

Architecture des ordinateurs

Mustapha JOHRI

ESTBM

2024/2025

Assembleur 8086

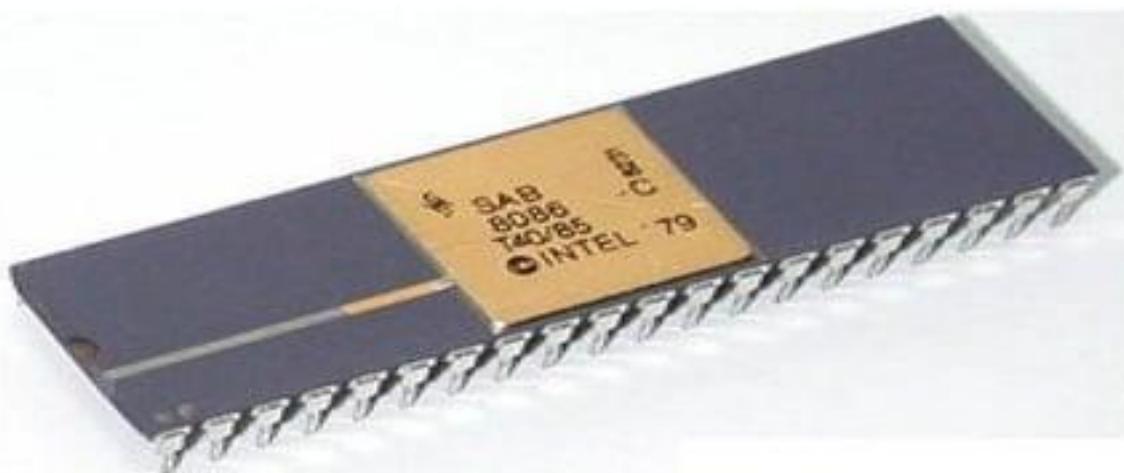
Langage machine

- Un langage machine est un langage compris directement par le processeur en vue d'une exécution. C'est un langage binaire : séquences de 0 et de 1.
- Dans la plupart des langages machine, une instruction commence par un opcode (une suite de bits qui porte la nature de l'instruction). Celui-ci est suivi des suites de bits codant les opérandes de l'instruction.

INTRODUCTION

PROCESSEUR INTEL 8086

- ❖ Disponible depuis 1978, le processeur 8086 fut le premier processeur 16 bits fabriqué par Intel.
- ❖ Il se présente sous forme d'un boîtier de 40 broches alimenté par une alimentation unique de 5V.



Langage machine

- Exemple
 - ▷ La suite 10101011 00000000 00000010 (AB 00 02 en hexa) est une instruction dont :
 - ▶ le **opcode** est 10101011
 - ▶ et l'**opérande** est 00000000 00000010.
- Ce langage est très difficile à maîtriser puisque chaque instruction est codée par une séquence propre de bits.
- Afin de faciliter la tâche du programmeur, un autre langage sera utilisé : C'est l'Assembleur.

Langage Assembleur

- Historiquement, le langage assembleur a été le premier langage de programmation non binaire accessible au programmeur. C'est un langage qui permet de décrire les instructions élémentaires qui manipulent dans le processeur les registres, l'accès à la mémoire.
- Le langage Assembleur est la représentation symbolique du langage machine. Autrement dit, le code en langage assembleur est la version lisible du code machine.
- Le langage assembleur est donc un langage de très bas niveau. Cela signifie qu'il est plus proche du langage machine.

Langage Assembleur

- Le langage assembleur permet au programmeur d'utiliser des codes symboliques (MOV, ADD, DIV, etc.) au lieu des codes numériques des opérations.
- Exemple :**

Additionner 2 et 3 produit le code suivant en langage Assembleur :

```
MOV AX,2  
ADD AX,3
```

Les variables en Assembleur

- Quelques descripteurs de taille utilisés :
 - ▷ **DB** (Define Byte) : 1 octet pour 8086
 - ▷ **DW** (Define Word) : 2 octets pour 8086
 - ▷ **DD** (Define Double) : 4 octets
- La valeur associée à la variable peut être un nombre (hexadécimal, décimal ou binaire) ou le symbole ? (si la variable n'est pas initialisée).
- Exemples :
 - ▶ **var1 DB 56d**
 - ▶ **var2 DB ?**
 - ▶ **var3 DW 1210h**

Les constantes

- Une constante numérique est définie par la directive EQU qui indique à l'assembleur que le nom de cette constante doit être remplacée par sa valeur au moment de la création de l'exécutable.
- Exemples

```
var1 EQU 50h  
MOV AX, var1
```

- **EQU** construit une constante dont la valeur est immédiate (et normalement invariante).

