

# Architecture des Ordinateurs

---

- Support Cours -  
SMI - S4

Hatim  
KHARRAZ AROUSSI

# SOMMAIRE

<b>CHAPITRE 1 : INTRODUCTION GENERALE .....</b>	<b>6</b>
1. Objectifs .....	6
2. Historique.....	6
3. Schéma classique d'un ordinateur .....	7
4. Types d'informations et numérisation .....	8
4.1. Texte.....	8
4.2. Image .....	8
4.3. Son et vidéo .....	9
4.4. Unités de mesure .....	10
<b>CHAPITRE 2 : ARCHITECTURE DE BASE .....</b>	<b>11</b>
1. Modèle de Von Neumann .....	11
2. Unité Centrale.....	11
3. Mémoire Principale.....	12
4. Interfaces d'Entrées / Sorties .....	12
5. Bus .....	12
6. Décodage d'adresses.....	13
<b>CHAPITRE 3 : MEMOIRES.....</b>	<b>14</b>
1. Introduction .....	14
2. Organisation d'une mémoire.....	14
3. Schéma de circuit mémoire .....	15
4. Cycle d'une opération en mémoire .....	15
5. Caractéristiques d'une mémoire.....	15
6. Chronogramme d'un cycle de lecture .....	16
7. Types de mémoires .....	16
7.1. Mémoire vive RAM .....	16
7.2. Mémoire morte ROM.....	17
8. Hiérarchie mémoire .....	17
8.1. Notions de base .....	17

8.2.	Schéma de la hiérarchie mémoire .....	18
<b>CHAPITRE 4 : MICROPROCESSEURS.....</b>		<b>19</b>
<b>1.</b>	<b>Introduction .....</b>	<b>19</b>
<b>2.</b>	<b>Architecture interne d'un microprocesseur .....</b>	<b>19</b>
2.1.	Bloc de registres .....	20
2.1.1.	Compteur Ordinal.....	20
2.1.2.	Registre d'adresse .....	21
2.1.3.	Registre d'instruction .....	21
2.1.4.	Registre temporaire .....	21
2.1.5.	Registre de données (Accumulateur) .....	21
2.1.6.	Registre d'état .....	22
2.2.	Bloc de calcul .....	22
2.3.	Bloc logique de commande .....	22
<b>3.</b>	<b>Fonctionnement d'un microprocesseur.....</b>	<b>23</b>
3.1.	Phase de recherche .....	23
3.2.	Phase de décodage et recherche d'opérande .....	24
3.3.	Phase d'exécution.....	24
<b>4.</b>	<b>Jeu d'instructions .....</b>	<b>25</b>
4.1.	Codage.....	25
4.2.	Temps d'exécution .....	25
<b>5.</b>	<b>Performances d'un microprocesseur.....</b>	<b>25</b>
<b>6.</b>	<b>Architecture CISC et RISC .....</b>	<b>26</b>
6.1.	Architecture CISC .....	26
6.2.	Architecture RISC .....	26
<b>7.</b>	<b>Amélioration de l'architecture de base .....</b>	<b>27</b>
7.1.	Architecture Pipeline.....	27
7.2.	Gain de performance .....	28
7.3.	Notion de cache mémoire.....	29
7.4.	Architecture superscalaire .....	29
7.5.	Architecture pipeline et superscalaire .....	30
<b>CHAPITRE 5 : MICROPROCESSEUR 8086 .....</b>		<b>31</b>
<b>1.</b>	<b>Description physique du 8086 .....</b>	<b>31</b>
<b>2.</b>	<b>Schéma fonctionnel du 8086 .....</b>	<b>31</b>
<b>3.</b>	<b>Description et utilisation des signaux du 8086.....</b>	<b>32</b>
<b>4.</b>	<b>Organisation interne du 8086.....</b>	<b>33</b>
4.1.	Registres internes .....	34
4.1.1.	Registres généraux .....	34
4.1.2.	Registres de pointeurs et d'index .....	34

4.1.3.	Pointeur d'instruction et flag .....	35
4.1.4.	Registres de segments .....	35
<b>5.</b>	<b>Segmentation de la mémoire .....</b>	<b>36</b>
<b>6.</b>	<b>Microprocesseur 8088 .....</b>	<b>36</b>
<b>CHAPITRE 6 : ASSEMBLEUR DU 8086 .....</b>		<b>38</b>
<b>1.</b>	<b>Introduction .....</b>	<b>38</b>
<b>2.</b>	<b>Format d'une instruction.....</b>	<b>38</b>
<b>3.</b>	<b>Mode d'adressage .....</b>	<b>39</b>
3.1.	Adressage direct .....	40
3.2.	Adressage indirect .....	41
3.2.1.	Adressage basé sur BX.....	41
3.2.2.	Adressage relatif avec BX.....	42
3.2.3.	Adressage indexé.....	42
3.2.4.	Adressage indexé avec BX.....	43
<b>4.</b>	<b>Déclaration de variables .....</b>	<b>43</b>
<b>5.</b>	<b>Instructions de transfert .....</b>	<b>43</b>
5.1.	Transfert de données .....	43
5.2.	Transfert entre le microprocesseur et la pile .....	44
<b>6.</b>	<b>Instructions arithmétiques .....</b>	<b>45</b>
6.1.	Addition.....	45
6.2.	Soustraction .....	47
6.3.	Multiplication .....	47
6.3.1.	Multiplication pour les opérandes de 8 bits .....	48
6.3.2.	Multiplication pour les opérandes de 16 bits .....	48
6.4.	Division .....	48
6.4.1.	Division avec une source de 8 bits .....	49
6.4.2.	Division avec une source de 16 bits .....	49
6.5.	Incrémentation.....	49
6.6.	Décrémentation .....	50
6.7.	Comparaison.....	50
<b>7.</b>	<b>Instructions logiques .....</b>	<b>51</b>
7.1.	Opérateurs logiques AND, OR, XOR .....	51
7.2.	Opérateur de complément à 1 .....	51
7.3.	Opérateur de complément à 2.....	52
7.4.	Opérateurs de décalage et rotation .....	52
7.4.1.	Décalage arithmétique à gauche.....	52
7.4.2.	Décalage arithmétique à droite.....	53
7.4.3.	Décalage logique à gauche .....	53
7.4.4.	Décalage logique à droite .....	53
7.4.5.	Rotation circulaire à gauche .....	53
7.4.6.	Rotation circulaire à droite .....	54

<b>8. Instructions de branchement.....</b>	<b>54</b>
8.1. Saut inconditionnel .....	54
8.2. Saut conditionnel .....	55
8.3. Boucles .....	56
8.4. Sous programme.....	57
8.5. Appel de sous programme.....	57
<b>9. Interruptions.....</b>	<b>60</b>
9.1. Interruptions matérielles .....	60
9.2. Interruptions logicielles .....	61
9.3. Vecteurs d'interruptions .....	61
9.4. Interruption du MS-DOS.....	62
9.4.1. Affichage à l'écran .....	62
9.4.2. Saisie du calvier .....	63
9.4.3. Arrêt de programme .....	63
<b>TD N°1 – MEMOIRE ET MICROPROCESSEUR – .....</b>	<b>66</b>
Exercice 1 .....	66
Exercice 2.....	66
Exercice 4.....	67
Exercice 5.....	67
Exercice 6.....	67
Exercice 7 .....	67
<b>TP N°1 – TURBO DEBUGGER –.....</b>	<b>68</b>
<b>TP N°2 – PROGRAMMATION EN ASSEMBLEUR – .....</b>	<b>71</b>
<b>TP N°3 – PROCEDURES EN ASSEMBLEUR –.....</b>	<b>72</b>

# *Chapitre 1 : Introduction générale*

## 1. Objectifs

- Connaitre la structure d'un ordinateur.
- Comprendre le rôle des composants de l'ordinateur.
- Assimiler les opérations élémentaires d'exécution d'une instruction.
- Apprendre la programmation en assembleur.

## 2. Historique

génération	date approximative	technologie	vitesse (opérations/s)
1	1946-1957	tube à vide	40 000
2	1958-1964	transistor	200 000
3	1965-1971	SSI/MSI	1 000 000
4	1972-1977	LSI	10 000 000
5	1978-	VLSI	100 000 000

- ✓ SSI – Small Scale Integration
- ✓ MSI – Medium Scale Integration
- ✓ LSI – Large Scale integration
- ✓ VLSI – Very Large Scale Integration

### ➤ 1946-1957

- Ordinateurs dédiés, exemplaire unique.
- Machines volumineuses et peu fiables.
- Technologie à lampes, relais, résistances.
- $10^4$  éléments logiques.
- Programmation par cartes perforées.

### ➤ 1958-1964

- Usage général, machine fiable.
- Technologie à transistors.
- $10^5$  éléments logiques.
- Apparition des langages de programmation évolués (COBOL, FORTRAN, LISP).

### ➤ 1965-1971

- Technologie des circuits intégrés (S/MSI small/medium scale integration).
- $10^6$  éléments logiques.
- Avènement du système d'exploitation complexe, des mini-ordinateurs.

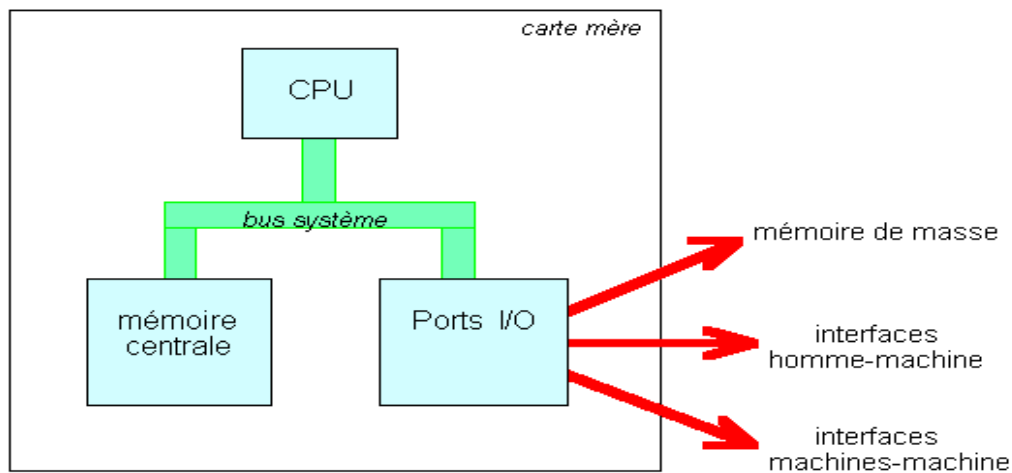
➤ **1972-1977**

- Technologie LSI (large SI).
- $10^7$  éléments logiques.
- Avènement de réseaux de machines.
- Traitement distribué/réparti.

➤ **1978 -**

- Technologie VL/WSI (very large, wafer).
- $10^8$  éléments logiques (le PII contient 7,5 millions de transistors, mémoire non comprise).
- Systèmes distribués interactifs.
- Multimédia, traitement de données non numériques (textes, images, paroles)
- Parallélisme massif.

### 3. Schéma classique d'un ordinateur



#### **CPU : Central Processing Unit**

- commande tout le système,
- fait des calculs arithmétiques,
- fait des opérations logiques,
- lit/écrit en mémoire, ou dans un port.

#### **Mémoire centrale :**

1. **RAM** : On peut y mémoriser des informations et les relire plus tard.
2. **ROM** : Est une mémoire qui ne peut pas être modifiée.

#### **Interfaces :**

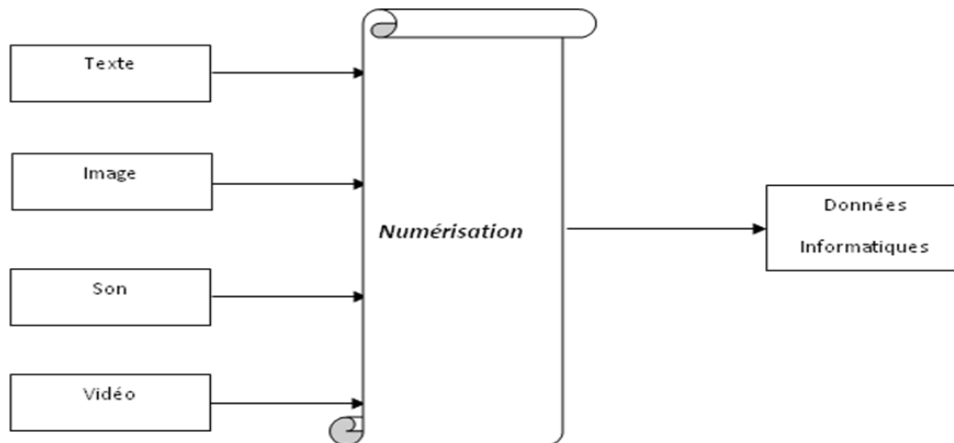
- Le CPU les utilise comme des mémoires. Ils permettent à la carte mère de dialoguer avec les dispositifs externes (mémoire de masse, clavier, écran, modem, ...).

#### **Bus :**

- permettent de transférer des données entre les composants

## 4. Types d'informations et numérisation

- On distingue 4 types d'informations :
  - Texte, image, son et vidéo.
  - Chaque type d'information nécessite une technique particulière de numérisation.



### 4.1. Texte

- On distingue 2 types de textes :
  - **Texte numérique** : la numérisation consiste à écrire ce nombre dans la base 2 (→ vers la base 2).
  - **Texte non numérique** :
    - Caractères, symboles spéciaux (\*, \$, ?, ...), touche clavier (espace, tab, ...)
    - La numérisation est basée sur un codage interne comme le code ASCII.
    - Etablir des correspondances entre le texte non numérique et des suites binaires.

### 4.2. Image

- Une image numérique est décrite par un ensemble de lignes, chaque ligne en un ensemble de points (**Résolution**) :
  - Image avec résolution 640\*480 (=480 lignes et 640 points par ligne).
- On distingue 3 types d'images :
  - En noir et blanc, en 256 nuances de gris, en couleur.

1. Image noir et blanc :



- Chaque point est codé sur 1 bit (0 pour noir et 1 pour blanc).
- Une image de  $640 \times 480 = 300\text{Kbits}$ .

2. Image en 256 nuances de gris :

- Chaque point est représenté par un octet (256 couleurs entre le noir et le blanc).
- Une image de  $640 \times 480 = 2,4\text{Mbits}$ .

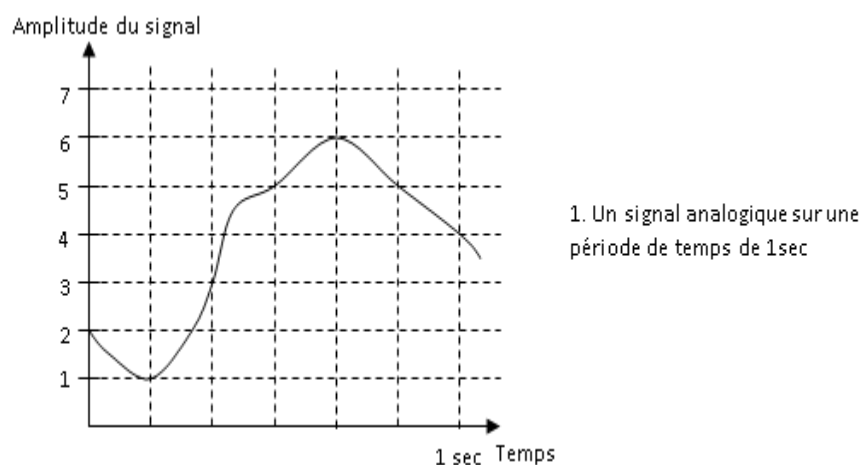
3. Image en couleurs :

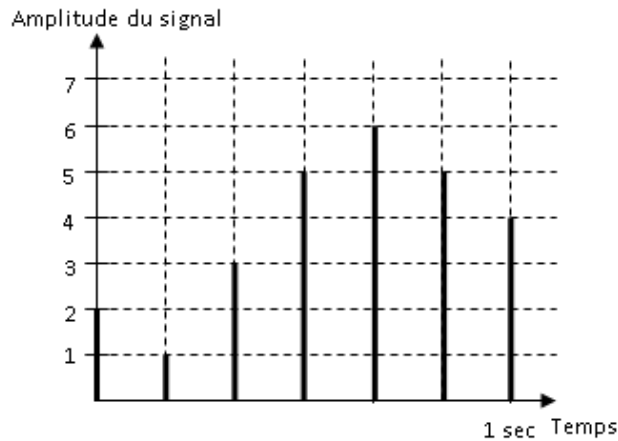
- Chaque couleur est une combinaison binaire de 3 couleurs de base (RBV).
- Une couleur  $X$  s'exprime par :  $X = aR + bB + cV$ .
- $a, b, c$  sont des doses de couleurs de base allant de 0 à 255.
- Une image de  $640 \times 480 = 7,37\text{Mbits}$ .

