	ER	Examine le registre	BX	Base
3 ST/DX	GO	Go	CX	Compteur
4 IB/SP	ST	Un pas (STEP)	DX	Données
5 OB/BP	IB	Entrée Octet	SP	Stacker Pointer
6 MV/SI	OB	Sortie Octet	BP	Base Pointer
7 EW/DI	MV	Move	SI	Source Index
8 IW/CS	EW	Examine par Word	DI	Destination Index
9 OW/DS	IW	Entrée Word	CS	Code Segment
A /SS	OW	Sortie Word	DS	Data Segment
B /ES	non	N/A	SS	Stack Segment
C /IP	non	N/A	ES	Extra Segment
D /FL	non	N/A	IP	Pointeur d'instruction
E	non	N/A	FL	Flag
F	non	N/A	non	N/A

#### 4- Procédures

1. utiliser la commande EB pour examiner le contenu de la mémoire pour les adresses suivantes : 0060:0000 à 0060:0009.

\_\_\_\_\_

\_\_\_\_\_

2. entrer les données : 01, 11, 21,31, ....., 91 dans les cases mémoires 0060:0000, 0060:0001, ..., 0060:0009 respectivement.

3. Répéter l'étape 1.

\_\_\_\_\_

\_\_\_\_\_

4. utiliser la commande EW pour examiner le contenu de la mémoire de 0040:0000 à 0040:0008.

\_\_\_\_\_

5. utiliser la commande ER pour examiner le contenu de s registres AX, BX, CX, DX, et FL

AX \_\_\_\_\_ BX \_\_\_\_\_ FL \_\_\_\_\_ CX \_\_\_\_\_ DX \_\_\_\_\_

6. remplacer le contenu des registres AX, BX, CX, et DX par : 1211, 00BA, BCA0, et 1423 respectivement.

7. Répéter l'étape 5.

AX \_\_\_\_\_ BX \_\_\_\_\_ FL \_\_\_\_\_ CX \_\_\_\_\_ DX \_\_\_\_\_

8. Le programme suivant échange le contenu des cases mémoire dont l'offset est 0200 et 0201.

Entrez le programme dans la case mémoire d'adresse 0060: 0000

0060:0000      8C CA 8E DA BB 00 02 8A 07

8A 57 01 88 47 01 88 17 F4

A. Entrer la donné AC dans la case 0060:0200 et FA dans la case 0060:0201.

B. utiliser la commande GO pour exécuter le programme.

C. examiner le contenu des cases 0060:0200 et 0060:0201 .

0060:0200 \_\_\_\_\_

0060:0201 \_\_\_\_\_

9. Entrer la donnée A0 dans la case 0060:0200 et 22 dans la case 0060:0201. utiliser la commande ST pour exécuter le programme et remplir le tableau suivant :

steps	AL	DL	0060:0200	0060:0201
1				
2				
3				
4				
5				
6				
7				

**10-** exécuter le programme en utilisant le logiciel “**hyper terminal**”

**Req :**

- 1- Utiliser l’instruction **G=seg : off** pour exécuter le programme.
- 2- Utiliser l’instruction **E adresse** pour examiner et changer le contenu de la mémoire.
- 3- Utiliser l’instruction **D adresse** pour examiner le contenu de 70 cases mémoire.
- 4- Utiliser l’instruction **R** pour examiner le contenu des différents registres.

## **TP 03 : Boucle LOOP**

### **1- Objectifs**

- ✓ Ecrire un programme en langage assembleur 8086
- ✓ Ecrire un programme en langage machine 8086
- ✓ Utiliser le clavier du Kit MTS-86C
- ✓ Saisir un programme en langage machine
- ✓ Comparer l'utilisation du langage machine et du langage assembleur.