Les tableaux

- Un tableau peut être vu comme une liste de variables qui sont associées à un même nom.
- Exemple de création d'un tableau :

tab DB 01h , 02h , 03h

- Il est alors possible d'accéder à une case précise du tableau en utilisant un opérateur d'indexation : nous écrivons le nom du tableau suivi du numéro de la case placé entre crochets (les cases étant numérotées de 0 à N-1 pour un tableau comportant N éléments) :

MOV AX, tab[1]

- Si un tableau est construit en répétant une ou plusieurs valeurs, il peut être utile d'utiliser l'opérateur DUP :

tab DB 5 DUP (1,2)

L'opérateur DUP nécessite 2 informations : le nombre de répétition (un entier placé après DB) et la liste des valeurs à répéter, que nous plaçons entre parenthèses.

Les chaînes de caractères

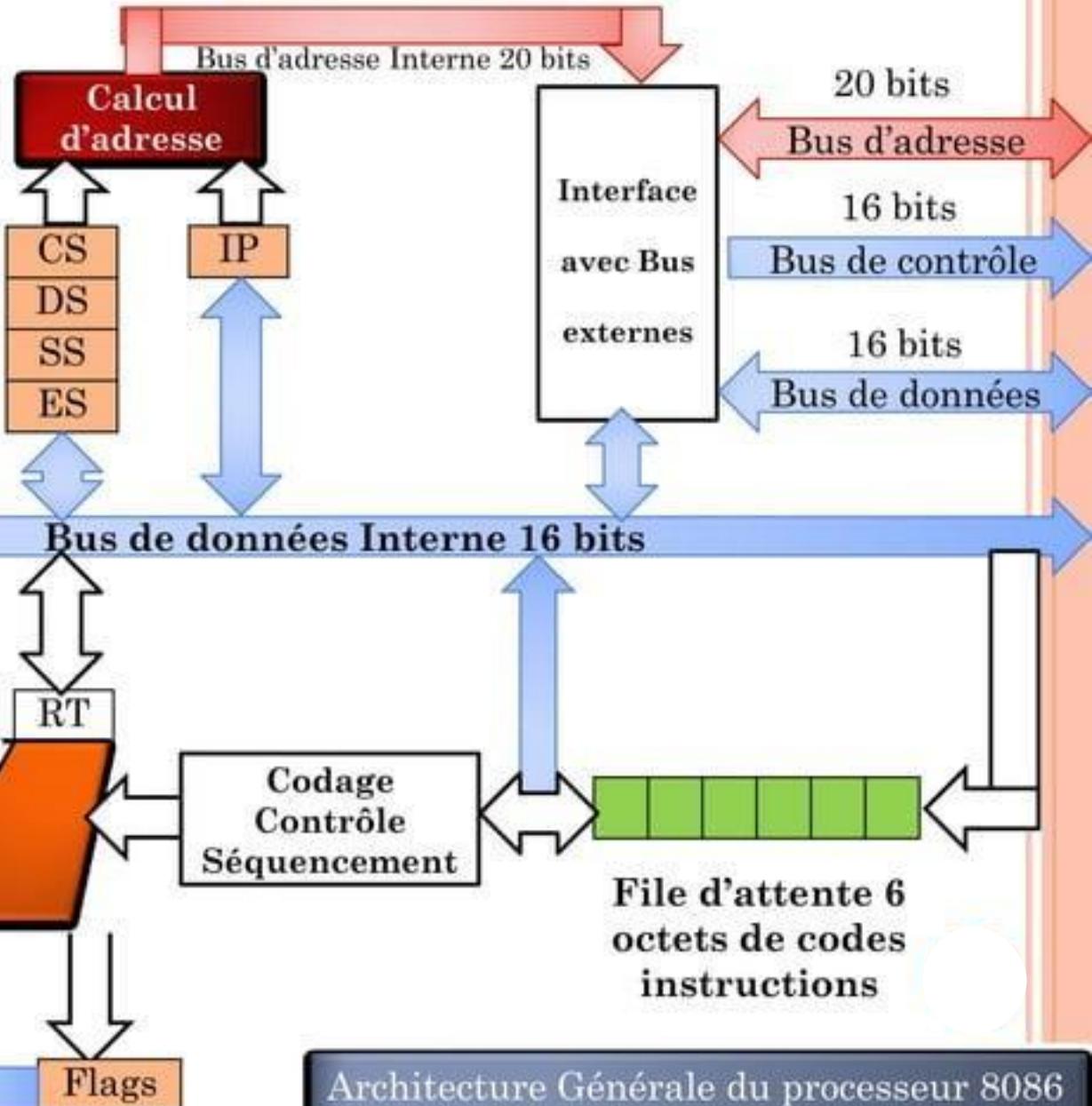
- Une chaîne de caractères peut être vue comme un tableau contenant une suite de codes ASCII séparés par des virgules, correspondant chacun à une lettre.
- Dans tous les cas, et pour faciliter les parcours et les recherches, les chaînes de caractères seront terminées par le symbole \$ (pour marquer la fin de la chaîne de caractères).
- **Exemples**
 - ▷ `var5 DB 'H','E','L','L','O', '$'`
 - ▷ `var6 DB 'HELLO$'`