### 4.3. Son et vidéo

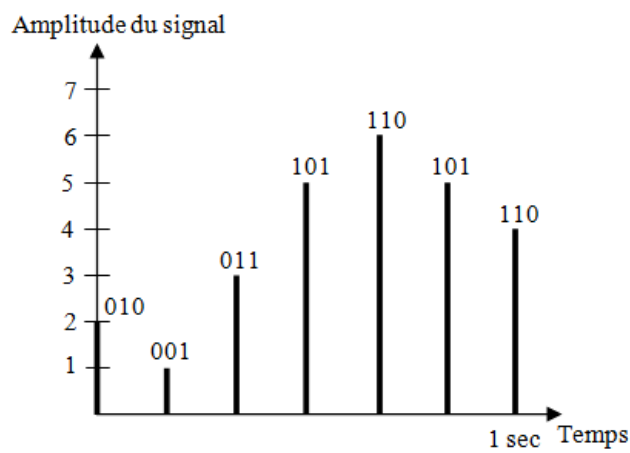
- Le son et la vidéo sont des données continues en fonction du temps.
- Ces données sont de types analogiques (forme sinusoidale).
- Les données analogiques sont numérisées par la technique **d'échantillonnage**.
- Echantillonnage : le signal sur une période de temps (1 seconde) :
  - Est divisée en plusieurs échantillons.
  - On obtient une séquence de mesures.
  - Ces mesures sont codées en binaire.

Exemple d'échantillonnage :





2. Des mesures de l'amplitude du signal sur des périodes de temps (échantillons)



3. Chaque valeur est transformée en sa conversion binaire. La suite de ces conversions est la numérisation de cette donnée :  
010001011101110101110

## 4.4. Unités de mesure

### ➤ Unités de mesure de capacité

- **Kilo** =  $10^3 \approx 2^{10} = 1024$
- **Méga** =  $10^6 \approx 2^{20} = 1\,048\,576$
- **Giga** =  $10^9 \approx 2^{30} = 1\,073\,741\,824$
- **Tera** =  $10^{12} \approx 2^{40} = 1\,099\,511\,627\,776$
- **Peta** =  $10^{15} \approx 2^{50} = 1\,125\,899\,906\,842\,624$

### ➤ Unités de mesure de temps

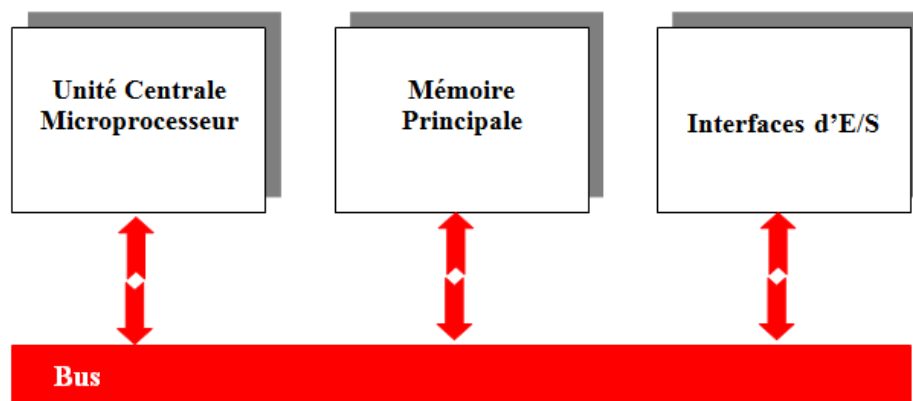
- **ms** = milliseconde =  $10^{-3} \text{ s} = 0,001 \text{ s}$
- **µs** = microseconde =  $10^{-6} \text{ s} = 0,000\,0001 \text{ s}$
- **ns** = nanoseconde =  $10^{-9} \text{ s} = 0,000\,000\,001 \text{ s}$
- **ps** = picoseconde =  $10^{-12} \text{ s} = 0,000\,000\,000\,001 \text{ s}$

## Chapitre 2 : Architecture de base

### 1. Modèle de Von Neumann

➤ *Le microprocesseur ne suffit pas pour traiter des informations :*

- 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 sert de base à la plupart des systèmes à microprocesseur actuel. Elle est composée des éléments suivants :
  - Une unité centrale (microprocesseur).
  - Une mémoire principale.
  - Des interfaces d'entrées / sorties.
- Les différents organes du système sont reliés par des liaisons de communication appelées **Bus**.



### 2. Unité Centrale

- Elle est composée du microprocesseur chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire ou de communiquer avec les unités d'échange.
- Toutes les activités du microprocesseur sont cadencés par une horloge.
- Le microprocesseur se caractérise par :
- Sa fréquence d'horloge.
  - 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.

### 3. Mémoire Principale

- Elle contient des instructions des programmes en cours d'exécution et les données associés à ces programmes.
- Physiquement, elle se décompose en :
  - **Mémoire morte (ROM)** : chargée de stocker le **BIOS** (Basic Input Output System, Système d'Entrées/Sorties de base) du système d'exploitation, il est donc responsable de la gestion du matériel. Ces données ne sont pas perdues à la mise hors tension. C'est une mémoire à lecture seule.
  - **Mémoire vive (RAM)** : 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.

### 4. Interfaces d'Entrées / Sorties

- Elles permettent d'assurer la communication entre le microprocesseur et les périphériques :
  - Capteur,
  - Clavier,
  - Moniteur ou afficheur,
  - Imprimante
  - Modem,
  - Carte réseau
  - etc

### 5. Bus

- Un bus est un dispositif destiné à assurer le transfert simultané d'informations entre les divers composants d'un ordinateur.
- C'est un ensemble de câbles transportant des signaux électriques
- On distingue 3 catégories de bus :
  - Bus de données,
  - Bus d'adresses,
  - Bus de commandes.
- **Bus de données** :
  - Véhicule des instructions et des données à traiter.
  - Il est bidirectionnel.
  - Son nombre de broches correspond à la capacité de traitement du microprocesseur.

➤ **Bus d'adresses :**

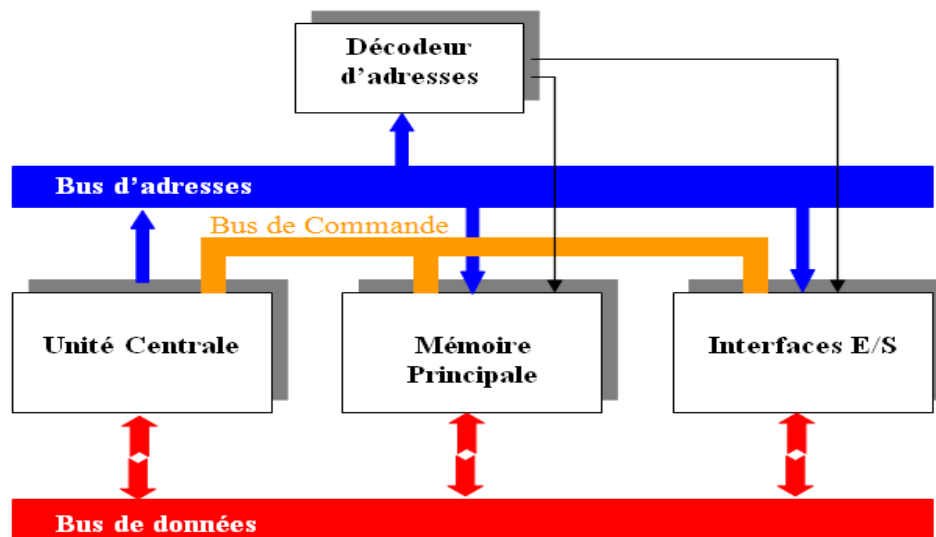
- Permet au microprocesseur d'adresser les cases mémoire et les interfaces d'entrées / sorties.
- Il est unidirectionnel.
- Le nombre de broches détermine la capacité maximale d'adressage ( $2^n$ ).

➤ **Bus de commandes :**

- Transporte les ordres et les signaux de synchronisation entre le microprocesseur et les autres composants.
- Permet d'envoyer les requêtes associées avec l'envoi des données et des adresses dans les autres bus (exemple une écriture lors du transfert entre le microprocesseur et la mémoire).

## 6. Décodage d'adresses

- La multiplicité des périphériques nécessite la présence d'un décodeur d'adresse chargé d'aiguiller les données existants dans le bus de données.
- Le microprocesseur peut communiquer avec les différentes mémoires et les différents périphériques. 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.
- Ainsi, on a besoin d'un décodeur d'adresses qui doit attribuer à chaque périphérique une zone d'adresses spécifique.



## Chapitre 3 : Mémoires

### 1. Introduction

- Une mémoire est un circuit intégré permettant d'enregistrer, de conserver et de restituer des informations (instructions et données).
- Deux opérations sont possibles en mémoire :
  - Ecriture, lorsqu'on enregistre des informations en mémoire.
  - Lecture, lorsqu'on récupère des informations précédemment enregistrées.

### 2. Organisation d'une mémoire

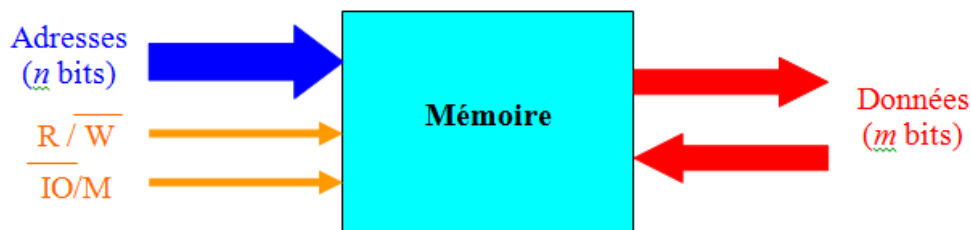
- Avec une adresse de  $n$  bits, il est possible de référencer au plus  $2^n$  cases mémoire.

Adresse	Case mémoire
7 = 111	
6 = 110	
5 = 101	
4 = 100	
3 = 011	
2 = 010	
1 = 001	
0 = 000	0001 0110

- 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.
- Chaque case est remplie par un mot de données (sa longueur  $m$  est toujours une puissance de 2).

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

### 3. Schéma de circuit mémoire



- Composé des :
  - Entrées d'adresse.
  - Entrées de données.
  - Sorties de données.
  - Entrées de commandes :
    - Une entrée de sélection de lecture ou écriture ( $R/\overline{W}$ )
    - Une entrée de sélection du circuit ( $\overline{IO}/M$ )

### 4. Cycle d'une opération en mémoire

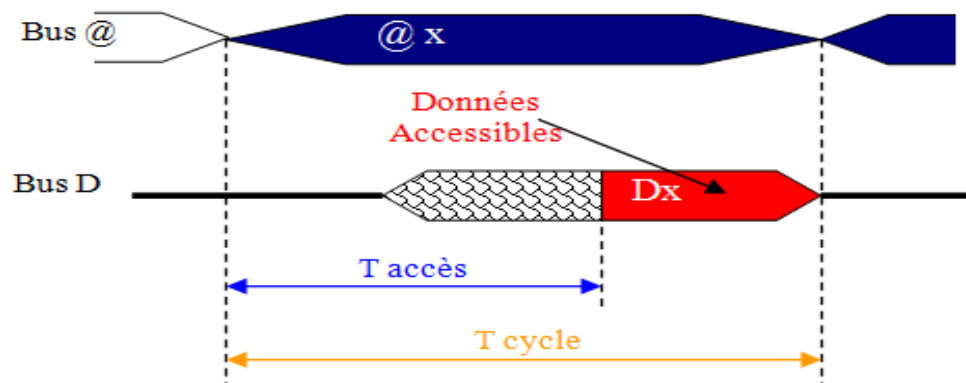
- Une opération de lecture ou écriture de la mémoire suit toujours le même cycle :
  1. Sélection de l'adresse.
  2. Choix de l'opération à effectuer ( $R/\overline{W}$ )
  3. Sélection de la mémoire ( $\overline{IO}/M = 1$ )
  4. Lecture ou écriture de la donnée.

### 5. Caractéristiques d'une mémoire

- **Capacité** : Nombre total de bits que contient la mémoire (en octets).
- **Format des données** : Nombre de bits que l'on peut mémoriser par case mémoire.

- **Temps d'accès** : Temps écoulé entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
- **Temps de cycle** : Intervalle minimum qui sépare deux demandes successives de lecture ou d'écriture.
- **Débit** : nombre maximum d'informations lues ou écrites par seconde.
- **Volatilité** : 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.

## 6. Chronogramme d'un cycle de lecture



## 7. Types de mémoires

### 7.1. Mémoire vive RAM

- Sert au stockage temporaire des données.
- Doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur.
- Il existe deux familles de mémoires **RAM** :
  - **RAM statique (SRAM)** :
    - le bit mémoire contient entre 4 et 6 transistors.
    - Les **SRAM** ont un temps d'accès plus rapide que les **DRAM**.
    - Les **SRAM** sont utilisées pour les mémoires de petite taille comme les caches et les registres.



- **RAM dynamique (DRAM) :**

- le bit mémoire contient un seul transistor.
- Est moins cher que la **SRAM**.
- Doit être rafraîchie régulièrement pour entretenir la mémorisation.
- La **DRAM** offre une densité d'information et un coût par bit plus faible est utilisé comme mémoire centrale.

## 7.2. Mémoire morte ROM

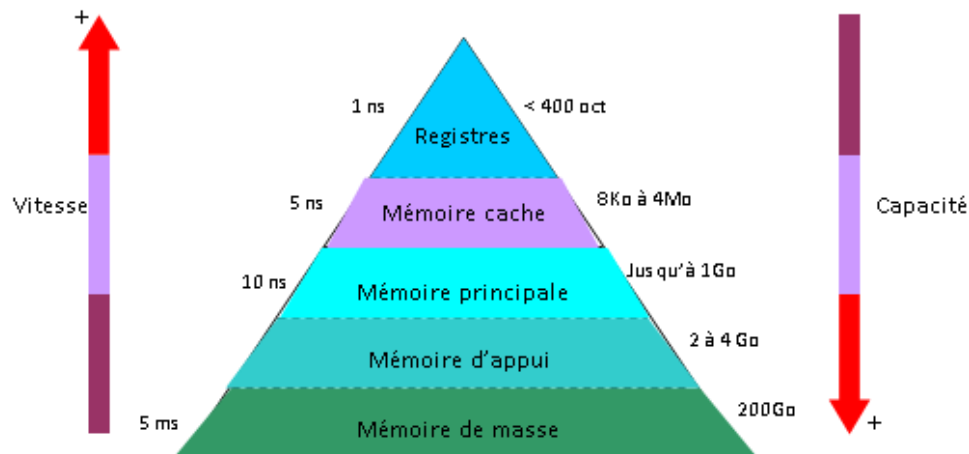
- Est non volatile.
- Peut être programmée.
- La méthode de programmation varie selon le type :
  - **ROM** : est programmée par le fabricant et son contenu ne peut être ni modifié ni effacé.
  - **PROM** : peut être programmée une seule fois par l'utilisateur.
  - **EPROM** : est une PROM qui peut être effacée.
  - **Flash EPROM** : est une mémoire programmable et effaçable électriquement. Elle peut être reprogrammée mot par mot.

## 8. Hiérarchie mémoire

### 8.1. Notions de base

- Une mémoire idéale est une mémoire de grande capacité, capable de stocker un maximum d'informations et possédant un temps d'accès très faible.
- Une telle mémoire n'existe pas. Ainsi, une mémoire de grande capacité est très lente et une mémoire rapide est très chère.
- Le microprocesseur est capable de traiter des informations très rapidement (3 GHZ) et utilise une mémoire relativement lente (400 MHZ).
- Or le microprocesseur n'a pas besoin de toutes les informations au même moment.
- Afin d'obtenir un compromis coût-performance, on définit une hiérarchie mémoire où:
  - On utilise des mémoires très rapides pour stocker les informations dont le microprocesseur se sert le plus et inversement.