### **2- Matériels utilisés :**

- Le Kit MTS-86C.
- Emulateur 8086

### **2- Programmation en langage machine :**

**Programme 01** : Soit le programme en assembleur suivant :

```
MOV AX , 4F
MOV CX, 16
MOV BL, 0
B 1: SHR AX , 1
      JNC Next
      INC BL
Next: LOOP B1
      MOV[1200h],BL
      HLT
```

1. Dressez un organigramme expliquant le fonctionnement de ce programme.
2. Emulez le programme et enregistrez les valeurs des registres BX et CX à chaque instruction.
3. On vous donne l'équivalent en langage machine du programme ci-dessus :

Adresse	Langage machine
0000	B8 4F 00
0003	B9 10 00
0006	B3 00
0008	D1 E8
000A	73 02
000C	FE C3
000E	E2 F8
0010	88 1E 00 12
0014	F4

1. Insérez le programme ci-dessus depuis l'adresse 00H.
2. Enregistrez les valeurs des registres AX et CX à chaque instruction.
3. Exécutez le programme complet avec le bouton [GO] et vérifiez vos résultats

**Programme 02 :**

Soit le programme écrit en langage machine suivant :

B8 54 12

BB 67 45

8B C8

8B C3

8B D9

F4

1. Insérez le programme à l'adresse 100H : 0H.
2. Enregistrez les valeurs du registre AX et BX à chaque instruction.
3. Que fait ce programme ?
4. Donnez son équivalent en assembleur.
5. Vérifiez vos résultats et conclure.



### **Références bibliographiques:**

1. M. Aumiaux, L'emploi des microprocesseurs, Masson, Paris, 1982.
2. M. Aumiaux, Les systèmes à microprocesseurs, Masson, Paris, 1982.
3. Zanella, Architecture et technologie des ordinateurs, Dunod.
4. B. Brey, Intel microprocessors 8086/8088, 80186/80188, 80286, 80386, Prentice Hall, 2009.
5. J.L. Hennessy ; Architecture des ordinateurs : Une approche quantitative, Ediscience.
6. A.S. Tanenbaum, Architecture de l'ordinateur, Dunod.
7. Joseph Haggège ; Microprocesseur, support de cours, l'ISSET de Rades, 2003.
8. Alain Cazes, Joëlle Delacroix, Architecture Des Machines et Des Systèmes Informatiques, 3e édition, 2008
- 9- N.Boukhennoufa, systèmes à microprocesseur : cours et exercices, support de cours université de msila

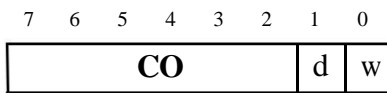
## 1- CODE MACHINE DES INSTRUCTIONS

Une instruction peut comporter de 1 à 7 octets dont 1 ou 2 octets pour coder l'opération, les autres servent à définir les opérandes.

Dans le cas le plus général, l'instruction se fait entre un registre et une case mémoire. Dans le code machine de l'instruction, on trouvera le code de l'opération (CO), le code du registre utilisé (REG), le code du mode d'adressage utilisé (MOD) et le code permettant de déterminer l'adresse de la case mémoire (ADR):

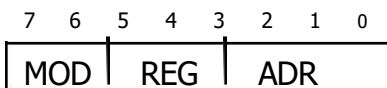
► Le premier octet est un octet optionnel qui représente un préfix qui peut être un préfix de répétition ou un préfix de changement de segment.

► Le 2<sup>ème</sup> octet dit Code Opération se présente comme suit :



- **CO** : C'est le code proprement dit de l'instruction
- **d** : désigne la destination du résultat  
     d = 0 → Résultat dans mémoire ou opération entre 2 registres  
     d = 1 → Résultat dans registre
- **w** : Opération 8 bits ou 16 bits  
     w = 0 → 8 bits  
     w = 1 → 16 bits

► Le 3<sup>ème</sup> octet permet de définir les opérandes



- ✓ **MOD** : Ce champ de 2 bits nous informe sur le mode d'adressage : registre, directe ou la nature du déplacement dans les autres cas,
- ✓ **REG** : Ce champ de 3 bits désigne le registre constituant un opérande
- ✓ **ADR** : Ce champs de 3 bits précise l'adresse de l'autre opérande quand il s'agit d'une position mémoire.
- ✓ Pour une opération entre deux registres R □ R :  
     REG = Registre source  
     ADR = Registre destination

► Les octets suivants concernent :

- Les déplacements sur 8 ou 16 bits utilisés dans le calcul d'adresse
- Les données sur 8 ou 16 bits dans le cas de l'adressage immédiat
- ...