Modes d'adressage

Modes d'adressage

- Les opérandes des instructions du microprocesseur 8086 peuvent être des registres, des constantes ou le contenu de cases mémoire
 ⇒ Modes d'adressage.
- Un mode d'adressage est un mécanisme utilisé par un processeur pour accéder à des données et des instructions en mémoire. Les modes d'adressage définissent les méthodes utilisées pour spécifier les adresses des opérandes dans les instructions du processeur.
- Les modes d'adressage les plus utilisés sont : mode immédiat, mode registre, mode direct, mode indirect (basé, indexé, basé-indexé)

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	BP	
	SP	
	SI	
	DI	



ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

SEGMENTATION DE LA MEMOIRE

- ❖ La taille du bus d'adresse égale à **20 bits** → La mémoire totale adressable égale **2^{20} octets = 1 Mo**
- ❖ La taille des registres est **16 bits** → on peut adresser seulement **2^{16} octets = 64 ko.**
- ❖ La mémoire est donc fractionnée en **pages** de 64 ko appelés **segments**.
- ❖ On utilise alors deux registres pour adresser une case mémoire donnée:
 - Un registre pour adresser le segment, appelé **registre segment: CS, DS, SS, ES**
 - Un registre pour adresser à l'intérieur du segment, appelé **registre offset: IP, SP, BP, SI, DI**.
- ❖ Une adresse se présente sous la forme **segment:offset**

ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

REGISTRES GÉNÉRAUX

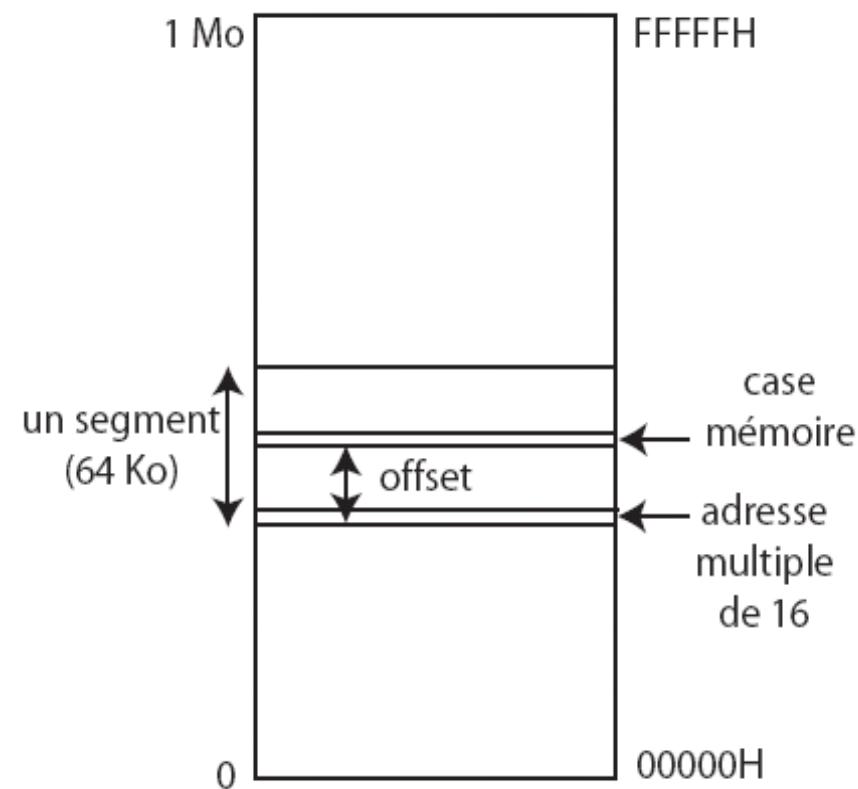
Registres	Usage
AX: Accumulateur	<ul style="list-style-type: none">▪ <i>Usage général,</i>▪ Obligatoire pour la multiplication et la division,▪ Ne peut pas servir pour l'adressage
BX : Base	<ul style="list-style-type: none">▪ <i>Usage général,</i>▪ Adressage
CX : Comptage et calcul	<ul style="list-style-type: none">▪ <i>Usage général,</i>▪ Compteur de répétition.▪ Ne peut pas servir pour l'adressage
DX : Data	<ul style="list-style-type: none">▪ <i>Usage général,</i>▪ Extension au registre AX pour contenir un nombre 32 bits Dans la multiplication et la division 16 bit▪ Ne peut pas servir pour l'adressage

ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

REGISTRES GÉNÉRAUX

Registres	Usage
SP : Pointeur de Pile	<ul style="list-style-type: none">Utilisé pour l'accès à la pile. Pointe sur la tête de la pile
BP : Pointeur de Base	<ul style="list-style-type: none"><i>Usage général</i>Adressage comme registre de base
SI : Registre d'index (source)	<ul style="list-style-type: none"><i>Usage général</i>Adressage comme registre d'index de l'opérande source.
DI : Registre d'index (destination)	<ul style="list-style-type: none"><i>Usage général</i>Adressage comme registre d'index de l'opérande destination

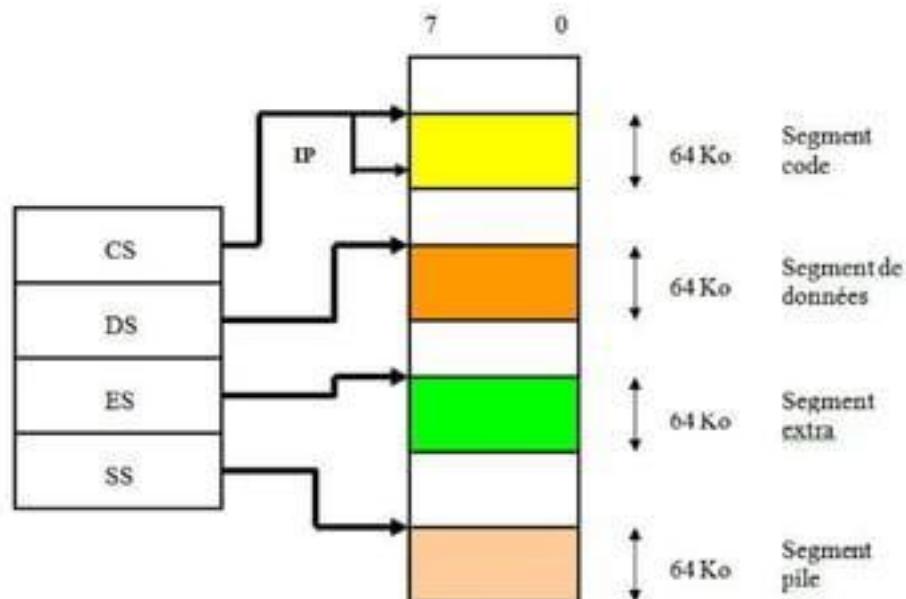
Ségmentation de la mémoire



ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

REGISTRES DE SEGMENT

- Ces registres sont combinés avec les registres offset (par exemple IP) pour former les adresses. Une case mémoire est repérée par une adresse de la forme $[R_{\text{seg}} : R_{\text{off}}]$
 - Le registre segment localise le début d'une zone mémoire de 64Ko
 - Le registre offset précise l'adresse relative par rapport au début de segment.