## 8.2. Schéma de la hiérarchie mémoire



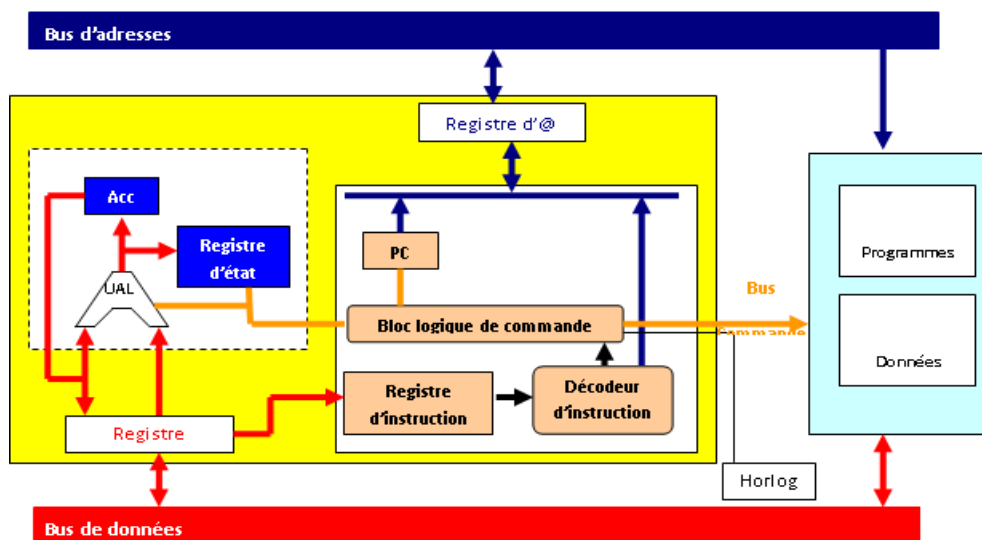
- **Registre** : élément mémoire le plus rapide situé au niveau du processeur et sert au stockage des opérandes et des résultats intermédiaires.
- **Mémoire cache** : est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- **Mémoire principale** : est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.
- **Mémoire d'appui** : sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- **Mémoire de masse** : est une mémoire périphérique de grande capacité utilisée pour le stockage permanent des informations. Elle utilise pour cela des supports magnétiques (**disque dur**) ou optiques (**CDROM, DVDROM**).

## Chapitre 4 : Microprocesseurs

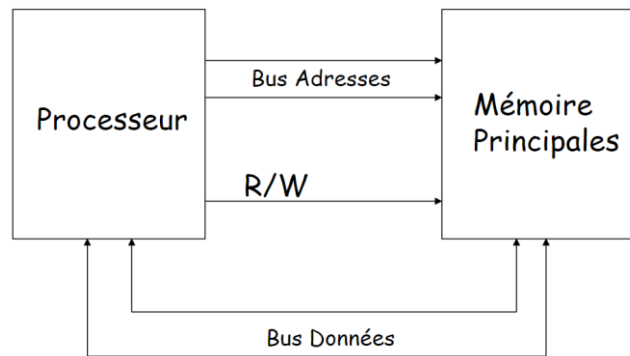
### 1. Introduction

- Un microprocesseur est un circuit électronique destiné à traiter des valeurs numériques.
- Le fonctionnement du microprocesseur est basé sur l'architecture de Von Neumann, soit un bus d'adresses, un bus de données et un bus de contrôle.

### 2. Architecture interne d'un microprocesseur



- Un programme est une suite d'instructions et un ensemble de données stockés dans des cases mémoire :
  - Chaque case mémoire est repérée par une adresse.
  - L'exécution d'un programme par le microprocesseur consiste donc à :
    - Lire les instructions dans la mémoire et les stocker dans le microprocesseur.
    - Décoder et exécuter ces instructions. Selon la nature de l'instruction, le microprocesseur peut effectuer une lecture de données à traiter, une écriture de données en mémoire.



- Pour effectuer ces opérations, le microprocesseur fait appel à des blocs fondamentaux :

- Bloc des registres.
- Bloc de calcul UAL.
- Bloc logique de commande.

## 2.1. Bloc de registres

- Un registre est un ensemble de bits qui permet de stocker une information binaire dans un microprocesseur.
- L'information peut être une instruction, une donnée ou une adresse.
- Ainsi, il existe six registres fondamentaux qu'on trouve dans tous les microprocesseurs :
  - Compteur d'instruction ou **compteur ordinal**.
  - Registre d'adresse.
  - Registre d'instruction.
  - Registres de données (Accumulateur).
  - Registre d'état.
  - Registre temporaire.

### 2.1.1. Compteur Ordinal

- Format d'une instruction : *Code\_Opération Code\_Opérande*
  - Le code opération spécifie la nature de l'opération à effectuer.
  - Le code opérande indique au microprocesseur les données qui vont être traitées par le code opération.
- Le compteur ordinal est un registre qui permet de localiser les instructions stockées en mémoire :

- Il est chargé au début de l'exécution d'un programme par l'adresse de la première case mémoire du premier code opération de la première instruction.
- Puisque l'exécution d'un programme s'effectue de façon séquentielle, le compteur ordinal s'incrémente automatiquement pour passer de l'adresse d'une case mémoire à l'adresse suivante.

### **2.1.2. Registre d'adresse**

- Ce registre est directement relié au compteur ordinal, il représente une interface entre ce compteur et le bus d'adresse du microprocesseur.
- Ce registre a le même nombre de bits que le compteur ordinal.

### **2.1.3. Registre d'instruction**

- Connaissant les adresses des cases mémoire où se trouve l'instruction, le microprocesseur va effectuer une lecture qui a pour objectif la recherche de l'instruction.
- Cette instruction sera chargée dans le registre d'instruction.
- Ce registre va garder cette instruction pendant le temps nécessaire à son décodage et son exécution.

### **2.1.4. Registre temporaire**

- L'unité arithmétique et logique possède deux entrées pour recevoir les données à traiter, au niveau de chaque entrée il y a un registre temporaire qui garde la donnée pendant son traitement.

### **2.1.5. Registre de données (Accumulateur)**

- L'accumulateur est un registre qui est relié :
  - D'un côté à un registre temporaire et au bus interne de données.
  - De l'autre côté à la sortie de l'unité arithmétique et logique (UAL).
- Si une opération traite deux données, alors :
  - Une donnée sera stockée d'abord dans l'accumulateur, ensuite elle passera au premier registre temporaire.
  - La deuxième donnée passera directement au deuxième registre temporaire.
  - Le résultat de l'opération est stocké dans l'accumulateur en sortant de l'UAL.
- L'accumulateur représente également un intermédiaire entre le microprocesseur et les interfaces d'E/S.

### 2.1.6. Registre d'état

- Ce registre est directement relié à l'UAL.
- Il stocke des résultats particuliers des opérations effectuées.
- Les bits les plus couramment utilisés sont :
  - **Bit de retenue** : il est dans l'état actif lorsque le huitième bit du résultat de l'opération génère une retenue.
  - **Bit de zéro** : ce bit est actif si le résultat de l'opération est nul.
  - **Bit de signe** : ce bit est actif si le bit le plus significatif du contenu de l'accumulateur est 1.
  - **Bit de débordement** : ce bit est actif en cas de dépassement de capacité maximale de codage.

## 2.2. Bloc de calcul

- Les traitements effectués par l'unité arithmétique et logique sont :
  - Opérations arithmétiques :
    - Addition
    - Soustraction
    - Multiplication
    - division
  - Opérations logiques :
    - Et logique
    - Ou logique
    - Ou exclusif

## 2.3. Bloc logique de commande

- Ce bloc a pour fonction :
  - Décodage des instructions stockées dans le registre d'instruction.
  - Exécution des instructions par les organes exécutifs (UAL, Accumulateur, CO, ...) :
    - Incrémentation du CO.
    - Mise à jour des bits du registre d'état en fonction des résultats fournis par l'UAL.
    - Nature de l'opération qui va être effectuée par l'UAL.

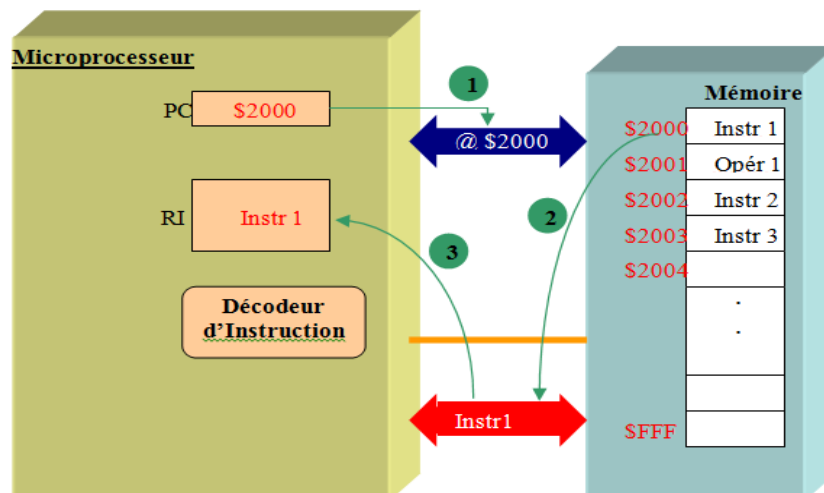
- Gestion des dialogues entre le microprocesseur, la mémoire et les interfaces d'E/S. Ceci se fait par le moyen du bus de commande et les signaux de contrôle :
  - **INT** : interruption d'exécution d'un programme (l'E/S veut communiquer avec le microprocesseur).
  - **INTA** : réponse du microprocesseur au signal INT.
  - **DMA** : Accès direct à la mémoire (Liaison E/S-mémoire).

### 3. Fonctionnement d'un microprocesseur

- L'exécution d'un microprocesseur se fait séquentiellement à partir du premier mot de la première instruction.
- Pour chaque mot d'une instruction, le microprocesseur effectue les phases suivantes :
  - Phase de recherche.
  - Phase de décodage et recherche d'opérande.
  - Phase d'exécution.

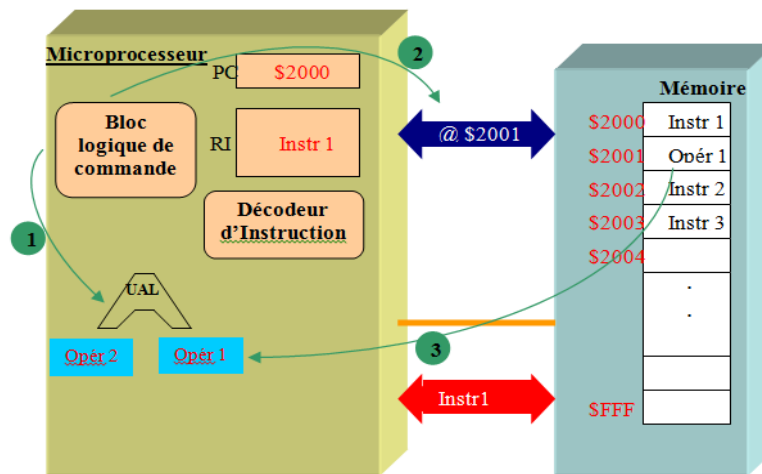
#### 3.1. Phase de recherche

- Le but de cette phase est de chercher le contenu d'un mot d'une instruction de la mémoire vers le registre d'instruction.
- Elle s'effectue de la manière suivante :
  1. Le compteur ordinal contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.
  2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
  3. L'instruction est stockée dans le registre d'instruction du processeur.



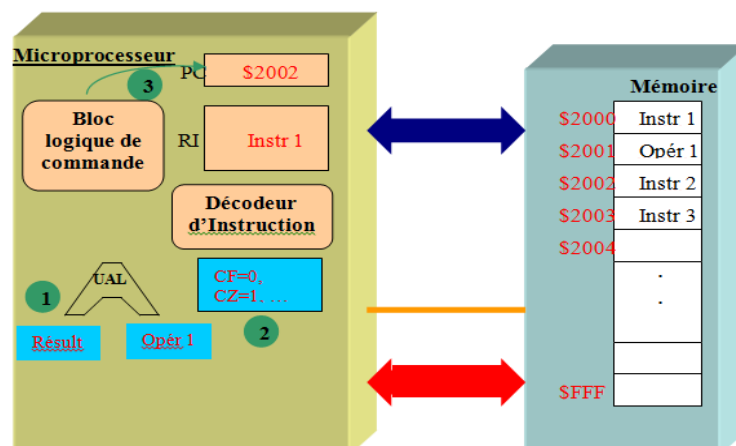
### 3.2. Phase de décodage et recherche d'opérande

- Le registre d'instruction contient le premier mot de l'instruction.
- Ce mot contient le code opération qui définit la nature de l'opération à effectuer et le nombre de mots de l'instruction. Ainsi :
  1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
  2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
  3. L'opérande est stocké dans un registre.



### 3.3. Phase d'exécution

1. Le micro-programme réalisant l'instruction est exécuté.
2. Les flags sont positionnés (registre d'état).
3. L'unité de commande positionne le CO pour l'instruction suivante.





## 4. Jeu d'instructions

- Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter.
- Il représente l'aspect programmable du microprocesseur (son fonctionnement).
- Ce jeu d'instruction doit respecter une certaine syntaxe, appelée syntaxe du langage de programmation : **Langage Assembleur**.

### 4.1. Codage

- Les instructions et leurs opérandes sont stockés en mémoire principale.
- La taille totale d'une instruction dépend du type d'instruction et aussi du type d'opérande.
- Chaque instruction est toujours codée sur un nombre entier d'octets afin de faciliter son décodage par le processeur.
- Une instruction est composée de deux champs :
  - Le code instruction : indique au processeur quelle instruction réaliser.
  - Le champ opérande : contient la donnée ou la référence à une donnée en mémoire.

Code Instruction	Code Opérande
10010011	00111110

### 4.2. Temps d'exécution

- Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer.
- Le nombre de cycles dépend de la complexité de l'instruction.
- Il est plus lent d'accéder à la mémoire principale qu'à un registre du processeur.
- La durée d'un cycle dépend de la fréquence d'horloge du séquenceur.

## 5. Performances d'un microprocesseur

- On peut caractériser la puissance d'un microprocesseur par le nombre d'instructions qu'il est capable de traiter par seconde. Pour cela, on définit :

- Le **CPI** (Cycle Par Instruction) qui représente le nombre moyen de cycles d'horloge nécessaire pour l'exécution d'une instruction pour un microprocesseur donné.
- Le **MIPS** (Millions d'Instructions Par Seconde) qui représente la puissance de traitement du microprocesseur :  $MIPS = \frac{F_H}{CPI}$  , avec  $F_H$  en MHz.
- Pour augmenter les performances d'un microprocesseur, on peut donc soit augmenter la fréquence d'horloge (limitation matérielle), soit diminuer le CPI (choix d'un jeu d'instruction adapté).

## 6. Architecture CISC et RISC

- Actuellement l'architecture des microprocesseurs se compose de deux grandes familles :
  - L'architecture **CISC** (Complex Instruction Set Computer)
  - L'architecture **RISC** (Reduced Instruction Set Computer)

### 6.1. Architecture CISC

- CISC est l'architecture la plus ancienne, était la seule envisageable pour les machines à microprocesseur.
- Vu que la mémoire travaillait très lentement par rapport au processeur, il était plus intéressant de soumettre au microprocesseur des instructions complexes.
- Coder une opération complexe par plusieurs instructions plus petites entraînerait des accès mémoire (lent).
- Ajouter au jeu d'instructions du microprocesseur une instruction complexe qui se chargerait de réaliser cette opération.
- 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.

### 6.2. Architecture RISC

- Des études statistiques menées aux années 1970 ont montré que les programmes générés par les compilateurs se contentaient le plus souvent d'affectations, d'additions et de multiplications par des constantes.
- Ainsi, 80% des traitements des langages de haut niveau faisaient appel à seulement 20% des instructions du microprocesseur.
- D'où l'idée de réduire le jeu d'instructions à celles le plus couramment utilisées et d'en améliorer la vitesse de traitement.

