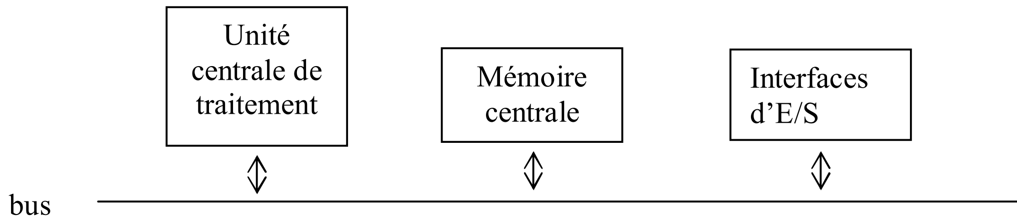
FACULTE DES SCIENCES DE TUNIS

Chapitre I : *LE LANGAGE ASSEMBLEUR*

1. Architecture générale d'un ordinateur :



La mémoire centrale :

- La mémoire centrale contient 2 types de d'informations : des instructions et des données.
  - ✓ Les instructions stockées sous forme de code machine.
  - ✓ Les données selon leurs types :
    a) Entiers : (représentés par codage binaire).
    b) Réels : (codage en virgule fixe ou en virgule flottante).
    c) Caractères : (code ASCII)

La mémoire est divisée physiquement en cellules dites mots mémoire. C'est l'unité d'information d'adressage.

L'unité centrale de traitement (le processeur) :

Il est formé de :

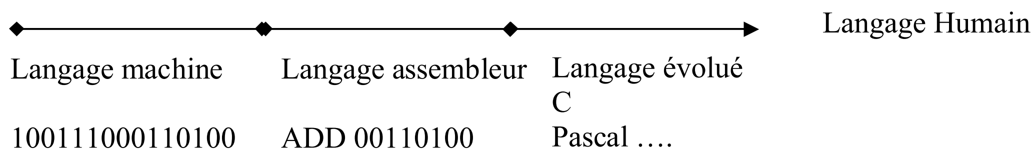
- ✓ Unité de commande : elle est responsable de la lecture en mémoire et du codage des instructions.
- ✓ Unité de traitement : dite unité algorithmique et logique, elle exécute les instructions qui manipulent les données.
- ✓ Les registres : ce sont des petites cases mémoire internes au processeur.
  - ❖ Le registre d'instructions RI : contient l'instruction en cours d'exécution
  - ❖ Le compteur ordinal : contient l'adresse de la prochaine instruction à exécuter.
  - ❖ Les registres de données.
  - ❖ Les registres de base ou d'index : permettent le calcul par rapport à une valeur de base.
  - ❖ Les registres d'état : indiquent le cas du système sur les opérations réalisés.

2. Introduction au langage assembleur :

Le processeur ne peut exécuter que les instructions écrites dans un langage binaire appelé langage machine. Ce langage est propre à chaque famille de processeurs.

Etant donné la forme de ce langage, un programmeur ne peut pas développer les programmes en utilisant un tel langage. Les langages assembleurs puissent exprimer les instructions en utilisant un code autre que le binaire, en plus ils sont très proches du processeur. Mais ils restent liés comme les langages machine à des familles de processeurs. (Le langage assembleur des processeurs INTEL est différent de celui des processeurs MOTOROLA). Ces langages utilisent un code qui utilise des symboles pour décrire des instructions du programme. Contrairement aux langages évolués (JAVA, C,…) le langage assembleur permet d'accéder à toutes les ressources (registres) et à faciliter le traitement par ordinateur.

Exemple : 3+4 :



Intérêt du langage assembleur :

- Le langage assembleur est intimement lié au processeur, ce qui permet de développer des programmes performants.
- Un programme en assembleur est exécuté plus rapidement que celui en langage évolué.

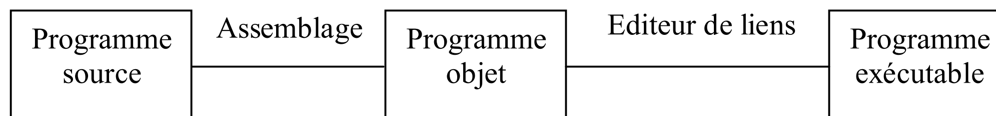
Mais :

- Le programme en assembleur est décomposé en instructions élémentaires, donc il est plus long, plus difficile à modifier et à corriger.