Calcul de l'adresse physique :

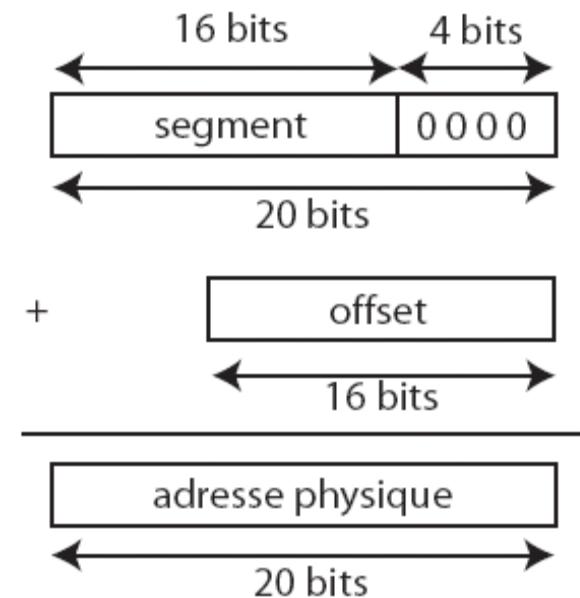
Adresse physique = 16 × segment + offset

Exemple :

l'adresse logique 3100h : 27EEh

l'adresse physique

$$31000h + 27EEh = 337EEh$$



ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

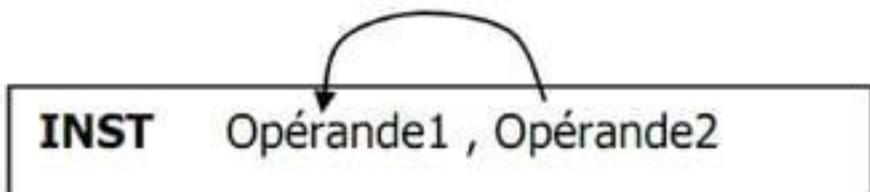
REGISTRES DE SEGMENT

Registres	Usage
CS : Code Segment	<ul style="list-style-type: none">Définit le début de la mémoire programme dans laquelle sont stockées les instructions du programme.Les adresses des différentes instructions du programme sont relatives à CS
DS : Data Segment	<ul style="list-style-type: none">Définit le début de la mémoire de données dans laquelle sont stockées toutes les données traitées par le programme.
SS : Stack Segment	<ul style="list-style-type: none">Définit le début de la pile.SP permet de gérer l'empilement et le dépilement.
ES : Extra Segment	<ul style="list-style-type: none">Définit le début d'un segment auxiliaire pour données

ARCHITECTURE GÉNÉRALE DU PROCESSEUR 8086

FORMAT D'INSTRUCTION

- La structure la plus générale d'une instruction est la suivante :



- L'opération est réalisée entre les 2 opérandes et le résultat est toujours récupéré dans l'opérande de gauche.
 - Il y a aussi des instructions qui agissent sur un seul opérande
- INST Opérande**
- Les opérandes peuvent être des registres, des constantes ou le contenu de cases mémoire.

Mode d'adressage immédiat

- Dans ce mode, l'opérande est directement spécifié dans l'instruction elle-même.
- Le CPU n'a pas besoin d'accéder à la mémoire pour récupérer la valeur de l'opérande.
- **Exemples :**

MOV AH,12H (donnée de 8 bits)

MOV BX,FFFFH (donnée de 16 bits)

Mode d'adressage registre

- Dans le mode d'adressage registre, l'opérande est spécifié en utilisant le nom d'un registre.
- L'opération se fait sur un ou 2 registres.
- Exemples

INC AX ; incrémenter le registre AX

MOV AX, BX ; Copier le contenu de BX dans AX

Mode d'adressage direct

- Un des deux opérandes se trouve en mémoire.
L'adresse de la case mémoire ou plus précisément son Offset est précisé directement dans l'instruction.
- L'adresse Rseg :Off doit être placée entre [], si le segment n'est pas précisé, DS est pris par défaut.
- **Exemples**

```
MOV AX, [243] ; Copier le contenu de la mémoire d'adresse DS:243 dans AX  
MOV [123], AX ; Copier le contenu de AX dans la mémoire d'adresse DS:123  
MOV AX, [SS:243] ; Copier le contenu de la mémoire SS:243 dans AX
```

Modes d'adressage indirect

- Un des deux opérandes se trouve en mémoire.
L'offset de l'adresse n'est pas précisé directement dans l'instruction, il se trouve dans l'un des 4 registres d'offset BX, BP, SI ou DI et c'est le registre qui sera précisé dans l'instruction : [Rseg : Roff]. Si Rseg n'est pas spécifié, le segment par défaut sera utilisé selon le tableau suivant :