- C'est donc une architecture dans laquelle les instructions sont en nombre réduit.
- 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'exécute ainsi en un cycle d'horloge.

## 7. Amélioration de l'architecture de base

- L'ensemble des améliorations des microprocesseurs visent à diminuer le temps d'exécution du programme :
- La première idée 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.

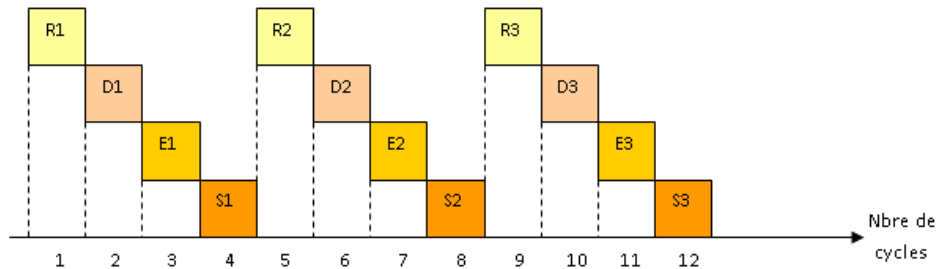
### 7.1. Architecture Pipeline

- L'exécution d'une instruction est décomposée en une succession d'étapes et chaque étape correspond à l'utilisation d'une des fonctions du microprocesseur. Lorsqu'une instruction se trouve dans l'une des étapes, les composants associés aux autres étapes ne sont pas utilisés. Le fonctionnement d'un microprocesseur simple n'est donc pas efficace.
- L'architecture pipeline permet d'améliorer l'efficacité du microprocesseur. En effet, lorsque la première étape de l'exécution d'une instruction est achevée, l'instruction entre dans la seconde étape de son exécution et la première phase de l'exécution de l'instruction suivante débute.
- Il peut donc y avoir une instruction en cours d'exécution dans chacune des étapes et chacun des composants du microprocesseur peut être utilisé à chaque cycle d'horloge.

- Une machine pipeline se caractérise par le nombre d'étapes utilisées pour l'exécution d'une instruction, on appelle aussi ce nombre d'étapes le nombre d'étages du pipeline.
- Exemple d'une exécution en 4 phases :



- Modèle classique :



- Modèle pipeliné :

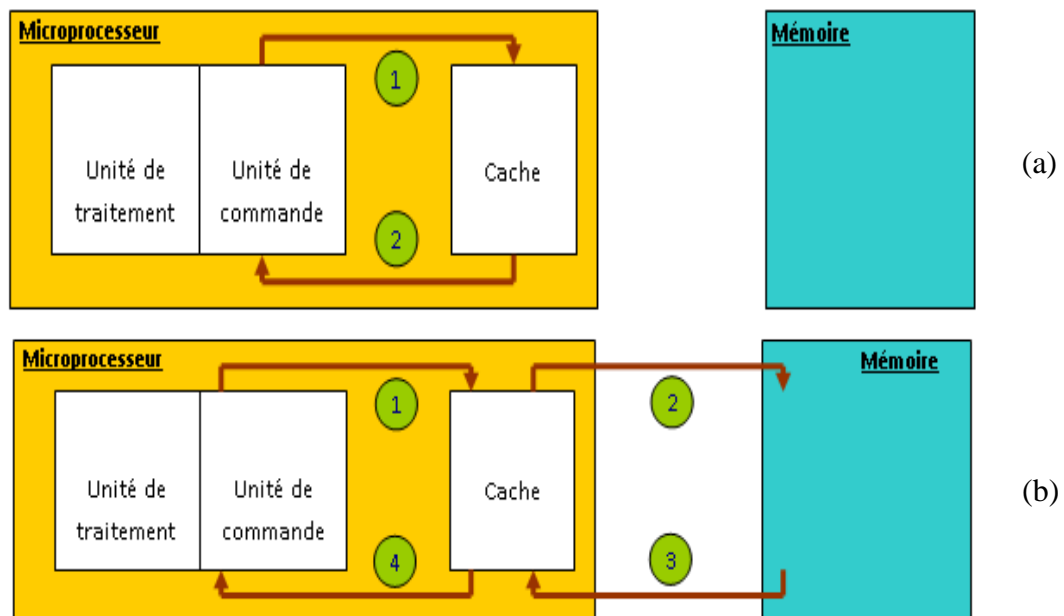


## 7.2. Gain de performance

- Dans cette structure, la machine débute l'exécution d'une instruction à chaque cycle et le pipeline est pleinement occupé à partir du quatrième cycle.
- Le gain obtenu dépend donc du nombre d'étages du pipeline.
- En effet, pour exécuter **n** instructions, en supposant que chaque instruction s'exécute en **k** cycles d'horloge, il faut :
  - **n.k** cycles d'horloge pour une exécution séquentielle.
  - **k** cycles d'horloge pour exécuter la première instruction puis **n-1** cycles pour les **n-1** instructions suivantes si on utilise un pipeline de **k** étages.
- ➔ Donc lorsque le nombre **n** d'instructions à exécuter est grand par rapport à **k**, on peut admettre qu'on divise le temps d'exécution par **k**.

### 7.3. Notion de cache mémoire

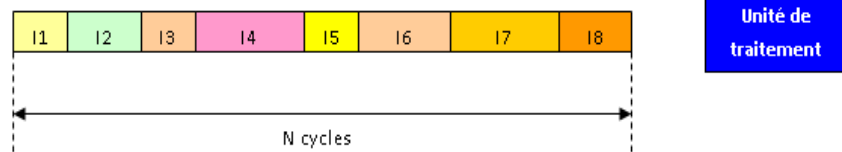
- Depuis le début des années 80, une des solutions utilisées pour masquer cette latence est de disposer une mémoire très rapide entre le microprocesseur et la mémoire. Elle est appelée **cache mémoire**.
- On compense ainsi la faible vitesse relative de la mémoire en permettant au microprocesseur d'acquérir les données à sa vitesse propre.
- Au départ cette mémoire était intégrée en dehors du microprocesseur mais elle fait maintenant partie intégrante du microprocesseur.
- Le principe de cache est très simple : le microprocesseur n'a pas conscience de sa présence et lui envoie toutes ses requêtes comme s'il agissait de la mémoire principale:
  - Soit la donnée ou l'instruction requise est présente dans le cache et elle est alors envoyée directement au microprocesseur. On parle de succès de cache.
  - (a)
  - Soit la donnée ou l'instruction n'est pas dans le cache, et le contrôleur de cache envoie alors une requête à la mémoire principale. Une fois l'information récupérée, il la renvoie au microprocesseur tout en la stockant dans le cache. On parle de **défaut de cache**. (b)



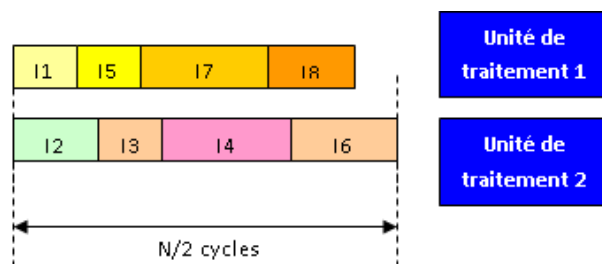
### 7.4. Architecture superscalaire

- Une autre façon de gagner en performance est d'exécuter plusieurs instructions en même temps.

- L'approche superscalaire consiste à doter le microprocesseur de plusieurs unités de traitement travaillant en parallèle.
- Les instructions sont alors réparties entre les différentes unités d'exécution.
- Architecture scalaire :

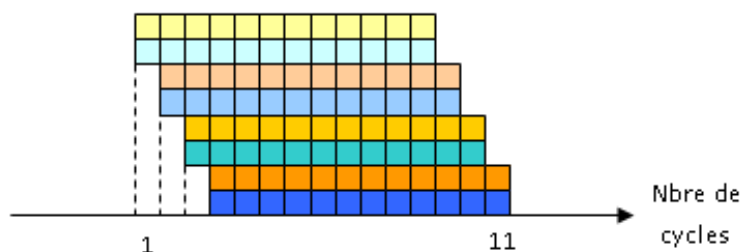


- Architecture superscalaire :



## 7.5. Architecture pipeline et superscalaire

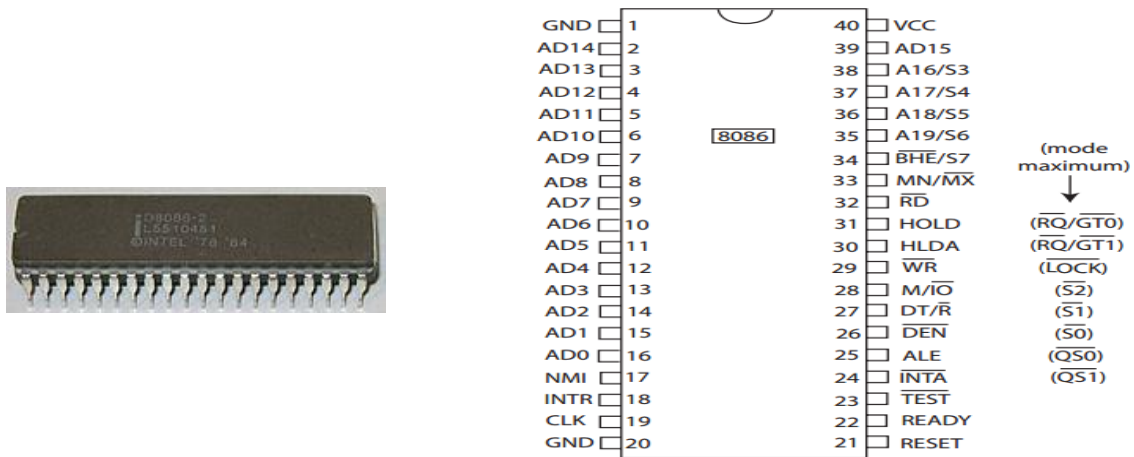
- Le principe est d'exécuter les instructions de façon pipeliné dans chacune des unités de traitement travaillant en parallèle.



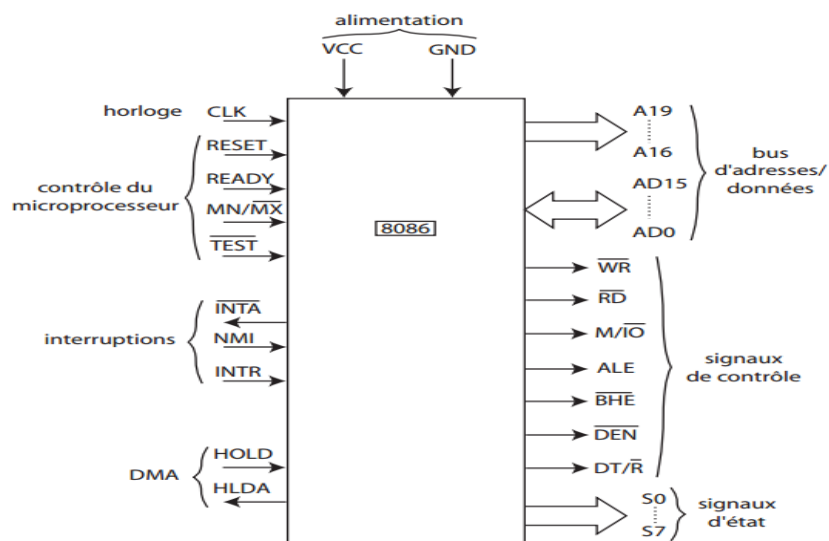
## Chapitre 5 : Microprocesseur 8086

### 1. Description physique du 8086

- Le microprocesseur Intel 8086 est un microprocesseur 16 bits.
- Il est apparu en 1978.
- C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...).
- Il se présente sous la forme d'un boîtier à 40 broches.



### 2. Schéma fonctionnel du 8086



### 3. Description et utilisation des signaux du 8086

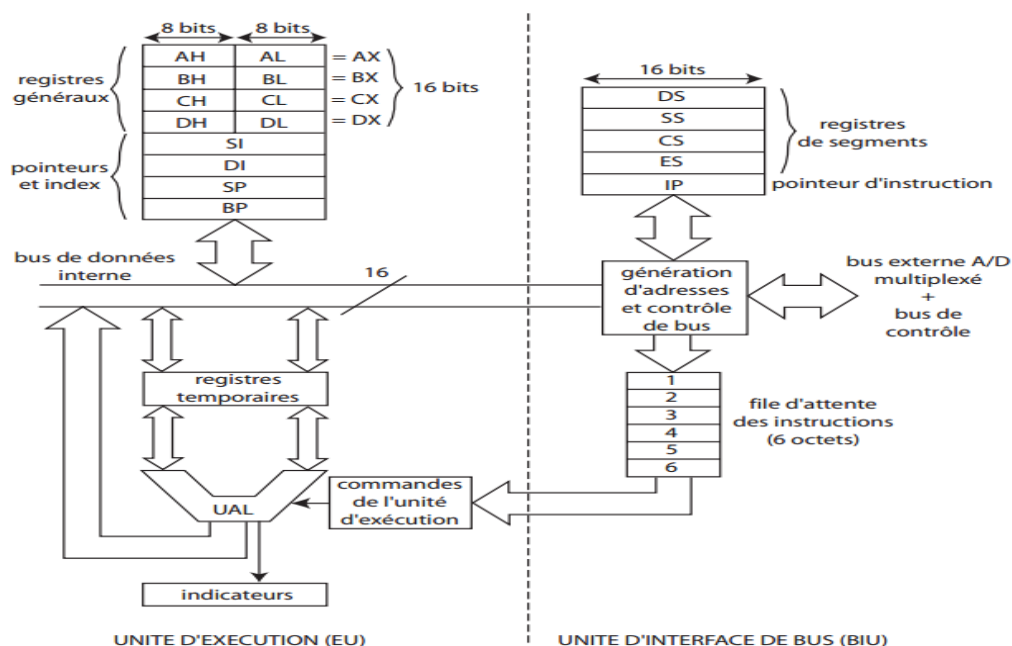
- **CLK** : entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un générateur d'horloge : le 8284.
- **RESET** : entrée de remise à zéro du microprocesseur. Lorsque cette entrée est mise à l'état haut pendant au moins 4 périodes d'horloge, le microprocesseur est réinitialisé : il va exécuter l'instruction se trouvant à l'adresse FFFF0H (adresse de bootstrap). Le signal de RESET est fourni par le générateur d'horloge.
- **READY** : entrée de synchronisation avec la mémoire. S'il est égal à 1, mémoire ou interface prête pour une lecture ou écriture. Ce signal provient également du générateur d'horloge.
- $MN/\overline{MX}$  : entrée de choix du mode de fonctionnement du microprocesseur :
  - Mode minimum (1) : le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande (RD, RW, ...).
  - Mode maximum (0) : ces signaux de commande sont produits par un contrôleur de bus, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs.
- $\overline{TEST}$  : entrée de mise en attente du microprocesseur d'un événement extérieur.
  - **NMI et INTR** : entrées de demande d'interruption. **INTR** : interruption normale,
  - **NMI (Non Maskable Interrupt)** : interruption prioritaire.
- $\overline{INTA}$  (**Interrupt Acknowledge**) : indique que le microprocesseur accepte l'interruption.
- **HOLD et HLDA** : signaux de demande d'accord d'accès direct à la mémoire (DMA).
- **S0 à S7** : signaux d'état indiquant le type d'opération en cours sur le bus.
- $\overline{RD}$  (**READ**) : signal de lecture d'une donnée.
- $\overline{WR}$  (**WRITE**) : signal d'écriture d'une donnée.
- $M/\overline{IO}$  (**Memory / Input-Output**) : indique si le 8086 adresse la mémoire ( $M/\overline{IO}=1$ ) ou les entrées / sorties ( $M/\overline{IO}=0$ ).
- **ALE** : validité des bus d'adresses et de données (0 : bus de données).
- $\overline{DNE}$  (**Data Enable**) : indique que des données sont en train de circuler sur le bus A/D (équivalent de ALE pour les données).



- $DT/\bar{R}$  (*Data Transmit / Receive*) : indique le sens de transfert des données.
- **Bus de données** de AD0 à AD15 et une partie du bus d'adresse.
- **Bus d'adresses** de AD0 à AD15 et de A16 à A19 .