## 5.1 LES CODES REG, ADR ET MOD

REG			
w = 1		w = 0	
000	AX	000	AL
001	CX	001	CL
010	DX	010	DL
011	BX	011	BL
100	SP	100	AH
101	BP	101	CH
110	SI	110	DH
111	DI	111	BH
REG Segment			
00	01	10	11
ES	CS	SS	DS

ADR	
000	BX+SI+d
001	BX+DI+d
010	BP+SI+d
011	BP+DI+d
100	SI+d
101	DI+d
110	- BP+d - Direct
111	BX+d

MOD	
00	- Directe
	- Indirecte avec dep = 0
01	Indirect déplacement <b>court</b> : 8 bits, ≤127, 1 ou 2 chiffres hex
10	Indirect dép. <b>long</b> : 16 bits, > 128 , + de 2 chiffres hex
11	R ← R ou R ← im Dans ce cas : ADR= code registre destination

### Préfix de changement de segment

ES	CS	SS	DS
26h	2Eh	36h	3Eh

## 5.2 TABLEAU DES CODES BINAIRES

<b>AAA</b>	0011 0111			
<b>AAD</b>	1101 0101	0000 1010		
<b>AAM</b>	1101 0100	0000 1010		
<b>AAS</b>	0011 1111			
<b>ADC</b> • R/M ↔ R • R/M ← im • AL/AX ← im	0001 00dw 1000 00sw 0001 010w	MOD REG ADR MOD 010 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>ADD</b> • R/M ↔ R • R/M ← im (**) • AL/AX ← im	0000 00dw 1000 00sw 0000 010w	MOD REG ADR MOD 000 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>AND</b> • R/M ↔ R • R/M ← im • Ac ← im	0010 00dw 1000 000w 0010 010w	MOD REG ADR MOD 100 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>BOUND</b>	0110 0010	MOD REG ADR	(Adr ou dep)	
<b>CALL</b> • intra segment direct • intra segment indirect • inter segment direct • inter segment indirect	1110 1000 1111 1111 1001 1010	adresse MOD 010 ADR Adresse	(Dep) Segment	
<b>CBW</b>	1001 1000			
<b>CLC</b>	1111 1000			
<b>CLD</b>	1111 1100			
<b>CLI</b>	1111 1010			
<b>CMC</b>	1111 0101			
<b>CMP</b> • R/M ↔ R • R/M ← im • AL/AX ← im	0011 10dw 1000 00sw 0011 110w	MOD REG ADR MOD 111 ADR Donnée	(Adr ou dep) (Adr ou dep)	donnée

<b>CMPS</b>	1010 011w			
<b>CWD</b>	1001 1001			
<b>DAA</b>	0010 0111			
<b>DAS</b>	0010 1111			
<b>DEC</b> • R/M 8 bits • R 16 bits	1111 111w 01001 REG	MOD 001 ADR	(Adr ou dep)	
<b>DIV R/M</b>	1111 011w	MOD 110 ADR	(Adr ou dep)	
<b>ENTER</b>	1100 1000	donnée		
<b>EDC</b>				
<b>HLT</b>	1111 0100			
<b>IDIV R/M</b>	1111 011w	MOD 111 ADR	(Adr ou dep)	
<b>IMUL R/M</b>	1111 011w	MOD 101 ADR	(Adr ou dep)	
<b>IN</b> • Port défini • Port dans DX	1110 010w 1110 110w	Port		
<b>INC</b> • R/M 8 bits • R 16 bits	1111 111w 01000 REG	MOD 000 ADR	(Adr ou dep)	
<b>INS</b>	0110 110w			
<b>INT</b>	1100 1101	Num. interruption		
<b>INTO</b>	1100 1110			
<b>IRET</b>	1100 1111			
<b>JA</b>	0111 0111	DepRel_8		
<b>JAE</b>	0111 0011	DepRel_8		
<b>JB</b>	0111 0010	DepRel_8		
<b>JBE</b>	0111 0110	DepRel_8		
<b>JC</b>	1110 0010	DepRel_8		
<b>JCXZ</b>	1110 0011	DepRel_8		
<b>JE</b>	0111 0100	DepRel_8		
<b>JG</b>	0111 1111	DepRel_8		
<b>JGE</b>	0111 1101	DepRel_8		
<b>JL</b>	0111 1100	DepRel_8		
<b>JLE</b>	0111 1110	DepRel_8		
<b>JMP</b> • intra segment direct • intra segment direct court • intra segment indirect • inter segment direct • inter segment indirect	1110 1001 1110 1011 1111 1111 1110 1010 1111 1111	Dep_Relatif_16 Dep_Relatif_8. MOD 100 ADR Adr. branch.16 MOD 101 ADR	(jmp etiq) (jmp short etiq)  SegL	SegH
<b>JNC</b>	0111 0011	DepRel_8		
<b>JNE</b>	0111 0101	DepRel_8		
<b>JNO</b>	0111 0001	DepRel_8		
<b>JNS</b>	0111 1001	DepRel_8		
<b>JNP</b>	0111 1011	DepRel_8		
<b>JO</b>	0111 0000	DepRel_8		
<b>JP</b>	0111 1010	DepRel_8		
<b>JS</b>	0111 1000	DepRel_8		
<b>LAHF</b>	1001 1111			
<b>LDS</b>	1100 0101	MOD REG ADR	(Adr ou dep)	

