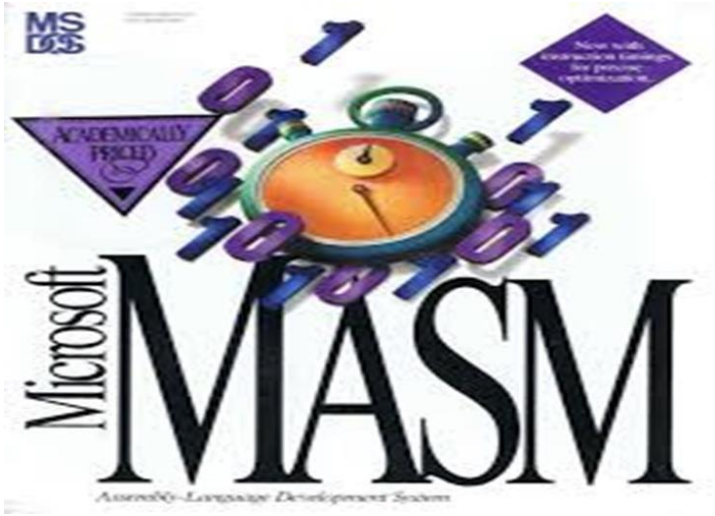
ASSEMBLEUR

Module N°3



BTS DSI

2020/2021

L'OBJECTIF DE CE COURS

L'objectif de ce cours est de savoir:

- Le fonctionnement interne d'un ordinateur.
- Comment le processeur exécute les instructions stockées dans la RAM.
- Quelles sont les opérations élémentaires que le microprocesseur pourra exécuter.
- Programmer avec des langages de bas niveau proche de langage binaire compris par le matériel.
- Comment le processeur exécute les conditions « if » et les boucles « for,while,… » de langage de haut niveau (C,VB,…)

Partie I

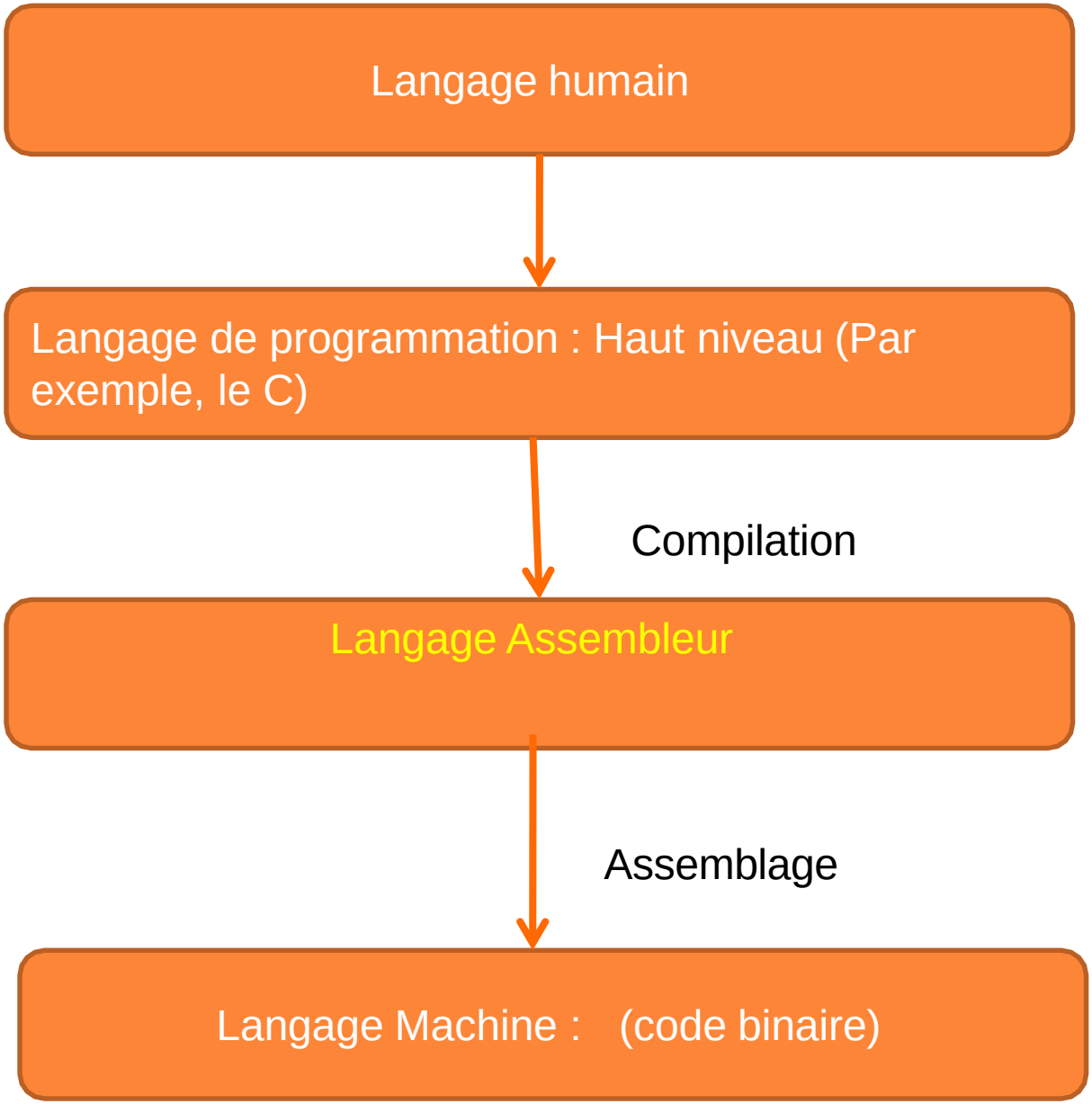
- Introduction à l'assembleur
- Instruction mov
- Instructions des opérations (add,div,and,or,…)
- Instructions de branchement (jmp,jnz,…)
- Instruction de comparaison (cmp)
- Appel des fonctions (invoke,call)