## 4. Organisation interne du 8086

- Le 8086 est constitué de deux unités fonctionnant en parallèle (principe du pipeline) :
  - **L'unité d'interface de bus (BIU : Bus Interface Unit).**
    - Constituée d'un ensemble de registres d'adressage (IP, CS, SS, DS, ES),
    - d'un générateur d'adresse et
    - d'une file d'attente (registre d'instruction).
    - Son rôle est de chercher les instructions et les données stockées en mémoire.
  - **L'unité d'exécution (EU : Execution Unit).**
    - Destinée au traitement des données.
    - Se compose essentiellement de l'Unité Arithmétique et Logique,
    - l'unité de commande,
    - les registres opérationnels et d'adressage.
    - Son rôle est de lire les instructions stockées dans la file d'attente pour décodage et exécution.



## 4.1. Registres internes

### 4.1.1. Registres généraux

4 registres sur 16 bits :

	15	8	7	0
AX	AH		AL	
BX	BH		BL	
CX	CH		CL	
DX	DH		DL	

➤ **Accumulateur AX (AH, AL) :**

- Registre préféré dans les instructions arithmétiques, logiques et de transfert de données.
- Doit être utilisé dans les opérations de multiplication et de division.
- Doit être utilisé dans les opérations d'E/S.

➤ **Registre de base BX (BH, BL) :**

- Sert aussi comme registre d'adresse.

➤ **Compteur CX (BH, BL) :**

- Utilisé comme compteur de boucles.
- Utilisé dans les opérations de rotation et de décalage.

➤ **Registre de données DX (DH, DL) :**

- Utilisé dans les opérations de multiplication et de division.
- Utilisé aussi dans les opérations d'E/S.

### 4.1.2. Registres de pointeurs et d'index

**Pointeurs :**

- Registre SP : Stack Pointer, pointeur de pile (la pile est une zone de sauvegarde de données au cours d'exécution d'un programme).
- Registre BP : Base Pointer, pointeur de base, utilisé pour adresser des données sur la pile.

**Index :**

- Registre SI : Source Index.
- Registre DI : Destination Index.
- Ils sont utilisés pour les transferts de chaînes d'octets entre deux zones mémoire.

Les pointeurs et les index contiennent des adresses de cases mémoires.

**4.1.3. Pointeur d'instruction et flag**

2 registres sur 16 bits :

- Pointeur d'instruction : IP, contient l'adresse de la prochaine instruction à exécuter.
- Flags :

					<b>O</b>	<b>D</b>	<b>I</b>	<b>T</b>	<b>S</b>	<b>Z</b>		<b>A</b>		<b>P</b>	<b>C</b>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- |                                 |                         |                              |
|---------------------------------|-------------------------|------------------------------|
| ○ <b>CF</b> (Carry Flag)        | <b>CY</b> (carry=1)     | <b>NC</b> (not carry=0);     |
| ○ <b>PF</b> (Parity Flag)       | <b>PE</b> (even=1)      | <b>PO</b> (odd=0);           |
| ○ <b>AF</b> (Auxiliary Flag)    | <b>AC</b> (Auxiliary=1) | <b>NA</b> (Not Auxiliary=0); |
| ○ <b>ZF</b> (Zero Flag)         | <b>ZR</b> (Zero=1)      | <b>NZ</b> (Not Zero=0);      |
| ○ <b>SF</b> (Sign Flag)         | <b>NG</b> (Negative=1)  | <b>PL</b> (Positive=0);      |
| ○ <b>TF</b> (Trap Flag)         | Exécution pas à pas;    |                              |
| ○ <b>IF</b> (Interruption Flag) | <b>EI</b> (Enabled=1)   | <b>DI</b> (Disabled=0);      |
| ○ <b>DF</b> (Direction Flag)    | <b>DN</b> (Decrement=1) | <b>UP</b> (Increment=0);     |
| ○ <b>OF</b> (Overflow Flag)     | <b>OV</b> (Overflow=1)  | <b>NV</b> (No Overflow=0);   |

**4.1.4. Registres de segments**

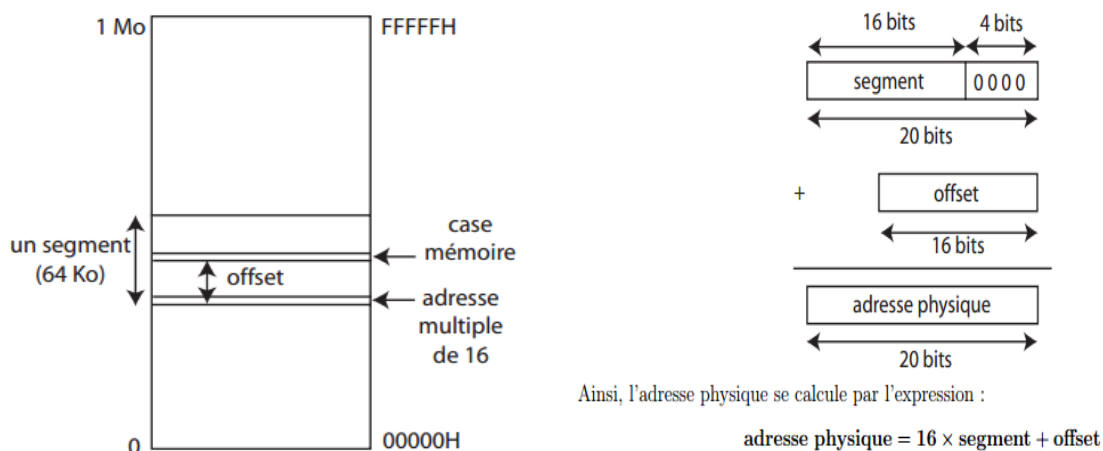
4 registres sur 16 bits :

- **CS** : Code Segment, registre de segment de code.
- **DS** : Data Segment, registre de segment de données.
- **SS** : Stack Segment, registre de segment de pile.
- **ES** : Extra Segment, registre de segment supplémentaire pour les données.

**Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.**

## 5. Segmentation de la mémoire

- Le microprocesseur 8086 possède un bus d'adresses de 20 bits, il peut adresser 1MO de mémoire.
- Or il possède des registres d'adresse de 16 bits (SP, SI, DI, IP, BP).
- Pour adresser la totalité de la mémoire, le 8086 utilise la technique de segmentation.
- Cette technique consiste à découper la mémoire en zones de 64KO appelés segments.
- Chaque segment est adressé par un registre segment et un registre d'adressage.
- Un registre segment contient l'adresse de la première case mémoire d'un segment.
- Un registre d'adressage contient le déplacement (**offset**) à l'intérieur de ce segment.
- La donnée d'un couple (segment, offset) définit une **adresse logique**, notée sous la forme **segment : offset**.
- L'adresse d'une case mémoire donnée sur 20 bits est appelée **adresse physique**.
- Correspondance entre **adresse physique** et **adresse logique** :



## 6. Microprocesseur 8088

- Le microprocesseur 8088 est identique au 8086 sauf que son bus de données externe est sur 8 bits au lieu de 16 bits, le bus de données interne restant est sur 16 bits.
- Différences avec le 8086 :
  - Les broches AD8 à AD15 deviennent A8 à A15 (bus de données sur 8 bits).
  - La broche  $\overline{BHE}$  n'existe pas dans 8088 car il n'y a pas d'octet de poids fort sur le bs de données.

- La broche  $M/\overline{IO}$  devient  $IO/\overline{M}$  pour la compatibilité avec les anciens circuits d'E/S.
- Au niveau de l'architecture interne, pas de différence avec le 8086 sauf que la file d'attente des instructions passe de 6 à 4 octets.

## Chapitre 6 : Assembleur du 8086

### 1. Introduction

- Chaque microprocesseur reconnaît un ensemble d'instructions appelé **jeu d'instructions** fixé par le constructeur.
- Le nombre d'instructions varie selon la nature du microprocesseur :
- Entre 75 et 100 instructions pour les microprocesseurs CISC.
- Entre 30 et 40 instructions pour les microprocesseurs RISC.
- Une instruction est définie par son code opératoire, valeur numérique binaire :
- On utilise donc une notation symbolique pour représenter les instructions : les mnémoniques.
- Un programme constitué de mnémoniques est appelé programme en assembleur.

### 2. Format d'une instruction

- Format d'une instruction :

[Etiquette]	Mnémonique	[opérandes]	[ ; commentaires]
1 <sup>er</sup> champ	2 <sup>ème</sup> champ	3 <sup>ème</sup> champ	4 <sup>ème</sup> champ

- Mnémonique : représente l'abréviation du nom du code opération.

Cop : Addition	➔	Mnémonique : ADD
Cop : Soustraction	➔	Mnémonique : SUB

- Une instruction contient au plus 4 champs :
  - Deux champs sont obligatoires (le mnémonique et les opérandes).
  - Les autres sont facultatifs.
- Les champs sont séparés entre eux par un espace blanc.
- Champ étiquette :
  - est obligatoire dans le cas où l'instruction va être repérée par d'autres instructions (saut, boucle, ...).

- peut avoir 31 caractères au maximum entre lettres et chiffres.
  - On ne peut utiliser un nom d'étiquette qui peut spécifier un mnémonique, ou un nom de registre, ce champ se termine toujours par ":".
- Champ mnémonique :
- représente le code opération qui exprime l'opération à effectuer par le microprocesseur.
- Champ opérandes :
- représente les données qui seront traitées par le code opération.
- Champ commentaires :
- est utilisé pour ajouter des explications supplémentaires sur l'instruction et commence toujours par ";".

### 3. Mode d'adressage

- En général, les instructions du 8086 possèdent les formes suivantes :

1. **[Etiq:] Cop [;Comm]**

- L'opération s'effectue à l'intérieur du microprocesseur sans faire appel aux opérandes.
- Par exemple, arrêt d'exécution d'un programme, synchronisation du microprocesseur avec un événement extérieur, ...

2. **[Etiq:] Cop opérande [;Comm]**

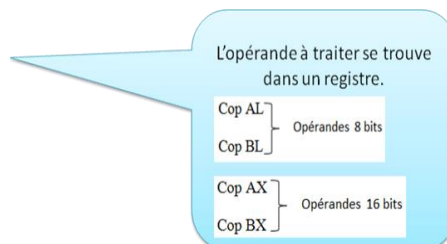
- Ce type d'instruction opère sur un seul opérande.

3. **[Etiq:] Cop opérande1, opérande2 [;Comm]**

- L'opération est effectuée sur deux opérandes.

- Dans les formes 2- et 3-, les opérandes peuvent se trouver soit dans les registres internes, soit dans la mémoire, où l'interface d'E/S, soit dans l'instruction elle-même.

**a) Cop registre**



**b) Cop opérande mémoire**

L'opérande est stocké dans un segment de données.

a) et b) sont deux écritures de la forme 2

**c) Cop registre1, registre2**

- Opérande1 et opérande2 se trouvent dans des registres.  
- Registre1 et registre2 sont de même taille.

Cop AL, BL

Cop DX, CX

**d) Cop registre, opérande mémoire**

- Registre et opérande mémoire de même taille.

Cop AL, opérande mémoire de 8 bits

Cop AX, opérande mémoire de 16 bits

**e) Cop opérande mémoire, registre**

- Registre et opérande mémoire de même taille.

Cop opérande mémoire de 8 bits, CL

Cop opérande mémoire de 16 bits, DX

**f) Cop registre, opérande immédiat**

- Un opérande est un registre, l'autre opérande est une constante donnée dans l'instruction.

- Elle peut être exprimée dans la base 2, 10 ou 16.

- Registre et opérande immédiat de même taille

Cop BX, 300

**g) Cop opérande mémoire, opérande immédiat**

- Les cas c) et g) représentent des écritures de la forme 3).  
- Les cas a) et c) utilisent le mode d'adressage registre.  
- Le mode adressage immédiat est utilisé dans le cas f).  
- Dans les cas d), e) et g), on peut utiliser plusieurs modes d'adressage pour accéder à un opérande stocké dans la mémoire.

### 3.1. Adressage direct

- L'opérande auquel on veut accéder est spécifié par un identificateur.
- Soient les déclarations suivantes :



*N DB 4*  
*M DW EF00h*  
*Tab DB 1,3,5,6,7*  
*Alpha DD 014FBC0Eh*

➤ **Cop opérande mémoire**

- Cop N
- Cop Tab+2

➤ **Cop registre, opérande mémoire**

- Cop AL, N
- Cop BL, Tab+4
- Cop BX, word PTR Alpha

➤ **Cop opérande mémoire, registre**

- Cop M, BX

## 3.2. Adressage indirect

1. Adressage basé sur le registre de base BX.
2. Adressage relatif avec le registre de base BX.
3. Adressage indexé.
4. Adressage indexé avec registre de base.

### 3.2.1. Adressage basé sur BX

- Le registre BX contient l'adresse de déplacement dans le segment de données où se trouve l'opérande.

**Exemple**

Cop opérande mémoire, registre

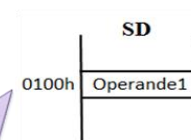
**Adresse de déplacement vers registre BX**

Cop [BX], registre ; informe l'assembleur qu'il s'agit d'une adresse et non d'un opérande

**Si BX=0100h**

Cop [BX], AL

**Le Cop va traiter l'opérande2 qui est stocké dans AL et l'opérande1 situé dans l'adresse 0100h**



- Le registre BX contient l'adresse de déplacement dans le segment de données où se trouve l'opérande.

**Exemple**

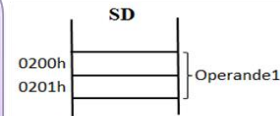
Cop opérande mémoire, registre

**Adresse de déplacement vers registre BX**

Cop [BX], registre ; informe l'assembleur qu'il s'agit d'une adresse et non d'un opérande

**Si BX=0200h**

Cop [BX], AX

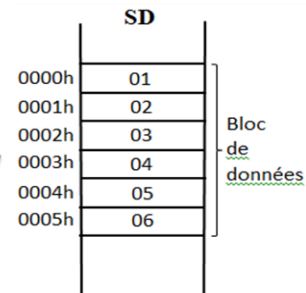
**Le Cop va traiter l'opérande2 qui est stocké dans AX et l'opérande1 stocké dans les adresses 0200h et 0201h**

### 3.2.2. Adressage relatif avec BX

- L'adresse de l'opérande mémoire est déterminée par la somme des contenus du registre BX et un déplacement dans le bloc de données où se trouve l'opérande.

**Exemple**

Cop registre, opérande mémoire

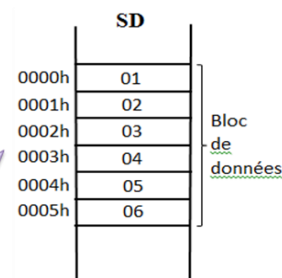
**Bloc DB 1, 2, 3, 4, 5, 6****MOV BX, OFFSET Bloc****; (BX) = 0000h****Cop AL, [BX+3]**

### 3.2.3. Adressage indexé

- L'adresse de l'opérande est obtenue par la somme du déplacement associé au nom d'une table de données et le contenu d'un registre d'index SI ou DI.

**Exemple**

Cop registre, opérande mémoire

**Bloc DB 1, 2, 3, 4, 5, 6****Cop AL, Bloc[DI]****Si DI=2****Adresse de l'opérande2=0000h+2**

### 3.2.4. Adressage indexé avec BX

- L'adresse de l'opérande mémoire est obtenue en additionnant le contenu du registre BX et le contenu d'un registre d'index DI ou SI.

#### Exemple

**Cop CL, [BX+DI]**

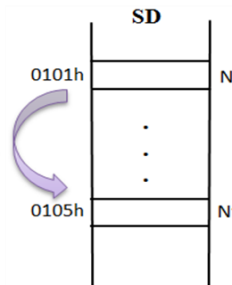
Si BX=0100h et DI=0001h

(BX)+(DI)=0101h

**MOV DI, 4**

**MOV BX, OFFSET N**

**Cop AL, [BX+DI]**



## 4. Déclaration de variables

- Les variables se déclarent de la manière suivante :
  - *v1 db ? ;v1 est un byte non initialisé*
  - *v2 db 0FFh ;v2 est un byte initialisé à FF*
  - *v3 dw ? ;v3 est un mot (16 bits)*
  - *t4 db 5 dup(?) ;t4 est un tableau de 5 bytes non initialisés*
  - *t5 dw 10 dup(15) ;t5 est un tableau de 10 byte initialisés à 15*
- De manière générale :
  - *db : 1 byte (8 bits) (Declare Byte)*
  - *dw : 1 word (16 bits) (Declare Word)*
  - *dd : 2 words (32 bits)(Declare Double)*
- Les constantes peuvent être écrites en :
  - **décimal** : 1, 2, 3, 123, 45
  - **hexadécimal** : 1h, 2h, 3h, 12h, 0Fh, 0AD4h (noter la présence du 0 quand le premier chiffre du nombre en hexadécimal commence par une lettre)
  - **binaire** : 1b, 0b, 1010b, 111101b

## 5. Instructions de transfert

### 5.1. Transfert de données

- **Syntaxe :**
  - *MOV destination, source*

- Copie de la source dans la destination. Destination et source doivent être de même taille.