<b>LEA</b>	10001101	MOD REG ADR	(Adr ou dep)	
<b>LEAVE</b>	11001001			
<b>LES</b>	1100 0100	MOD REG ADR	(Adr ou dep)	
<b>LOCK</b>	1111 0000			
<b>LODS</b>	1010 110w			
<b>LOOP</b>	1110 0010	DepRel_8		
<b>LOOPZ</b>	1110 0001	DepRel_8		
<b>LOOPNZ</b>	1110 0000	DepRel_8		
<b>MOV</b> • R/M $\leftrightarrow$ R • m $\leftarrow$ im • R $\leftarrow$ im • AX/AL $\leftarrow$ M_direct • M_direct $\leftarrow$ AX/AL • Rseg $\leftarrow$ R/M • R/M $\leftarrow$ Rseg	1000 10dw 1100 011w 1011 w REG 1010 000w 1010 001w 1000 1110 1000 1100	MOD REG ADR MOD 000 ADR donnée Adresse Adresse MOD 0 Rseg ADR MOD 0 Rseg ADR	(Adr ou dep) (Adr ou dep)    (Adr ou dep) (Adr ou dep)	donnée
<b>MOVS</b>	1010 010w			
<b>MUL R/M</b>	1111 011w	MOD 100 ADR	(Adr ou dep)	
<b>NEG</b>	1111 011w	MOD 011 ADR	(Adr ou dep)	
<b>NOP</b>	1001 0000			
<b>NOT</b>	1111 011w	MOD 010 ADR	(Adr ou dep)	
<b>OR</b> • R/M $\leftrightarrow$ R • R/M $\leftarrow$ im • AX/AL $\leftarrow$ im	0000 10dw 1000 000w 0000 110w	MOD REG ADR MOD 001 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>OUT</b> • port défini • port dans DX	1110 011w 1110 111w	port		
<b>OUTS</b>	0110 111w			
<b>POP</b> • M • R • Rseg	1000 1111 0101 1REG 000REG111	MOD 000 ADR	(Adr ou dep)	
<b>POPA (*)</b>	0100 0001			
<b>POPF</b>	1001 1101			
<b>PUSH</b> • M • R • Rseg	1111 1111 01010 REG 000REG110	MOD 000 ADR	(Adr ou dep)	
<b>PUSHA (*)</b>	0110 0000			
<b>PUSHF</b>	1001 1100			
<b>RCL</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 010 ADR MOD 010 ADR MOD 010 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>RCR</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 011 ADR MOD 011 ADR MOD 011 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>REP/REPZ, REPNZ</b>	1111 001z			

