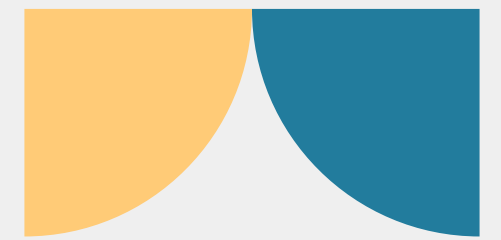




LANGUAGE ASSEMBLEUR



Realiser par:

- **Osama Jammi**
- **Walid Islah**
- **Marwane El Mazouzy**
- **Ziad Lachguer**
- **Youssef Outakhrouft**

LES AXES:

01 - INTRODUCTION

02 - PROGRAMMATION EN ASSEMBLEUR


03 - DES EXEMPLES PRATIQUES





INTRODUCTION

L'assembleur, ou « langage d'assemblage » est constitué d'instructions directement compréhensibles par le microprocesseur : c'est ce qu'on appelle un langage de bas niveau. Il est donc intimement lié au fonctionnement de la machine. C'est pourquoi il est relativement difficile à assimiler, en tout cas beaucoup plus que les langages de haut niveau. Mais l'avantage du langage assembleur est de pouvoir gérer jusqu'au moindre octet de mémoire et de toujours savoir quel code est exécuté par le microprocesseur à un instant donné.





HISTOIRE

Les premiers programmes informatiques étaient rédigés en utilisant des mnémoniques alphabétiques pour chaque instruction. La traduction était alors faite à la main par les programmeurs. Cette opération était longue, fastidieuse et entachée d'erreurs.

Le premier programme assembleur a été écrit par Nathaniel Rochester pour l'IBM 701 en 1954.

Les langages d'assemblage ont éliminé une grande partie des erreurs commises par les programmeurs de la première génération d'ordinateurs, en les dispensant de mémoriser les codes numériques des instructions et de faire des calculs d'adresses. La programmation en assembleur était alors utilisée pour écrire toutes sortes de programmes.

USAGE

Il y a des débats sur l'utilité du langage assembleur. Dans beaucoup de cas, des compilateurs-optimiseurs peuvent transformer du langage de haut niveau en un code qui tourne aussi efficacement qu'un code assembleur écrit à la main par un très bon programmeur, tout en restant beaucoup plus facile, rapide (et donc moins coûteux) à écrire, à lire et à maintenir. Et réduisant les risques d'un bug (parce que lisible, parce que les compilateurs détectent automatiquement de plus en plus d'erreurs ou de warnings). Il y a aussi des programmes spéciaux qui font des suggestions pour écrire mieux le code ou signaler quelque chose non repéré par le compilateur.



"PROGRAMMATION EN ASSEMBLEUR"

LES REGISTRES DU 8086

1. Les registres de travail

AX: Accumulateur principal

BX: Spécialisé dans l'adressage indexé

CX (Count): utilisé pour le comptage

DX (Data): utilisé par la multiplication et la division

LES REGISTRES DU 8086

2. Les registres d'offset

EBP : Extended Base Pointer – pointeur de base

ESP : (Extended Stack Pointer) pointeur de pile

ESI : (Extended Source Index) pointeur source

EDI : (Extended Destination Index) pointeur destination

LES REGISTRES DU 8086

3. Les registres de segment

CS : pointe vers les instructions du programme

DS : pointe vers les données du programme

**ES : pointe vers les données du programme
multi-segments**

**FS : pointe vers les données du programme
multi-segments en mode protégé.**

**GS : pointe vers les données du programme
multi-segments en mode protégé.**

LES MODES D'ADRESSAGE DU MICROPROCESSEUR 8086

Un mode d'adressage est la méthode de localisation des opérandes qui interviennent dans une opération.

MODE D'ADRESSAGE IMMEDIAT

2. Adressage immédiat : l'opérande est une constante fournie immédiatement dans l'instruction.

Exemple : MOV DX, 4 ; copie dans DX la valeur 4 (codé sur 16 bit)

MOV AL, 12 ; copie dans AL la valeur 12 (8 bits)

ADD AX, 177 : additionner AX avec la valeur 177h

MODE D'ADRESSAGE REGISTRE

- Dans le mode d'adressage registre, l'opérande est spécifié en utilisant le nom d'un registre.
- Les deux registres doivent être de la même taille

Exemples :

INC AX ; incrémenter le registre AX

MOV AL, BL ; Copier le contenu de BL dans AL

MODE D'ADRESSAGE DIRECT

- Un des deux opérandes se trouve en mémoire.
- L'adresse de la case mémoire ou plus précisément son Offset est précise 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

MODE 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 on distingue :

- l'adressage basé
- l'adressage indexé
- l'adressage basé indexé

MODE D'ADRESSAGE INDIRECT

Exemples d'adressage indirect

Adressage basé

-Il utilise les registres de base (BX,BP)

exemple:

MOV DX, [BX] ; copie dans DX le mot d'adresse DS:BX

MOV 5[BP], AL ; AL est copié dans l'octet d'adresse SS:(BP+5)