I-INTRODUCTION À L'ASSEMBLEUR:

BTS
DSI

I.1 DÉFINITION DE L'ASSEMBLEUR:

- est le langage de programmation de plus bas niveau. Cela signifie qu'il est trop proche du matériel, qui oblige le programmeur à se soucier de concepts proches du fonctionnement de la machine, comme la mémoire, processeur.
- transforme un fichier source contenant des instructions, en un fichier exécutable que le processeur peut comprendre.
- Les programmes faits en ASM sont plus petits, plus rapides et beaucoup plus efficaces que ceux fait avec des compilateurs(C,java,vb,…)


Langage humain
Langage de programmation : Haut niveau (Par exemple, le C)
Compilation
Langage Assembleur
Assemblage
Langage Machine : (code binaire)

Haut niveau

Langage humain :
"Si x est plus grand que 10,
alors décrémenter x..."

Langage haut niveau (C, PHP, ...) :
"if(x > 10) x–;"

Langage assembleur :
"cmp x, 10 dec: dec x
jb dec "

Langage machine :
"00111010011101101101101010111010000011000110111011011001101001"

Bas niveau

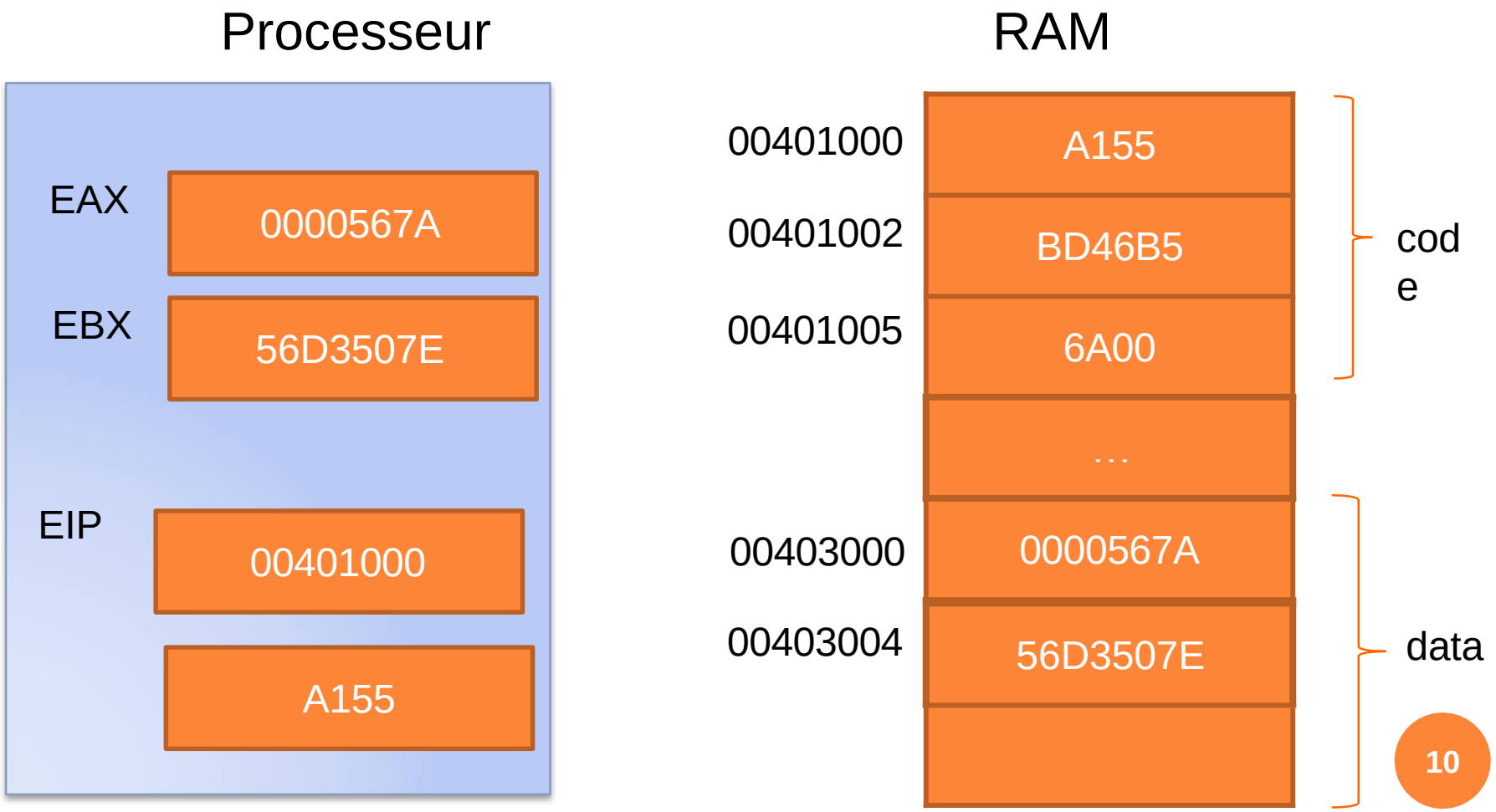
I.2 Le choix de langage assembleur