➤ **Différentes formes :**

1. MOV registre1, registre2
2. MOV registre, opérande mémoire
3. MOV opérande mémoire, registre
4. MOV registre, opérande immédiat
5. MOV opérande mémoire, opérande immédiat

➤ **Exemples :**

- *MOV AX, 300* (4)
- *MOV BL, AL* (1)
- *MOV CL, N* (2)

➤ **Syntaxe**

- ***XCHG destination, source***
- Echange de contenu entre la source et la destination. Destination et source doivent être de même taille.

➤ **Différentes formes :**

1. *XCHG registre1, registre2*
2. *XCHG registre, opérande mémoire*
3. *XCHG opérande mémoire, registre*

➤ **Exemples :**

- *MOV BL, 20*
- *MOV CL, 10*
- *XCHG BL, CL*

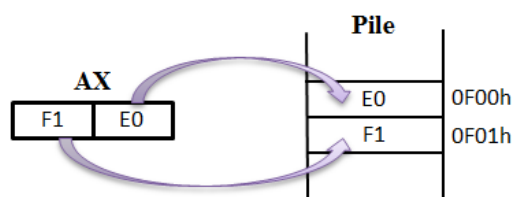
## 5.2. Transfert entre le microprocesseur et la pile

➤ **Syntaxe :**

- ***PUSH source***
- Empilement de la source (information de 16 bits) dans la pile.
- $(SP) = (SP) - 2$ .

➤ **Exemple :**

- *PUSH AX*

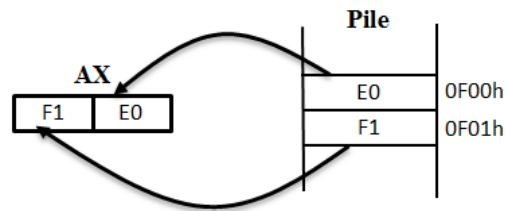


➤ **Syntaxe :**

- **POP destination**
- Dépilement du sommet de la pile dans la destination (information de 16 bits).
- $(SP) = (SP) + 2$ .

➤ **Exemple :**

- **POP AX**



➤ **Syntaxe :**

- **PUSHF**
- Sauvegarde des flags sur la pile.

➤ **Syntaxe :**

- **POPF**
- Restauration des flags à partir de la pile.

## 6. Instructions arithmétiques

### 6.1. Addition

➤ **Syntaxe :**

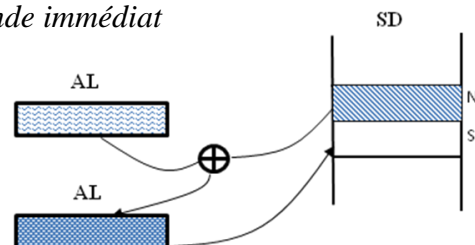
- **ADD destination, source**
- Addition de source et destination. Le résultat est mis dans destination. Source et destination doivent être de même taille.

➤ **Différentes formes :**

1. *ADD registre1, registre2*
2. *ADD registre, opérande mémoire*
3. *ADD opérande mémoire, registre*
4. *ADD registre, opérande immédiat*
5. *ADD opérande mémoire, opérande immédiat*

➤ **Exemples :**

- **MOV AL, M**
- **ADD AL, N**
- **MOV S, AL**



➤ **Syntaxe :**

- *ADC destination, source*
- Addition de source et destination avec retenue. Le résultat est mis dans destination. Source et destination doivent être de même taille.

➤ **Différentes formes :**

1. *ADC registre1, registre2*  
 $(\text{registre1}) + (\text{registre2}) + (CF) \text{ ----} \rightarrow (\text{registre1})$
2. *ADC registre, opérande mémoire*
3. *ADC opérande mémoire, registre*
4. *ADC registre, opérande immédiat*
5. *ADC opérande mémoire, opérande immédiat*

➤ **Remarque :**

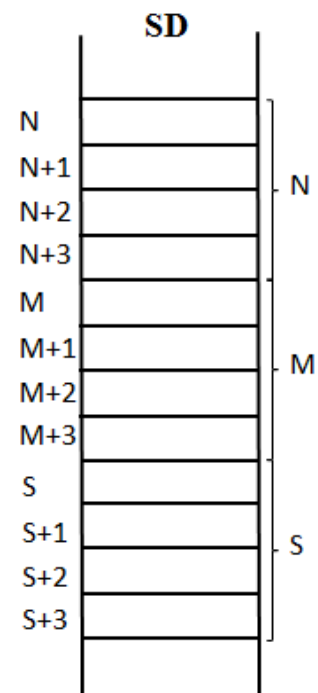
- Les instructions ADD et ADC peuvent modifier certains indicateurs d'état et en particulier les bits CF, ZF, SF, PF, AF, OF.

➤ **Exemple 1 :**

- *N DB 20*
- *MOV BL, 10*
- *MOV CL, N*
- *ADC CL, BL ; avec CF=1*
- $(CL) = 00010100 + 00001010 + 1 = 00011111$

➤ **Exemple 2 :**

- *N DD ?*
- *M DD ?*
- *S DD ?*
- 1. *MOV AX, WORD PTR N*  
*MOV BX, WORD PTR M*
- 2. *ADD AX, BX*
- 3. *MOV WORD PTR S, AX*
- 4. *MOV AX, WORD PTR N+2*
- 5. *ADC AX, WORD PTR M+2*
- 6. *MOV WORD PTR S+2, AX*  
*Au niveau 5 CF=1*  
 $(AX) + (\text{WORD PTR } M+2) + (CF) = (AX)$



## 6.2. Soustraction

➤ **Syntaxe :**

- *SUB destination, source*
- Soustraction de source et destination. Le résultat est mis dans destination. Source et destination doivent être de même taille.

➤ **Différentes formes :**

1. *SUB registre1, registre2*
2. *SUB registre, opérande mémoire*
3. *SUB opérande mémoire, registre*
4. *SUB registre, opérande immédiat*
5. *SUB opérande mémoire, opérande immédiat*

➤ **Exemples :**

- *MOV AL, 30*
- *ADD BL, 10*
- *SUB AL, BL*

➤ **Syntaxe :**

- *SBB destination, source*
- Soustraction de source et destination avec retenue. Le résultat est mis dans destination. Source et destination doivent être de même taille.
- Mêmes formes que celles de SUB.

➤ **Remarque :**

- Les instructions SUB et SBB peuvent modifier certains indicateurs d'état et en particulier les bits CF, ZF, SF, PF, AF, OF.

## 6.3. Multiplication

➤ **Syntaxe :**

- *MUL source*
- Multiplication non signée d'octets ou de mots. Elle est valable pour les opérandes en binaires pur.

➤ **Différentes formes :**

1. *MUL registre*
2. *MUL opérande mémoire*

➤ **Syntaxe :**

- ***IMUL source***
- Multiplication signée d'octets ou de mots. Elle est valable pour les opérandes en complément à deux.

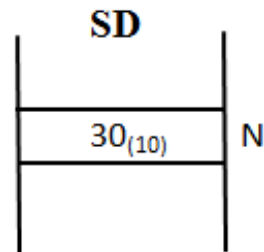
➤ **Différentes formes**

- *IMUL registre*
- *IMUL opérande mémoire*

### 6.3.1. Multiplication pour les opérandes de 8 bits

➤ **But :**

- Le multiplicateur doit être dans le registre AL.
- Le multiplicande est dans la source.
- Le résultat est dans le registre AX.



➤ **Exemple :**

- *MOV AL, 5*
- *MUL N*
- $(AX) = 150_{(10)}$

### 6.3.2. Multiplication pour les opérandes de 16 bits

➤ **But :**

- Le multiplicateur doit être dans le registre AX.
- Le multiplicande est dans la source.
- Le résultat est dans le registre AX et DX.
- AX contient les deux octets les moins significatifs et DX contient les octets les plus significatifs.

## 6.4. Division

➤ **Syntaxe :**

- ***DIV source***
- Division non signée d'octets ou de mots. Elle est valable pour les opérandes en binaires pur.

➤ **Différentes formes**

- *DIV registre*



- *DIV opérande mémoire*

➤ **Syntaxe :**

- *IDIV source*
- Division signée d'octets ou de mots. Elle est valable pour les opérandes en complément à deux.

➤ **Différentes formes :**

- *IDIV registre*
- *IDIV opérande mémoire*

### 6.4.1. Division avec une source de 8 bits

➤ **But :**

- Le dividende doit être dans le registre AX.
- Le diviseur est dans la source.
- Le résultat quotient est dans le registre AL et le reste de la division est dans le registre AH.

➤ **Exemple :**

- *MOV AX, 600*
- *MOV BL, 10*
- *DIV BL*
- *(AL) = 60      (AH) = 0*

### 6.4.2. Division avec une source de 16 bits

➤ **But :**

- Le dividende doit être dans les registres AX et DX.
- Le diviseur est dans la source (opérande de 16 bits).
- Le résultat quotient est dans le registre AX et le reste de la division est dans le registre DX.

## 6.5. Incrémentation

➤ **Syntaxe :**

- *INC registre*                      *(de 8 bits ou 16 bits)*
- *INC opérande mémoire* *(de 8 bits ou 16 bits)*

➤ **But :**

- $(registre) = (registre) + 1$
- $(op\acute{e}r\grave{a}nde\ m\acute{e}moire) = (op\acute{e}r\grave{a}nde\ m\acute{e}moire) + 1$

➤ **Exemples**

- *INC DI*      ➔       $(DI) = (DI) + 1$
- *INC AX*      ➔       $(AX) = (AX) + 1$
- *INC M*      ➔       $(M) = (M) + 1$

## 6.6. Décrémentation

➤ **Syntaxe :**

- *DEC registre*      (*de 8 bits ou 16 bits*)
- *DEC op\acute{e}r\grave{a}nde m\acute{e}moire* (*de 8 bits ou 16 bits*)

➤ **But :**

- $(registre) = (registre) - 1$
- $(op\acute{e}r\grave{a}nde\ m\acute{e}moire) = (op\acute{e}r\grave{a}nde\ m\acute{e}moire) - 1$

➤ **Exemples :**

- *DEC BX*      ➔       $(BX) = (BX) - 1$
- *DEC CL*      ➔       $(CL) = (CL) - 1$
- *DEC SI*      ➔       $(SI) = (SI) - 1$

## 6.7. Comparaison

➤ **Syntaxe :**

- *CMP destination, source*
- Comparaison entre la source et la destination
- Destination – source ➔ modification des bits du registre d'état ZF, SF, CF, AF, PF, OF.

➤ **Différentes formes :**

- *CMP registre1, registre2*
- *CMP registre, op\acute{e}r\grave{a}nde m\acute{e}moire*
- *CMP op\acute{e}r\grave{a}nde m\acute{e}moire, registre*
- *CMP registre, op\acute{e}r\grave{a}nde imm\acute{e}diat*
- *CMP op\acute{e}r\grave{a}nde m\acute{e}moire, op\acute{e}r\grave{a}nde imm\acute{e}diat*

➤ **Exemples :**

- *CMP BL, AL*
  - $ZF = 1 \rightarrow (AL) = (BL)$
  - $ZF = 0 \rightarrow (AL) \neq (BL)$
  - $SF = 1 \rightarrow (BL) < (AL)$

## 7. Instructions logiques

### 7.1. Opérateurs logiques AND, OR, XOR

➤ **Syntaxe :**

- *OP\_LOGIQUE destination, source*
- Destination reçoit le résultat de l'opération logique. Source est appelée un masque, elle est utilisée pour positionner à 1 ou à 0 les bits de la destination.

➤ **Différentes formes :**

1. *OP\_LOGIQUE registre1, registre2*
2. *OP\_LOGIQUE registre, opérande mémoire*
3. *OP\_LOGIQUE opérande mémoire, registre*
4. *OP\_LOGIQUE registre, opérande immédiat*
5. *OP\_LOGIQUE opérande mémoire, opérande immédiat*

➤ **Exemples :**

- *MOV BL, 10001000b*                      *MOV AL, 11000000b*
- *AND BL, 10000000b*                      *OR AL, 00001111b*
- ; positionnement à 0 du 4ème bit.                      ; positionnement à 1 des 4 premiers bits.

### 7.2. Opérateur de complément à 1

➤ **Syntaxe :**

- *NOT registre*
- *NOT opérande mémoire*
- Complément à 1 d'un octet ou d'un mot
- $NOT\ registre \Rightarrow \overline{registre}$
- $NOT\ op\ mémoire \Rightarrow \overline{op\ mémoire}$

➤ **Exemple :**

- *MOV AL, 0Fh*                      ; *AL=00001111*
- *NOT AL*                              ; *AL=11110000*

## 7.3. Opérateur de complément à 2

➤ **Syntaxe :**

- *NEG registre*
- *NEG opérande mémoire*
- Complément à 2 d'un octet ou d'un mot
- $NEG\ registre \Rightarrow \overline{registre} + 1$
- $NEG\ op\ mémoire \Rightarrow \overline{op\ mémoire} + 1$

➤ **Exemple :**

- *MOV AL, 0Fh* ; *AL=00001111*
- *NEG AL* ; *AL=11110001*

➤ **Remarque :**

- Les bits qui peuvent être modifiés par les opérateurs logiques sont ZF, SF, PF.

## 7.4. Opérateurs de décalage et rotation

➤ **Syntaxe :**

- *OPERATEUR destination, compte*
- Destination : un registre ou un opérande mémoire de 8 bits ou de 16 bits.
- compte : nombre de décalages ou de rotations :
  - Si compte=1 => un seul décalage ou une seule rotation.
  - => on utilise l'adressage immédiat.
  - Si compte>1 => plusieurs décalages ou rotations
  - => compte sera chargé dans le registre CL.
  - Le nombre de décalage ou de rotations est compris entre 1 et 255.