Offset utilisé	Valeur	DI	SI	BX	BP
Registre Segment par défaut qui sera utilisé par le CPU			DS		SS

- Exemples

```
MOV AX, [BX] ;Charger AX par le contenu de la mémoire d'adresse DS:BX
MOV AX, [BP] ;Charger AX par le contenu de la mémoire d'adresse SS:BP
MOV AX, [SI] ;Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [DI] ;Charger AX par le contenu de la mémoire d'adresse DS:DI
MOV AX,[ES:BP] ;Charger AX par le contenu de la mémoire d'adresse ES:BP
```

Mode d'adressage indirect

Dans le mode d'adressage indirect , selon le registre d'offset utilisé, on distingue :

- l'adressage basé (avec/sans déplacement) ;
- l'adressage indexé (avec/sans déplacement) ;
- l'adressage basé indexé (avec/sans déplacement).

Adressage basé

- L'offset se trouve dans l'un des deux registres de base BX ou BP.
- On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset.
- **Exemples**

```
MOV AX, [BX] ; Charger AX par le contenu de la mémoire d'adresse DS:BX  
MOV AX, [BX+3] ; Charger AX par le contenu de la mémoire d'adresse DS:BX+3  
MOV AX, [BP-100]; Charger AX par le contenu de la mémoire d'adresse SS:BX-  
MOV AX, [ES:BP] ; Charger AX par le contenu de la mémoire d'adresse ES:BP
```

Adressage indexé

- L'offset se trouve dans l'un des deux registres d'index SI ou DI.
- On peut préciser un déplacement qui sera ajouté au contenu de RI pour déterminer l'offset.
- Exemples

```
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI  
MOV AX, [SI+200]; Charger AX par le contenu de la mémoire d'adresse DS:SI+  
MOV AX, [DI-7] ; Charger AX par le contenu de la mémoire d'adresse DS:ID-7  
MOV AX, [ES:SI+4]; Charger AX par le contenu de la mémoire d'adresse ES:SI+
```

Adressage basé indexé

- L'offset de l'adresse de l'opérande est la somme d'un registre de **base**, d'un registre **d'index** et /ou d'un **déplacement optionnel**.
- Si **Rseg** n'est pas spécifié, le segment par défaut du registre de base est utilisé.
- **Exemples**

MOV AX, [BX+SI] ; Charger AX par le contenu de la mémoire d'adresse DS:BX+SI

MOV AX, [BX+DI+7] ; Charger AX par le contenu de la mémoire d'adresse DS:BX+DI+7

MOV AX, [BP+SI-3] ; Charger AX par le contenu de la mémoire d'adresse SS:BP+SI-3

MOV AX, [BP+DI] ; Charger AX par le contenu de la mémoire d'adresse SS:BP+DI

Jeu d'instructions

- **Syntaxe** : MOV dest, source
 - **Description** : *dest* reçoit *source*.
-
- **Exemples** :
- MOV AL, 0 # $AL \leftarrow$
MOV n, AL # $n \leftarrow (AL)$

Instruction MOV

Quelques mouvements autorisés

MOV Registre général, Registre quelconque

MOV Mémoire, Registre quelconque

MOV Registre général, Mémoire

MOV Registre général, Constante

MOV Mémoire, Constante

MOV Registre de segment, Registre général

Instruction XCHG

- **Syntaxe :** `XCHG Destination, Source`
- **Description :** Échange les contenus de Source et de Destination.
- **Quelques mouvements autorisés :** XCHG Registre général,
Registre général
XCHG Registre général, Mémoire
XCHG Mémoire, Registre général
- **Exemples :**
 - XCHG AL, AH
 - XCHG Alpha, BX
 - XCHG DX, Beta

Instruction ADD

- **Description** : Addition de deux opérandes.

- **Syntaxe** : ADD Destination, Source

Destination \leftarrow Destination + Source.

La destination peut être : registre, mémoire.

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

- **Exemple**

MOV AL, 5 ; AL = 5

ADD AL, -3 ; AL = 2

Instruction INC

- **Description :** Incrémentation d'un opérande.
- **Syntaxe :**

INC Destination ;

Destination \leftarrow Destination+1.