Adressage indexé

Il utilise les registres (SI,BI)

exemple:

MOV AH, [SI] ;AH ← le contenu de la case memoire d'adresse DS:SI

MOV AH, [SI+1000] ;AX←le contenu de la case memoire d'adresse DS:SI+1000

MODE D'ADRESSAGE INDIRECT

Exemples d'adressage indirect

Adressage basé indexé

C'est un mélange entre le mode d'adressage basé et le mode d'adressage indexé

-Il utilise les registres d'index (SI,DI) ainsi que les registres de base (BX,BP) comme pointeur

exemple:

MOV AL, [BP+SI-8] ;AX← le contenu de la case memoire
DS:BP+SI-8

DÉCLARATION

ASSEMBLY

```
section .data
i dd 10
a dq 3.14
a dq 3.14
my_string db 'Hello, world!',0
```

descripteurs de taille suivants
DB (Define Byte : 1 octet pour 8086)
DW (Define Word : 2 octets pour 8086)
DD (Define Double : 4 octets)

C

```
int i= 10;

float a= 3.4;

char my_string[] = "Hello";
```

LES OPÉRATEURS ARITHMÉTIQUES

C

```
int a = 10, b = 5, c;
```

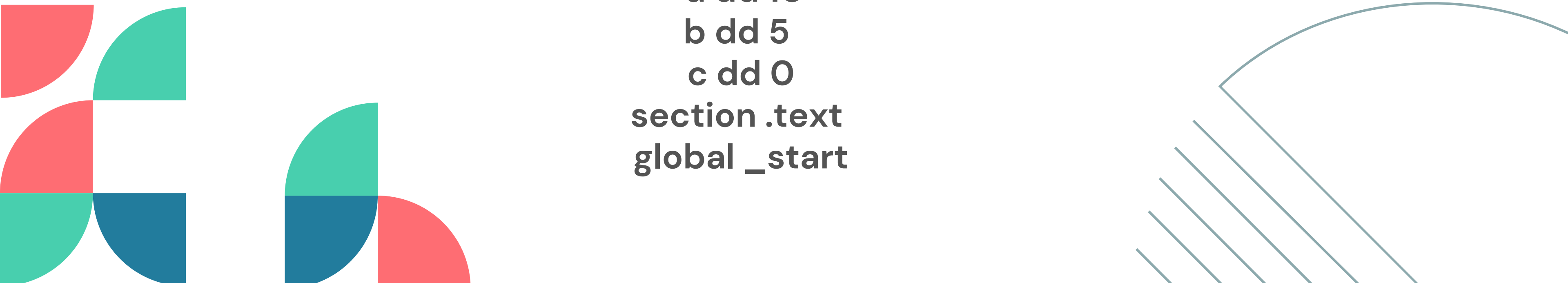
```
c = a + b;  
c = a - b;  
c = a * b;  
c = a / b;  
c = a % b;
```



LES OPÉRATEURS ARITHMÉTIQUES

ASSEMBLY

```
section .data  
    a dd 10  
    b dd 5  
    c dd 0  
section .text  
global _start
```



LES OPÉRATEURS ARITHMÉTIQUES

ASSEMBLY

_start:

; Addition

```
mov al, a  
mov bl, b  
add al,bl
```

; Soustraction

```
mov al, a  
mov bl, b  
sub al,bl
```

; mltn

```
mov al,a  
mov bl,b  
mul bl
```

; div

```
mov ah,0  
mov al,a  
mov bl,b  
mov dx,0  
div bl
```

```
mov eax, dword [a] ; Charge la valeur de  
a dans le registre eax
```

LES OPÉRATEURS RELATIONNELS COMPARENT

```
int a = 10;  
int b = 5;  
int c;
```

```
c = a == b;  
c = a < b;  
c = a <= b;  
c = a > b;  
c = a >= b;
```

```
mov al, a  
cmp al, b  
je equal  
jg a_greater
```

LES BOUCLES

```
#include <stdio.h>
```

```
int main() {  
    int ax = 0;  
  
    do {  
        ax++;  
    } while (ax < 15);  
  
    printf("%d", ax);  
  
    return 0;  
}
```

```
boucle:  
    inc ax  
    cmp ax,15  
    jl boucle
```

```
mov dx,ax  
mov ah,02h  
int 21h
```



LES FONCTION

CALCUL DE FACTORIELLE

```
mov ax,n  
mov bx,1
```

boucle :

```
mul bx  
inc bx  
cmp bx,n  
jl boucle
```

```
mov dx,ax  
mov ah,02h  
int 21h
```




**"DES EXEMPLES
PRATIQUES"**

The background features four decorative geometric patterns in the corners. The top-left corner has a series of parallel diagonal lines. The top-right corner contains a cluster of overlapping semi-circles in yellow, red, teal, and blue. The bottom-left corner features a similar cluster of overlapping semi-circles in red, teal, blue, and red. The bottom-right corner has a large, faint semi-circle outline with several parallel diagonal lines inside it.

**MERCI POUR
VOTRE ATTENTION!**