### 7.4.1. Décalage arithmétique à gauche

➤ **Syntaxe :**

- *SAL destination, compte (opérande signé cà2)*



➤ **Exemple :**

- *MOV AL, 10101100b*
- *SAL AL, 1* ; *(AL)=01011000* *CF=1*
- *MOV CL, 2*

- *MOV BL, 11001011b*
- *SAL BL, CL* ;(BL)=00101100 CF=1

### 7.4.2. Décalage arithmétique à droite

➤ Syntaxe :

- *SAR destination, compte (opérande signé cà2)*



➤ Exemple :

- *MOV AL, 10111110b*
- *SAR AL, 1* ;(AL)=11011111 CF=0
- *MOV CL, 3*
- *MOV BL, 11001011b*
- *SAR BL, CL* ;(BL)=11111001 CF=0

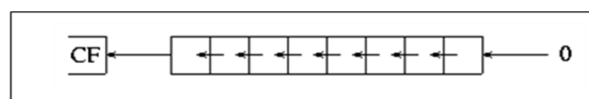
➤ Remarque :

- 1 décalage arithmétique à gauche  $\equiv$  1 multiplication par 2
- 1 décalage arithmétique à droite  $\equiv$  1 division par 2

### 7.4.3. Décalage logique à gauche

➤ Syntaxe :

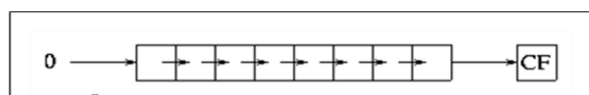
- *SHL destination, compte (binaire pur)*
- Equivalent à SAL



### 7.4.4. Décalage logique à droite

➤ Syntaxe :

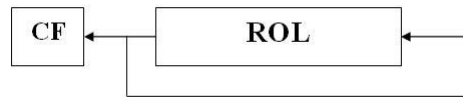
- *SHR destination, compte (binaire pur)*



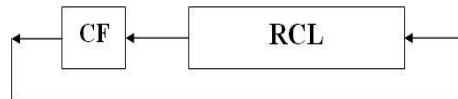
### 7.4.5. Rotation circulaire à gauche

➤ Syntaxe

- *ROL destination, compte (rotation sans CF)*



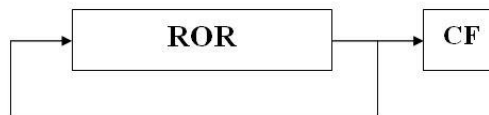
- *RCL destination, compte (rotation avec CF)*



#### 7.4.6. Rotation circulaire à droite

##### ➤ Syntaxe

- *ROR destination, compte (rotation sans CF)*



- *RCR destination, compte (rotation avec CF)*



## 8. Instructions de branchement

- Les instructions de branchement (ou saut) permettent de modifier l'ordre d'exécution des instructions en fonction de certaines conditions.
- Il existe 3 types de sauts :
  - Saut inconditionnel
  - Saut conditionnel
  - Appel de sous-programme

### 8.1. Saut inconditionnel

##### ➤ Syntaxe :

- *JMP etiquette*

- cette instruction effectue un saut à l'étiquette où se trouve l'instruction désirée.

```
        ⋮ } ← instructions précédant le saut
      jmp suite
        ⋮ } ← instructions suivant le saut (jamais exécutées)
suite : ... ← instruction exécutée après le saut
```

➤ **Exemple :**

- *boucle : inc ax*
- *dec bx ;boucle infinie*
- *jmp boucle*

➤ **Remarque**

- JMP ajoute à IP un déplacement (saut) codé en complément à 2.

## 8.2. Saut conditionnel

➤ **Syntaxe :**

- *Jcondition etiquette*
- 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 des indicateurs d'état (flags).

instruction	nom	condition
JZ label	Jump if Zero	saut si $ZF = 1$
JNZ label	Jump if Not Zero	saut si $ZF = 0$
JE label	Jump if Equal	saut si $ZF = 1$
JNE label	Jump if Not Equal	saut si $ZF = 0$
JC label	Jump if Carry	saut si $CF = 1$
JNC label	Jump if Not Carry	saut si $CF = 0$
JS label	Jump if Sign	saut si $SF = 1$
JNS label	Jump if Not Sign	saut si $SF = 0$
JO label	Jump if Overflow	saut si $OF = 1$
JNO label	Jump if Not Overflow	saut si $OF = 0$
JP label	Jump if Parity	saut si $PF = 1$
JNP label	Jump if Not Parity	saut si $PF = 0$

➤ **Remarque :**

- les indicateurs sont positionnés en fonction du résultat de la dernière opération.

➤ **Exemple :**

```
        ⋮ } ← instructions précédant le saut conditionnel
      jnz suite
        ⋮ } ← instructions exécutées si la condition  $ZF = 0$  est vérifiée
suite : ... ← instruction exécutée à la suite du saut
```

➤ **Remarque :**

- il existe un autre type de saut conditionnel, les sauts arithmétiques. Ils suivent en général l'instruction CMP.

condition	nombres signés	nombres non signés
=	JEQ label	JEQ label
>	JG label	JA label
<	JL label	JB label
≠	JNE label	JNE label

➤ **Exemple :**

```

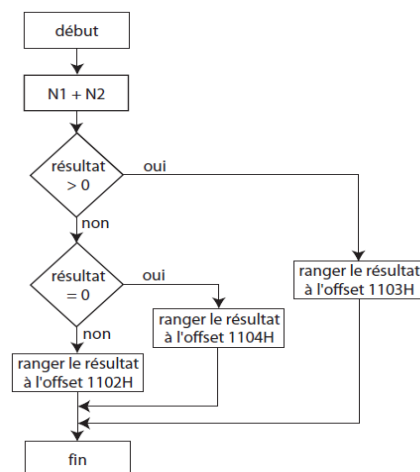
                cmp ax,bx
                jg superieur
                jl inferieur
superieur :    ...
                :
inferieur :    ...

```

- on veut additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul.

○

Organigramme :



Programme :

```

mov    al,[1100H]
add    al,[1101H]
js     negatif
jz     nul
mov    [1102H],al
jmp    fin
negatif : mov [1103H],al
        jmp fin
nul :    mov [1104H],al
fin :    hlt

```

## 8.3. Boucles

➤ **LOOP etiquette**

- $(cx)=(cx)-1$



- $(cx) \neq 0 \rightarrow$  branchement à l'étiquette
- $(cx) = 0 \rightarrow$  exécution de l'instruction suivante

➤ **LOOPZ étiquette ou LOOPE étiquette**

- $(cx) = (cx) - 1$ 
  - $(cx) \neq 0$  et  $(ZF) = 1 \rightarrow$  branchement à l'étiquette
  - $(cx) = 0$  ou  $(ZF) = 0 \rightarrow$  exécution de l'instruction suivante

➤ **LOOPNZ étiquette**

- $(cx) = (cx) - 1$ 
  - $(cx) \neq 0$  et  $(ZF) = 0 \rightarrow$  branchement à l'étiquette
  - $(cx) = 0$  ou  $(ZF) = 1 \rightarrow$  exécution de l'instruction suivante

## 8.4. Sous programme

- Un sous-programme (ou procédure) permet d'éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme.
- Le sous programme est défini une seule fois et peut être appelé plusieurs fois.

➤ **Ecriture d'un sous programme :**

```
nom_sp    PROC  
    .      ← instructions du sous-programme  
    ret    ← instruction de retour au programme principal  
nom_sp    ENDP
```

➤ **Remarque :**

- Une procédure peut être de type **NEAR** si elle se trouve dans le même segment ou de type **FAR** si elle se trouve dans un autre segment.

➤ **Exemple :**

```
ss_prog1    PROC    NEAR  
ss_prog2    PROC    FAR
```

## 8.5. Appel de sous programme

➤ **Appel de sous programme :**

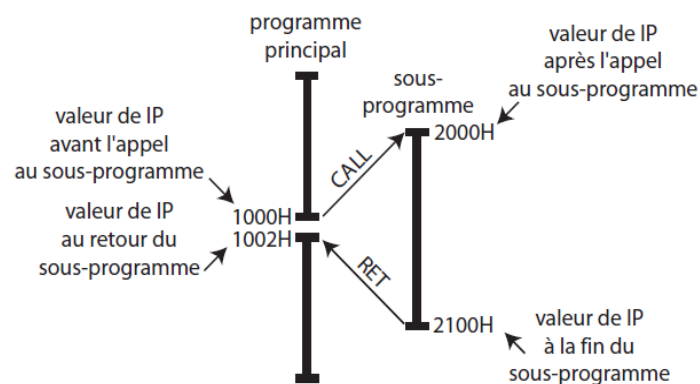
```
    : }      ← instructions précédant l'appel au sous-programme  
call nom_sp  ← appel au sous-programme  
    : }      ← instructions exécutées après le retour au programme principal
```

➤ **Lors de l'exécution de l'instruction call :**

- IP est chargé avec l'adresse de la première instruction du sous programme.

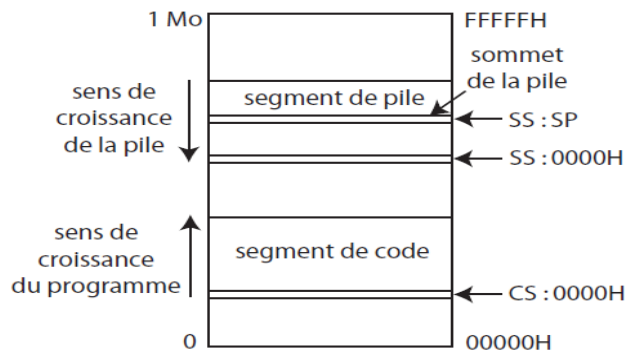
➤ **Lors du retour au programme principale :**

- L'instruction suivant le call doit être exécutée.
  - IP doit être rechargé avec l'adresse de cette instruction.
- Avant de charger IP avec l'adresse du sous programme, le contenu de IP (**retour**) est sauvegardé dans la pile.
- Lors de l'exécution de l'instruction **ret**, cette adresse est récupérée à partir de la pile et rechargée dans IP. Ainsi le programme appelant se poursuit.



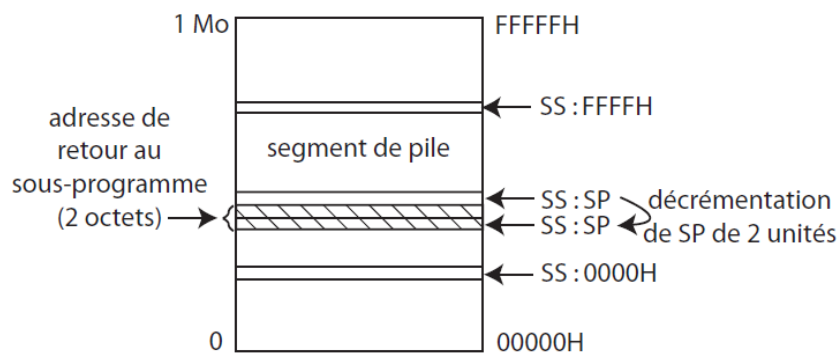
➤ **Fonctionnement de la pile :**

- La pile est une zone mémoire fonctionnant en mode **LIFO**.
- Deux opérations sont possibles dans la pile :
- **Empiler** une donnée : placer la donnée au sommet de la pile.
  - **Dépiler** une donnée : lire la donnée se trouvant au sommet de la pile.
- Le sommet de la pile est repéré par le registre **SP**.
- La pile est définie dans le segment de pile dont l'adresse de segment est contenue dans le registre **SS**.



➤ **Remarque :**

- La pile et le programme croissent en sens inverse pour diminuer le risque de collision entre le code et la pile sont dans le même segment (SS=CS).
- Lors de l'appel à un sous programme, l'adresse de retour au programme appelant (IP) est empilée et le registre SP est automatiquement décrémenté.
- Au retour du sous programme, le registre IP est rechargé automatiquement avec le sommet de la pile et SP est incrémenté.



➤ **Remarque :**

- La valeur de SP doit être initialisée avant l'utilisation de la pile.
- **Utilisation de la pile pour le passage de paramètres :**
  - Pour transmettre des paramètres à une procédure :
    - on les empile avant l'appel de la procédure,
    - puis celle-ci les récupère en effectuant un adressage utilisant le registre BP.

➤ **Exemple :**

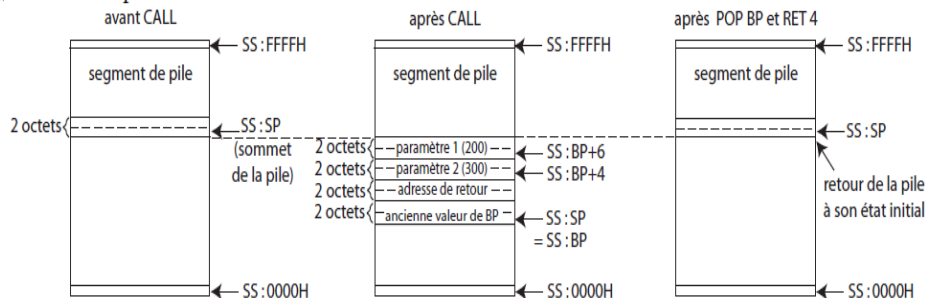
- soit une procédure effectuant la somme de deux nombres et retournant le résultat dans le registre AX :

```

• programme principal :
  mov ax,200
  push ax          ; empilage du premier paramètre
  mov ax,300
  push ax          ; empilage du deuxième paramètre
  call somme        ; appel de la procédure somme

• procédure somme :
  somme
  proc
    push bp        ; sauvegarde de BP
    mov bp,sp      ; faire pointer BP sur le sommet de la pile
    mov ax,[bp+4]  ; récupération du deuxième paramètre
    add ax,[bp+6]  ; addition au premier paramètre
    pop bp         ; restauration de l'ancienne valeur de BP
  
```

soit l'état de la pile :

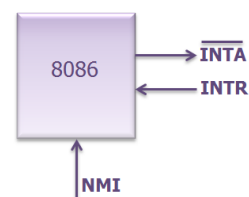


## 9. Interruptions

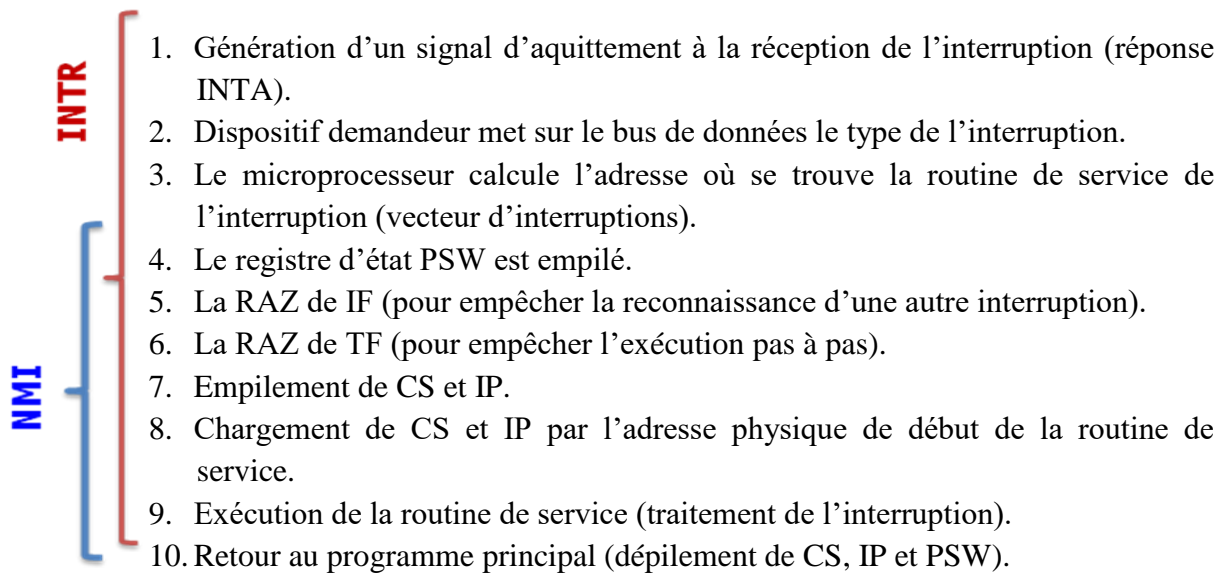
- Sur une machine mono-processeur, un seul programme (processus) est exécuté à la fois.
- Une autre composante peut demander de l'interrompre pour effectuer temporairement un autre traitement :
  - **Périphérique** : appui sur une touche clavier, faire bouger la souris, arrivée d'un paquet réseau, ...
  - **Gestion d'erreur** : division par zéro, accès à une zone mémoire protégée, ...
- Une interruption est un mécanisme qui permet d'interrompre l'exécution d'un processus suite à un événement extérieur ou intérieur et de passer le contrôle à une routine dite **traitement d'interruption**.
- On distingue deux sortes d'interruptions :
  - Interruptions matérielles
  - Interruptions logicielles

### 9.1. Interruptions matérielles

- Il existe deux interruptions matérielles :
  - **INTR, NMI** sont générées par un dispositif externe.
  - **INTA** est la réponse à la demande de l'interruption.



- Etapes effectuées lors d'une reconnaissance d'une interruption :



## 9.2. Interruptions logicielles

- **Syntaxe :**

- *INT type*
- Appel à une routine de service de l'interruption logicielle indiquée par le type.
- *type* : un nombre codé en hexadécimal sur un octet de 00 à FF.
- On a au total 256 interruptions.

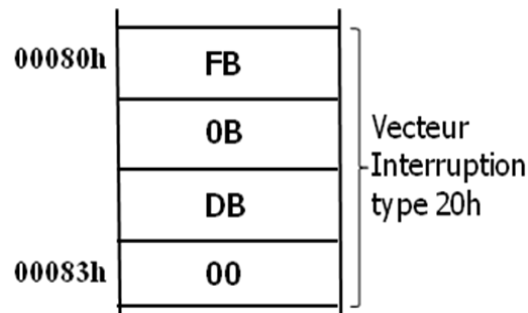
- **Remarques**

- Ce type d'interruptions est toujours accepté par le microprocesseur.
- Le processus de traitement de ces interruptions commence à partir de la troisième étape.
- Elles sont utilisées par le programmeur en cas de besoin.

