# Cours Architecture des ordinateurs
# Langages Assembleur et Machine

**Resp. Dr. Mohamed Feredj**
**Courriel : archiFeredj@gmail.com**

# 1) Introduction

Les langages hauts niveau permettent de cacher la complexité des machines afin de permettre aux programmeurs de manipuler des concepts plus élaborés (objet, composants, etc.) sans se préoccuper des détails de la machine. Exemple, Java et PHP. Par contre, les langages bas niveau permettent aux programmeurs de manipuler directement le matériel en utilisant des instructions spécifiques. Exemple, Langage Assembleur, langage machine.

## 1.1) Langage Machine

Langage Machine est très lourd pour programmer la machine et on doit se soucier des détails de la machine. De plus, le fait de manipuler des instructions sous forme binaire, il est très facile de se tromper. C'est pourquoi on a inventé l'assembleur.

## 1.2) Langage Assembleur

Est une description symbolique du langage machine. Cette description est basée sur une syntaxe plus compréhensible par les programmeurs.

# 2) Structure du programme assembleur

Tout programme assembleur est composé de :

- Segment de code (Obligatoire) ;
- Segment de données (si nécessaire) ;
- Segment de pile (si nécessaire) ;
- Segment étendu (si nécessaire) ;

La déclaration des segments est de la forme suivante :

***<Nom_Segment>  SEGMENT***

Contenu du segment

***<Nom_Segment>  ENDS***

*<Nom_Segment> : est un nom au choix du programmeur.*

Par contre, la déclaration du segment de code doit être comme suite :

```
<Nom_Segment>  SEGMENT
        Assume DS : <Nom_Segment1>, CS : <Nom_Segment2>
Debut : mov ax, <Nom_Segment1>
        mov ds, ax
        ; les instructions du programme
        ……..
<Nom_Segment>  ENDS
END   Debut
```

*Remarque:*

- *Assume indique au µP les segments du programme où se situent les données et les instructions.*
- *L'étiquette Debut indique le point d'entrée du programme.*

## 3) Déclaration des variables

2.1) Variable simple :

```
X    DB    10H, ?, 4H, 10, "A",  "B"
Y    DW    100H, 100, -5
Z    DD    3*20
```

2.2) Variable tableau :

```
Tab1        DB    100    DUP( ?)
Tab2        DB    2      DUP( 0,1,2,?)
```

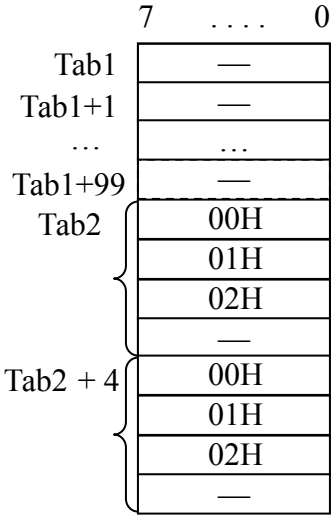



Tableau à 2 dimensions :

```
Tab    DB    100    DUP(100 DUP( ?))
```

## 4) Instructions assembleurs

La forme générale des instructions assembleurs est :

Etiquette : Opérateur Opérande1, Opérande2 *; commentaire*

*Remarque :*

- *Une instruction peut ne pas contenir des opérandes ;*
- *Si Opérande1 et Opérande2 sont présents alors ils sont appelés respectivement argument destinataire et argument source.*

Elles sont classées en 6 catégories (voir annexe jeu d'instructions 8086 pour plus de détails):

1) Instructions de déplacement des données (transfert des données) :
   mov, push, pop, lea, in, out, etc.
2) Instructions arithmétiques
   add, sub, mul, etc.
3) Instructions logiques
   and, or, xor, not.
4) Instructions de positionnement des flags (drapeaux) du mot d'état
   STC, STI, CLC, CLI, etc.
5) Instructions de test et comparaison :
   Test (et booléen), CMP (fait la différence)
6) Instructions de branchement (sauts)
   JMP, JZ, JE, CALL, INT, RET, IRET, etc.
7) Instructions portant sur les chaînes
   LODS, CMPS (comparaison de 2 chaînes), etc.
8) Pseudo-instructions
   PTR (avec BYTE, WORD, DWORD, etc.), ASSUME, OFFSET, SEG, STRUC et ENDS (pour définir un enregistrement – pas de sous enregistrement-), etc.
   *Exemple :*

```
Nom_Enreg       STRUCT
            champs1     DW    ?
            champs2     DB    ?
Nom_Enreg       ENDS
```

   Définir une var de type Nom_Enreg :

```
Nom_Enreg  Nom_Var <champ1=5, champ2=6>
```

   Au niveau du code assembleur :

```
MOV AX, Nom_Var.champ1
```

9) Instructions diverses :
   CWD (étendre un nombre de 8 bit de AX à DX = Convertir Word à DoubleWord), NOP.