Avantages	Inconvénients
Rapide	Code trop long
Optimisé (moins d'espace mémoire)	Complexe
Accès direct au matériel	N'est pas portable

3. Assemblage et programme assembleur :

Le langage assembleur est un langage ***bas niveau***. Demander à un processeur d'exécuter un programme en assembleur n'est pas possible car les circuit du processeur ne comprennent que le binaire. Donc il faut traduire le programme assembleur en instructions en langage machine. Cette traduction est assurée par un logiciel spécifique appelé assembleur. Cette opération est dite assemblage.



L'éditeur de liens permet de faire les liens aux éléments (fonctions, bibliothèques, ….) auxquels le programme fait référence mais ne sont pas dans le fichier source. Puis

créer un programme exécutable qui contient tout ce qu'il lui faut pour fonctionner d'une manière autonome.

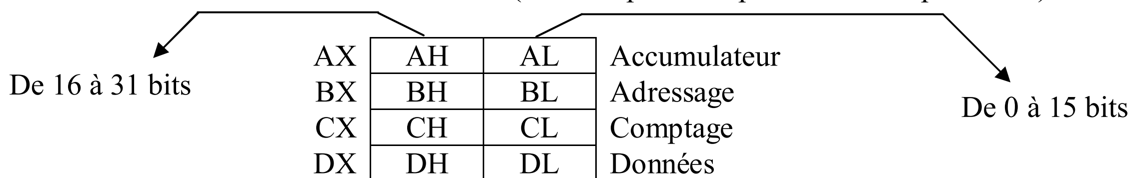
4. Registre 8086 :

Un registre est une suite de 8 ou de 16 bits, il sert à stocker les informations (données, adresses, etc.)
Les registres 8086 sont répartis en 4 catégories :

a) Registre de données (registres généraux) :

Cette catégorie est utilisé pour faire du calcul, elle contient 8 registres sur 2 octets chacun (16 bits), mais peuvent être regroupés par paire (4 registres de 4 octets chacun (32 bits)). Dans ce cas le nom général des registres est appelé ABCD, il est suivi du suffixe :

- X : (le mode groupé 16 bits).
- L : (le mode par octet pour l'octet de poids faible).
- H : (le mode par octet pour l'octet de poids fort).



AX	AH	AL	Accumulateur
BX	BH	BL	Adressage
CX	CH	CL	Comptage
DX	DH	DL	Données

b) Registres d'adressage (pointeurs et index) :

SI	Index source
DI	Index destination
BP	Pointeur de base

Ce sont des registres de 16 bits utilisés pour l'assemblage c à d pour produire des adresses d'opérantes.

c) Registre de commandes (pointeurs d'instructions et d'indicateurs) :

SP	Pointeur de pile
IP	Pointeur d'instructions
FLAGS	Indicateur

Ces registres ont une taille de 16 bits. Le registre IP est similaires au compteur ordinal : il contient l'adresse de la prochaine instruction à exécuter. Le registre FLAGS contient des indicateurs (retenue, reste, signe, etc.). ces indicateurs servent pour le contrôle des opérations arithmétiques et logiques.

d) Registre de segments :

CS	Segment code

DS	Segment de données
SS	Segment pile

Ce sont des registres de 16 bits qui pointent vers des segments mémoire en activité. En effet un programme assembleur rangé en mémoire centrale occupera au moins 3 zones mémoire appelées segments :

- Pile de sauvegarde (zone de rangement de sécurité pointé par CS).
- Données pointées par DS.
- Programme ou code (liste des instructions à exécuter pointée par CS).

5. Les instructions :

Etiquette :	Code opération	Opérande(s)

Chaque instruction est représentée par une ligne de codes. En assembleur, une instruction est divisée en champs. Les différents champs d'une instruction sont généralement séparés par un ou plusieurs espaces. Le nombre d'opérandes du 3ème champs varie d'une machine à l'autre (0 à 3), après ce 3ème champs, il est recommander d'ajouter des commentaires.