## 9.3. Vecteurs d'interruptions

- Pour exécuter une routine de traitement d'une interruption, les registres CS et IP sont chargés par le contenu de quatre cases mémoires appelées **vecteur d'interruption**.
- Ce vecteur représente l'adresse physique sur 20 bits de la routine de traitement de l'interruption.
- Calcul de l'adresse de l'interruption de type FF:
- $AD = 4 \times FF = 003FCh$

- Le mot le moins significatif du vecteur est transféré vers IP, et le mot le plus significatif vers CS.
- Adresse du vecteur d'interruption de type 20h :
  - AD = 00080h
  - IP = 0BFBh
  - CS = 00DBh
  - Adresse physique : 019Abh



## 9.4. Interruption du MS-DOS

- MS-DOS fournit un ensemble de fonctions comme :
  - Caractères de contrôle ASCII,
  - Fonctions d'affichage,
  - Fonctions de saisie,
  - Fonctions date/heure,
  - Etc.
- L'appel d'une fonction s'effectue en :
  - Transférant au registre ah le numéro de la fonction.
  - Introduisant les paramètres d'entrée.
  - Exécutant l'instruction : **INT 21h**.

### 9.4.1. Affichage à l'écran

- **Afficher un caractère :**

```
mov ah, 02h    ;fonction no. 2
mov dl, 'a'    ;caractère à transférer dans dl
int 21h        ;appel à MS-DOS
```

- **Afficher un message :**

```
.data
msg db 'Hello world',13,10,'$'
```

*.code*

```
.....  
mov ah, 09h      ;fonction no. 9  
mov dx, offset msg ;pointe vers la chaîne  
int 21h          ;appel à MS-DOS
```

#### 9.4.2. Saisie du calvier

➤ **Saisir un caractère avec echo :**

```
mov ah, 01h  
int 21h
```

➤ **Saisir un caractère sans echo :**

```
mov ah, 08h  
int 21h
```

➤ **Saisir une chaîne de caractères :**

```
mov ah, 10h  
lea dx, chaîne  
int 21h
```

#### 9.4.3. Arrêt de programme

➤ **Arrêt de programme et retour au DOS :**

- *mov ah, 4ch*
- *mov al, 00h*
- *int 21h*

➤ **Lire l'heure courante :**

- *mov ah, 2ch*
- *int 21h*

Au retour de l'appel, ch contient l'heure, cl les minutes et dh les secondes.

➤ **Lire la date courante :**

```
mov ah, 2ah  
int 21h
```

Au retour de l'appel, al contient le jour de la semaine codé (0:dimanche, 1:lundi, ...), cx l'année, dh le mois et dl le jour.





UNIVERSITE IBN TOFAIL  
Faculté des Sciences  
Département de Mathématique  
Et d'Informatique



# Architecture des Ordinateurs

---

- Support TP -

Hatim  
KHARRAZ AROUSSI

## ***TD N°1 – Mémoire et Microprocesseur –***

### **Exercice 1**

Soit un microprocesseur avec un bus d'adresse de 24 broches et bus de données de 16 broches.

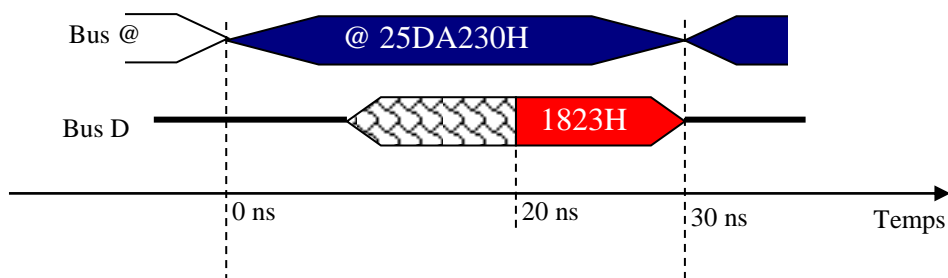
1. Quel est le nombre de cases mémoire adressable par ce microprocesseur ?
2. Calculer la capacité **C** de cette mémoire en bits ?
3. Soit le circuit mémoire suivant :



Définir en une phrase l'opération décrite dans ce circuit.

### **Exercice 2**

Soit une mémoire caractérisée par le chronogramme d'un cycle de lecture suivant :



1. Quelle est la capacité **C** de cette mémoire en bits.
2. Donner le débit de lecture **D**.
3. calculer le temps nécessaire **T<sub>lecture</sub>** pour lire à partir de cette mémoire un texte de taille **5Moct**.

### Exercice 3 :

Soit une image numérisée affichée sur un écran d'un ordinateur ayant un microprocesseur 16 bits utilisant la matrice d'affichage 800\*600 pixels, chaque pixel pouvant prendre 1024 couleurs différentes.

1. Quelle est la taille en bits de cette image ?
2. Combien de segments de données (DS) en mémoire on a besoin pour stocker cette image ? Rappel : **Taille<sub>1Seg</sub> = 64KOct.**

### Exercice 4

Supposons un son analogique de durée 2 minutes, numérisé par la technique d'échantillonnage avec une fréquence de **F<sub>ech</sub>=10<sup>3</sup>ech/sec**. Chaque échantillon est codé sur 2 octets. Le son est stocké dans la mémoire d'une machine avec un microprocesseur 16 bits.

1. Calculer la taille en bits de ce son **Taille<sub>son</sub>**.
2. Donner l'adresse physique **Adr@**, en hexadécimal, de la case mémoire contenant la dernière donnée de ce son, sachant que l'adresse de base du segment de données est **DS=1200H**

### Exercice 5

Sur un microprocesseur 16 bits, donner le résultat des opérations suivantes sur 1 octet et positionner les indicateurs du registre d'état : CF, CZ, CS et OF :

1. 25H + 5BH
2. 4BH + 67H
3. CEH + 32H

### Exercice 6

Pour un microprocesseur de fréquence 2,5Ghz (1Ghz = 10<sup>9</sup>hz). La phase de recherche et la phase de décodage et d'exécution ont besoin de 2 périodes d'horloge chacune. Calculez la durée d'un cycle machine.

### Exercice 7

Soit un microprocesseur d'architecture RISC ayant une fréquence de 20MHz (1MHz = 10<sup>6</sup>Hz), Chaque instruction nécessite au moyen 4 périodes d'horloge.




1. Calculer la valeur de cette période (T), et la puissance du traitement de ce processeur (MIPS).
2. On désire exécuter un programme de 10 instructions élémentaires sur ce processeur. Calculer le temps d'exécution de ce programme dans les cas suivants :
  - a. Le microprocesseur à une architecture scalaire simple.
  - b. Le microprocesseur à une architecture pipeliné à 4 phases.
  - c. Le microprocesseur à une architecture superscalaire à 2 unités de traitements, chaque unité est pipeliné de 4 phases.

## ***TP N°1 – Turbo Debugger –***

Un debugger est un outil qui permet d'exécuter instruction par instruction un programme tout en suivant en permanence l'évolution du contenu des registres, du segment de données, la pile, ... C'est un outil indispensable pour simplifier et accélérer la mise en point de programmes.

### **Objectifs**

Il est important de comprendre ce que l'on peut attendre d'un debugger et le principe général de son utilisation. Un debugger permet de répondre aux questions suivantes :

-  pourquoi mon programme ne fonctionne-t-il pas?
-  pourquoi mon programme boucle-t-il ?
-  où se plante mon programme?

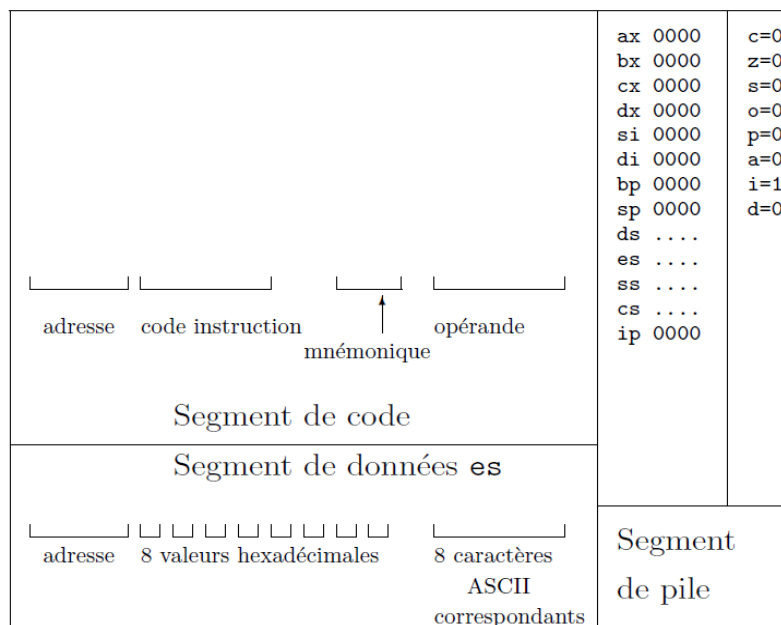
Pour cela, une fois dans le debugger, on peut placer des points d'arrêt dans le programme afin qu'il s'y arrête en indiquant les instructions où l'on veut que le programme s'arrête. A cet instant, on pourra consulter le contenu des registres, de la mémoire ou de la pile.

### **Lancement du debugger**

Après avoir obtenu un programme exécutable (par exemple prg.exe), on lance la commande :





**td prg**

L'écran s'efface et affiche la fenêtre ci-après. On quitte le debugger en tapant Alt-X.



## Fonctionnalités les plus utiles



### a- Menu Run

-  **Run** lance le programme. Si son exécution a déjà débutée, le programme poursuit son exécution là où il avait été arrêté. Sinon, il démarre à son début. Il ne s'arrête que lorsqu'il arrive à son terme ou sur le point d'arrêt si l'on en a placé.
-  **Trace into** arrêté sur une instruction call, considère l'exécution de l'ensemble du sous-programme comme un pas d'exécution ;
-  **Step over** exécute la prochaine instruction et s'arrête.
-  **Prg Reset** remise à zéro de l'exécution du programme. Remet le debugger dans son état initial, comme si l'on n'avait pas encore lancé l'exécution du programme.

### b- Points d'arrêt

Un point d'arrêt est un endroit dans le programme où on indique que l'on veut que celui-ci arrête son exécution.

Il existe deux manières de mettre en place des points d'arrêt :

-  soit implicitement en cliquant sur une instruction du programme et en lançant ensuite l'exécution du programme par un **Go to cursor**.
-  soit explicitement en cliquant sur une instruction du programme et en choisissant dans le menu **Breakpoints** l'option **At**, d'y placer un point d'arrêt. On peut placer plusieurs points d'arrêt dans le programme. A son exécution (par **Run** ou **Go to cursor**), le programme s'arrêtera automatiquement à chaque fois qu'il arrivera sur une instruction où on a placé un point d'arrêt.

## Opérations de base

### a. Afficher le segment de données

Au lancement de **td**, le debugger ne connaît pas l'emplacement en mémoire du segment de données, le registre **ds** étant initialisé par le programme généralement à l'exécution des premières instructions du programme. Une fois **ds** initialisé, on peut visualiser le contenu du registre de segment dans une fenêtre en choisissant dans le menu **View** l'option **Dump**.

### b. Afficher le contenu d'un segment de mémoire quelconque

On peut visualiser le contenu de la mémoire à partir de n'importe quelle adresse en spécifiant un numéro de segment et un offset. Pour cela, ouvrir un 'dump' mémoire en sélectionnant **Dump** dans le menu **View**. Ensuite, en cliquant dans la fenêtre avec le bouton droit de la souris, un menu apparaît. Sélectionner **Goto...** Le debugger demande l'adresse à partir de laquelle il doit afficher le contenu de la mémoire dans la fenêtre. On pourra alors taper **es:0**, **ds:0** ou **53DA:14**, selon les besoins.

**c. Modifier la valeur d'un registre, d'un indicateur ou d'une donnée en mémoire**

On peut facilement modifier la valeur d'un registre, d'un indicateur du registre d'état ou un octet en mémoire en cliquant dessus, en entrant sa nouvelle valeur et en validant.

**Travail à faire :**

- ✚ Editier le programme ci-dessous et sauvegarder le sous le nom **prg.asm** :

```
.model small
.data
    msg db 'Mon premier programme avec DEBUG',0dh,0ah,'$'
.stack
.code
main:
    mov ax,@data
    mov ds,ax

    mov ah,09h
    mov dx, offset msg
    int 21h

    mov ah,4ch
    int 21h
end main
```

- ✚ Compiler le programme avec la commande : **tasm /l /zi prg.asm**
- ✚ Faire l'édition de liens avec la commande : **tlink /v prg.obj**
- ✚ Lancer le debugage de td **prg.exe**
- ✚ Faites l'exécution pas à pas en consultant à chaque fois les registres.
- ✚ Faites l'exécution pas à pas et relever les valeurs des indicateurs pour le programme suivant :

```
mov ax, ffff
mov bx, ffff
add ax, bx
mov ax, 0001
dec ax
inc ax
sub ax, 0002
mov ah, 70
mov bh, 50
add ah, bh
```

## ***TP N°2 – Programmation en assembleur –***

### **Exercice 1 :**

Ecrire un programme assembleur qui permet d'afficher le message suivant : *"mon premier programme en assembleur"*.

### **Exercice 2 :**

Ecrire un programme assembleur qui affiche le message *"simulation de la boucle for"* 10 fois.

### **Exercice 3 :**

Ecrire un programme assembleur qui permet d'afficher le deuxième élément d'un tableau de caractères.

### **Exercice 4 :**

Ecrire un programme assembleur qui permet d'afficher les éléments, l'un après l'autre, d'un tableau de caractères. Afficher un élément par ligne.

### **Exercice 5 :**

Ecrire un programme assembleur qui permet de lire et d'afficher des caractères tant que le caractère lu est différent de "\$".

### **Exercice 6 :**

Ecrire un programme assembleur qui permet d'échanger et d'afficher le contenu des registres al et bl contenant des caractères.

### **Exercice 7 :**

Ecrire un programme assembleur qui permet de lire une chaîne de caractères en minuscule, la convertir en une chaîne de caractère en majuscule et de l'afficher.

### **Exercice 8 :**

Ecrire un programme assembleur qui permet de saisir deux entiers nommé a et b, d'incrémenter a, de décrémenter b, de calculer leur somme et leur produit, de calculer le reste et le quotient de la division de a par b.

### **Exercice 9 :**

Ecrire un programme assembleur qui permet de saisir deux entiers et de les afficher sur écran.

## ***TP N°3 – Procédures en assembleur –***

### **Exercice 1 :**

Ecrire un programme assembleur contenant trois procédures nommées **prints**, **scans**, **toupper**.

- **Procédure scans**
  - Saisie d'une chaîne de caractères.
- **Procédure prints**
  - Affichage d'une chaîne de caractères.
- **Procédure toupper**
  - Conversion d'une chaîne de caractères en minuscules en une chaîne en majuscules.
- **Programme principal**
  - Saisir quatre chaînes de caractères en minuscules.
  - Convertir ces chaînes en majuscules.
  - Afficher les quatre chaînes.

### **Exercice 2 :**

Ecrire un programme assembleur contenant deux procédures nommées respectivement **printi** et **scani**.

- **Procédure scani**
  - Saisie d'un entier.
- **Procédure printi**
  - Affichage d'un entier
- **Programme principal**
  - Déclarer un tableau d'entiers dont la taille maximale est 100
  - Saisir le nombre des éléments à remplir dans le tableau
  - Saisir les éléments du tableau
  - Afficher les éléments du tableau

### **Exercice 3 :**

Ecrire un programme assembleur contenant au moins deux procédures nommées respectivement **max**, **som**.

- **Procédure max**



Recherche et affiche le plus grand élément d'un tableau

– **Procédure som**

Calcul et affiche la somme des éléments d'un tableau

– **Programme principal**

- Déclare un tableau d'entiers sur 2 octets
- Transmet aux deux procédures
  - La longueur du tableau
  - L'adresse de son premier élément