<b>RET</b> • intra segment • inter segment	1100 0011 1100 1011			
<b>ROL</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 000 ADR MOD 000 ADR MOD 000 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>ROR</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 001 ADR MOD 001 ADR MOD 001 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>SAHF</b>	1001 1110			
<b>SAL/SHL</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 100 ADR MOD 100 ADR MOD 100 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>SAR</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 111 ADR MOD 111 ADR MOD 111 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>SBB</b> • R/M $\leftrightarrow$ R • R/M $\leftarrow$ im • AL/AX $\leftarrow$ im (*)	0001 10dw 1000 00sw 0001 110w	MOD REG ADR MOD 011 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>SCAS</b>	1010 111w			
<b>SHR</b> • R/M,1 • R/M,CL • R/M,im8 (*)	1101 000w 1101 001w 1100 000w	MOD 101 ADR MOD 101 ADR MOD 101 ADR	(Adr ou dep) (Adr ou dep) (Adr ou dep)	Donnée_8
<b>STC</b>	1111 1001			
<b>STD</b>	1111 1101			
<b>STI</b>	1111 1011			
<b>STOS</b>	1010 101w			
<b>SUB</b> • R/M $\leftrightarrow$ R • M $\leftarrow$ im (**) • R $\leftarrow$ im (**) • AL/AX $\leftarrow$ im	0010 10dw 1000 00sw 1000 00sw 0010 110w	MOD REG ADR MOD 101 ADR 11 101 REG donnée	(Adr ou dep) (Adr ou dep) donnée	donnée
<b>TEST</b> • R/M $\leftrightarrow$ R • R/M $\leftarrow$ im • AL/AX $\leftarrow$ im	1000 010w 1111 011w 1010 100w	MOD REG ADR MOD 000 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée
<b>WAIT</b>	1001 1011			
<b>XCHG</b> • R/M $\leftrightarrow$ R • R $\leftrightarrow$ AX	1000 011w 10010 REG	MOD REG ADR	(Adr ou dep)	
<b>XLAT</b>	1101 0111			
<b>XOR</b> • R/M $\leftrightarrow$ R • R/M $\leftarrow$ im • AL/AX $\leftarrow$ im	0011 00dw 1000 000w 0011 010w	MOD REG ADR MOD 110 ADR donnée	(Adr ou dep) (Adr ou dep)	donnée

- Les champs entre ( ) sont présents dans le cas de l'adressage direct [aaaa] ou de l'adressage indirect avec déplacement [R+dep] ou [Rb + Ri + dep]
- Un champ adresse est toujours constitué de 2 octets : AdrL AdrH
- Un champ de donnée peut être de 1 ou de 2 octets selon l'instruction, DL suivie éventuellement de DH

(\*) Ces instructions ne tournent pas sur le 8086 mais sur les processeurs qui l'on suivi

(\*\*) s=1 dans le cas R/M<sub>16</sub> ↔ im<sub>8</sub>, une extension de signe 8 bits vers 16 bits est effectués sur la donnée immédiate avant l'opération.

## Exemples :

mov ax , 3456h  
R<sub>16</sub> ← im<sub>16</sub>  
1011 w REG donnée  
1011 1 000 5634  
B8 56 34

Mov bx , 56h  
R<sub>16</sub> ← im<sub>16</sub>  
1011 w REG donnée  
1011 1 011 56 00  
BB 56 00

Mov DX , [123h]  
R<sub>16</sub> ← M  
1000 10dw MOD REG ADR adresse  
1000 1011 00 010 110 2301  
8B 16 23 01

Mov [SI + 146h] , BL  
M ← R<sub>8</sub>  
1000 10dw MOD REG ADR deplong  
1000 1000 10 011 100 4601  
88 9C 46 01

mov AX , [3456h]  
AX ← Mdirect  
1010 000w adresse  
1010 0001 5634  
A1 56 34

mov CL , [BP+SI]  
R<sub>8</sub> ← M  
1000 10dw MOD REG ADR  
1000 1010 00 001 010  
8A 0A

Mov bl, 56h  
R<sub>8</sub> ← im<sub>8</sub>  
1011 w REG donnée  
1011 0 011 56  
B3 56

Mov AX , BX  
R<sub>16</sub> ← R<sub>16</sub>  
1000 10dw MOD REG ADR  
1000 1001 11 011 000  
89 D8

Mov [BX+DI + 46h] , CX  
M ← R<sub>16</sub>  
1000 10dw MOD REG ADR depcourt  
1000 1001 01 001 001 46  
89 49 46

AND BL , 38h  
R<sub>8</sub> ← im  
1000 000w MOD 100 ADR donée  
1000 0000 11 100 011 38  
80 E3 38