La Destination peut être : registre, mémoire.

```
MOV CL, 4
```

```
INC CL ; CL = 5
```

Instruction SUB

- **Description** : Soustraction entre deux opérandes.

- **Syntaxe** : **SUB Destination, Source**
Destination ← Destination – Source.

La Destination peut être : registre, mémoire.

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

- **Exemples** :

MOV AL, 5

SUB AL, 1 ; AL = 4

Instruction DEC

- **Description** : Décrémentation d'un opérande
- **Syntaxe** :

DEC Destination ; Destination← Destination – 1

La Destination peut être : registre, mémoire.

MOV CL, 255 ; CL = OFFh (255)

DEC CL ; CL = OFEh (254)

Instruction MUL

- **Description :** Multiplication entre l'accumulateur et un opérande.
L'opérande peut être : registre, mémoire.
- **Exemple :**

```
MOV AL, 200 ; AL = 0C8h  
MOV BL, 4  
MUL BL ; AX = 0320h (800)
```

Instruction DIV

- Description : Division entre l'accumulateur et un opérande (non signé).
- Syntaxe : **DIV Opérande.**
- **Exemple :**

```
MOV AX, 203 ; AX = 00CBh  
MOV BL, 4  
DIV BL ; AL = 50 (32h), AH = 3
```

Exemples

a = b + c;

MOV AL, b #AL \leftarrow (b)

ADD AL, c # AL \leftarrow (AL) + (c)

MOV a, AL #a \leftarrow (AL)

Exemples

d = a - e;

MOV AL, a # AL \leftarrow (a)

SUB AL, e # AL \leftarrow (AL) - (e)

MOV d, AL # d \leftarrow (AL)

Exemples

$$f = (g + h) - (i + j);$$

```
MOV AL, i # AL ← (i)
ADD AL, j # al (AL) + (j)
MOV BL, AL # BL← (AL)
MOV AL, g # AL + (g)
ADD AL, h # AL ← (AL) + (h)
SUB AL, BL # AL (AL) - (BL)
MOV f, AL # f← (AL)
```

Instructions Logiques

Instruction AND

- ET logique entre deux opérandes bit à bit.

- Syntaxe :

AND Destination, Source ;

Destination \leftarrow Destination AND Source.

Destination peut être : registre, mémoire.

Source peut être : registre, mémoire, valeur immédiate.

- Exemple :

MOV AL, 01100001b

AND AL, 11011111b ; AL = 01000001b

Instruction OR

- OU logique entre deux opérandes bit à bit.
- Syntaxe :
OR Destination, Source
 $\text{Destination} \leftarrow \text{Destination OR Source.}$
Destination peut être : registre, mémoire.
Source peut être : registre, mémoire, immédiate.
- Exemple :
MOV AL, 01000001b
OR AL, 00100000b ; AL = 01100001b

Instruction XOR

- OU Exclusif entre deux opérandes bit à bit.

- Syntaxe :

XOR Destination, Source ;

Destination ← Destination XOR Source.

Destination peut être : registre, mémoire.

Source peut être : registre, mémoire, immédiate.

- Exemple :

MOV AL, 00000111b

XOR AL, 00000010b ; AL = 00000101b

Instruction TEST

- Effectue le ET logique entre deux opérandes et positionne uniquement les FLAGs sans modifier la destination.

- **Syntaxe :**

TEST Op1, Op2 ;

Résultat \leftarrow Op1 (ET) Op2 ; Puis la mise à jour des FLAGs.

Op1 peut être : registre, mémoire.

Op2 peut être : registre, mémoire, immédiate.

- **Exemple :**

MOV AL, 00000101b

TEST AL, 1 ; ZF = 0. TEST AL, 10b ; ZF = 1.

Instruction NOT

- Inversion bit à bit d'un opérande.
- Effectue le complément à 1.
- **Syntaxe :**

NOT Destination ;

Destination \leftarrow non (Destination).

Destination peut être : registre, mémoire.

- **Exemple :**

MOV AL, 00011011b

NOT AL ; AL = 11100100b

Instruction NEG

- Effectue le complément à 2.

- Syntaxe :

NEG Destination ;

Destination \leftarrow non (Destination) +1.

Destination peut être : registre, mémoire.

Exemple :

MOV AL, 5 ; AL = 05h NEG AL ; AL = 0FBh (-5)

NEG AL ; AL = 05h (5)