Exemple :

```
Donnée1 :   DW      1
Adresse1 :  MOV     AX, Donnée1
            ADD     AX, BX
            JMP     Adresse2
```

- La 1ère instruction (DW) appelée directive permet de définir la variable Donnée1 et de lui réserver un *mot mémoire*.
- La 2ème instruction portant l'étiquette Adresse1, transfère le contenu de la variable Donnée1 dans le registre AX.
- La 3ème instruction effectue une addition entre deux registres et stocke le résultat dans le 1er registre AX.
- La dernière instruction effectue un saut inconditionnel à l'adresse Adresse2 qui se trouve quelque part dans le programme.

❖ ***Opérandes et étiquettes :***

Contrairement au langage machine, le langage assembleur permet de donner des noms alphanumériques aux variables et aux étiquettes (adresses des instructions) ce qui facilite beaucoup la programmation. Par exemple, si on veut faite un branchement en langage machine, on doit donner en binaire la position mémoire exacte où se trouve l'instruction à laquelle on veut se brancher. Dans le langage assembleur, il suffit de faire précéder l'instruction où l'on veut se brancher d'une étiquette comme opérande de l'instruction de branchement.
De la même façon, pour les opérandes, on n'est pas obligé de donner l'adresse binaire exacte de chacune d'elles. Les opérandes ont un nom qui permet de les référencer comme pour les variables dans le cas des langages évolués.

6. Principales instructions du microprocesseur 8086 :

ADD	AX, val	Ajouter à AX la valeur val et stocker le résultat dans AX.
ADD	AX, adr	Ajouter à AX la valeur stockée dans adr.
CMP	AX, val	Compare AX et la valeur val.
CMP	AX, adr	Compare AX et la valeur stockée à l'adresse adr.
DEC	AX	Décrémente AX (soustraire 1).
INC	AX	Incrémente AX (ajouter 1).
JA	Adr	Saut à l'adresse adr si CF=0.
JB	adr	Saut à l'adresse adr si CF=1.
JE	Adr	Saut à l'adresse adr si égalité.
JG	Adr	Saut à l'adresse adr si supérieur.
JLE	Adr	Saut à l'adresse adr si inférieur.
JNE	Adr	Saut à l'adresse adr si non égalité.

7. Expressions arithmétiques :

Contrairement au langage évolué, les expressions arithmétiques utilisés pour calculer la valeur d'une variable comme par exemple l'instruction d'affectation : A=B+C/D ne sont pas admises dans le langage d'assemblage. Elles doivent être programmés en utilisant plusieurs instructions. Par exemple, l'affectation doit être décomposée en actions élémentaires (division, addition, affectation), ainsi une instruction simple d'un langage évolué peut donner une séquence d'instructions en assembleur.

8. Macro et sous-programme :

Certain assembleurs permettent de structurer des programmes. Ainsi, il est possible de grouper une séquence d'instructions assembleur sous forme d'une unité appelé *sous programme* ou *macro instruction*. Ces deux structures ont pour but de modéliser le programme et d'éviter l'écriture répétée d'un groupe d'instructions fréquemment utilisé.

a) Macro :

L'idée d'une macro instruction ou d'une macro consiste à isoler la séquence d'instructions que l'on veut éviter de répéter et à lui attribuer un nom symbolique par lequel on peut lui faire référence. Chaque fois dans le programme on fait référence à ce nom, l'assembleur le remplace par la séquence d'instructions correspondante. L'utilisation de macro présente plusieurs avantages, elle permet d'abord d'étendre le jeu d'instructions de la machine car chaque macro peut être utilisée comme toute autre instruction, ensuite les programmes source sont plus courts. Les instructions qui servent à définir et à délimiter une macro (MACRO et ENDM). Lors de l'assemblage, chaque appel à une macro est remplacé par le corps de la macro et ces deux pseudo-instructions (MACRO et ENDM) sont éliminées.

Exemple : Calcul d'un cube d'un nombre

```
MACRO CUBE(valeur, valeur_cube)
        MOV         AX, valeur
        MOV         BX, valeur
```

```
        MUL         AX, BX
        MUL         AX, BX
        MOV         valeur_cube, AX
ENDM
```

b) Sous programme :

Les sous programmes sont définis comme les Macros, ils ont comme objet d'éviter la répétition des séquences d'instructions que l'on veut utiliser plusieurs fois. La principale différence entre macro et sous programme : les appels à une macro sont remplacés par le corps de la macro pendant la traduction alors que les appels à un sous programme sont traités lors de l'exécution.