*Remarque :*

*Les flags du registre d'état affectés par certaines opérations. Sur l'annexe des instructions assembleur 8088/8086, les valeurs des flags sont codées comme suit :*

1) *X : Flag est positionné par l'instruction en fonction du résultat.*
2) *U : Flag indéfini par l'instruction.*
3) *0/1 : Flag est fixé par l'instruction.*
4) *R : Restored flag (flag restauré par IRET).*

## 5) Mon premier programme assembleur

Ecrire un programme assembleur qui déclare deux variables de type Byte et calcule leur somme ?

```
; Mon premier programme qui fait
; la somme de deux variables
MesDonnees      SEGMENT
        X       DB      10H
        Y       DB      34H
MesDonnees      ENDS

MonCode         SEGMENT
        ASSUME          DS :MesDonnees, CS:MonCode
Main:   MOV     AX,     MesDonnees
        MOV     DS,     AX
        MOV     AX,     MonCode
        MOV     CS,     AX
        MOV     AL,     X
        ADD     AL,     Y
MonCode         ENDS
END     Main
```

# 6) Modes d'adressage (Cas du 8086)

### 6.1) Mode immédiat

*Opérateur* *Opérande1,* *Data* avec Data = donnée immédiate
Exemple : MOV AX, 1234H

### 6.2) Mode registre

*Opérateur* *Opérande1,* *Reg* avec Reg = Registre du µP
Exemple : MOV AX, BX

### 6.3) Mode indirect par registre

*Opérateur* *Opérande1,* *[Reg]* avec Reg = Registre du µP
Exemple : MOV AX, [BX]

### 6.4) Mode indirect par registre relatif

*Opérateur* *Opérande1,* *[Reg +Dep]* avec Reg = Registre du µP
Ou *Opérateur* *Opérande1,* *[Reg][Dep]* Dep = Déplacement relatif
Exemple : MOV AX, [BX+78H]

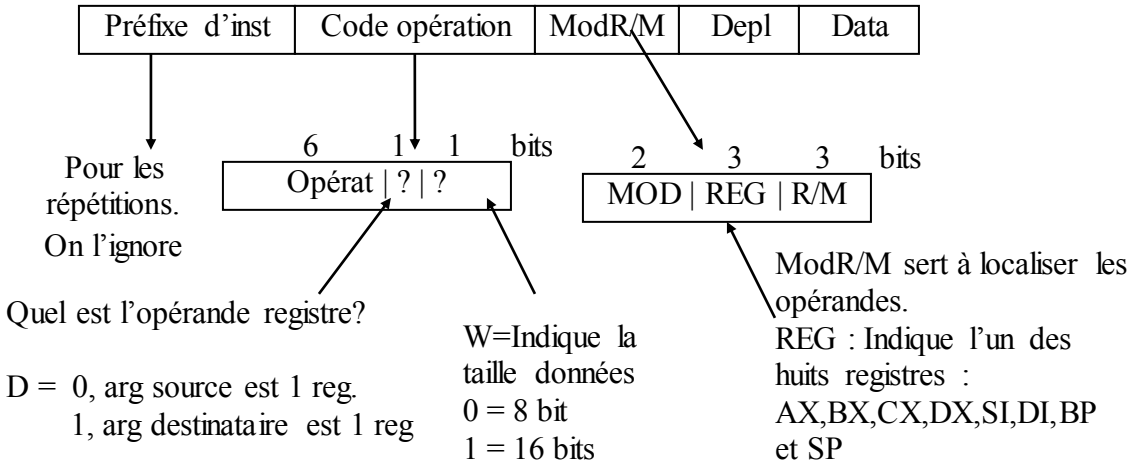
### 6.5) Mode indirect basé indexé

*Opérateur* *Opérande1,* *[RegB +RegI]* avec RegB = BX ou BP (Reg de base)
Ou *Opérateur* *Opérande1,* *[RegB][RegI]* RegI = SI ou DI (Reg d'index)
Exemple : MOV AX, [BX][SI]

### 6.6) Mode indirect basé indexé relatif

*Opérateur* *Opérande1,* *[RegB +RegI+Dep]* avec RegB = BX ou BP
Ou *Opérateur* *Opérande1,* *[RegB][RegI][Dep]* RegI = SI ou DI
Dep = Déplacement relati
Exemple : MOV AX, [BX][DI+56H]

## 7) Format des instructions en code machine




A partir du 80386, un nouveau champ (SIB) est ajouté pour prendre en charge des instructions plus évoluées,

Par exemple : Pour l'instruction suivante : ADD AX, 3456H, il s'agit d'une valeur immédiate. Donc,
Pas de déplacement ➔ 4 octets utilisés.