9. Structure du programme assembleur :

Un programme assembleur est organisé sous forme de segments. Le stockage d'un programme assembleur en mémoire centrale occupera au moins 3 zones mémoire (segments), dont l'assembleur se chargera d'établir les adresses physiques.
Ces zones constituent les segments du programme :

- Pile de sauvegarde : zone de rangement de sécurité.
- Données.
- Code : la liste des instructions qui doivent être exécutées.

Chacune de ces zones saura pointée par un registre segments :

- Registre SS : pointe vers la pile de sauvegarde.
- Registre DS : pointe vers le segment de données.
- Registre CS : pointe vers le segment code.

```
TITLE       nom_programme

Pile    SEGMENT    STACK
        ………….., déclaration de la pile.
Pile ENDS

Données     SEGMENT
        ………….., déclaration des données.
Données ENDS

Code    SEGMENT
nom_programme   PROC        FAR             (1)

ASSUME      CS :code, DS : données, SS : pile  |
MOV         AX, données                         |  (2)
MOV         DS, AX                              |
```

nom_programme	ENDP	Indique la fin de la prcédure
	ENDS	Indique la fin du segment code
	END	Indique la fin du programme

(1) Un programme peut être divisé en séquences complètes appelées des procédures. Chaque procédure porte un nom et son début est indiqué par la directive PROC et comporte une information (la situation par rapport à la séquence précédente). Cette information peut être :

NEAR (proche).
FAR (loin).

On spécifie NEAR lorsqu'on veut indiquer à l'assembleur que cette procédure se situe dans le même segment que la séquence précédente. On spécifie FAR quand on veut changer le segment.
(2)L'écriture suivante est toujours obligatoire :

ASSUME : signifie que l'on indique à l'assembleur qu'il doit affecter à :

- CS : l'adresse de départ du segment code.
- DS : l'adresse de départ du segment de données.
- SS : l'adresse de départ du segment pile.

En plus de la directive ASSUME, on doit aussi faire une affectation explicite des données dans DS. On remarque ici que l'affectation est effectuée sur deux étapes. La raison c'est qu'on ne peut changer un registre qu'à partir d'un registre.

10.Les interruptions :

En assembleur, chaque interruption porte un numéro de référence(N° en hexadécimal suivi de h). lorsque le microprocesseur rencontre une instruction d'interruption notée INT avec son numéro, il accède au début de la mémoire centrale dans une partie appelée (***Table des vecteurs d'interruption***). Il trouve dans cette table d'adresses le programme qu'il doit exécuter en réponse à cette interruption. Il l'exécute puis revient au programme interrompu. En ce qui concerne les opérations d'entrées/sorties (tel que l'affichage à l'écran, saisie au clavier) on fait appel au système d'exploitation *MS-DOS* pour réaliser ces opérations. Pour cela, il faut déclancher une interruption du programme en cours pour donner la main au *DOS*. Par exemple, l'interruption à déclancher dans le cas d'un affichage est l'interruption N°**21h** (spécifique au système MS-DOS). Mais comme le MS-DOS est très vaste, il faut en plus spécifier quelle séquence MS-DOS nous voulons exécuter. C'est le rôle de l'instruction :
MOV AH, valeur qui précise la nature de l'opération à effectuer.
Exemples :

- Affichage d'un caractère

```
MOV        DL, "A"      ; Caractère.
MOV        AH, 2        ; Fonction N°2.
INT        21h          ; Appel système.
```

- Saisie d’un caractère (avec echo)

```
MOV     AH, 1     ; attend un caractère au clavier et le met dans AL et l’affiche.
INT     21h
```

- Saisie d’un caractère (sans echo)

```
MOV     AH, 7     ; attend un caractère au clavier et le met dans AL sans l’afficher.
INT     21h
```

- Arrêt du programme

```
MOV  AH, 4ch      ; arrêt du programme.
INT  21h
```

- Saisie d’une chaîne de caractères

```
MOV     AH, 10
INT     21h
```

IF THEN ELSE

```
Si (AX = =1)
BX =10 ;
sinon
{
BX =0 ;
CX = 10 ;
}
Endif
```

```
IF : CMP AX,1
     JNE    ELSE
THEN : MOV  BX, 10
       JMP   ENDIF
ELSE : MOV   BX, 0
       MOV  CX, 10
ENDIF :
```