- Vu que l'assembleur est qualifié comme étant le langage de programmation le plus bas niveau, il dépend donc fortement du type de processeur. Ainsi il n'existe pas un langage assembleur, mais un langage assembleur par type de processeur.
- Dans notre cours on va utiliser assembleur MASM32 (Macro Assembleur de Microsoft) pour la famille de processeurs x86 32bits.
- MASM32 maintenu par Steve Hutchesson permet de programmer directement et relativement aisément des applications 32 bits fonctionnant sous Windows

II- Le processeur(CPU) et la mémoire(RAM)

BTS
DSI

II.1 FONCTIONNEMENT D'UN PROGRAMME



II.2 LES REGISTRES

Il existe plusieurs types de registres et chacun a son utilité.

- **registres généraux (EAX, EBX, ECX, EDX)**
  Ils servent à manipuler des données, à transférer des paramètres lors de l'appel de fonction DOS et à stocker des résultats intermédiaires.

- **registres d'offset ou pointeur (EIP, ESI, EDI, ESP, EBP)** Ils contiennent une valeur représentant un offset à combiner avec une adresse de segment

- **registres de segment(CS, DS, SS, ES, FS, GS)**
  Ils sont utilisés pour stocker l'adresse de début d'un segment. Il peut s'agir de l'adresse du début des instructions du programme, du début des données ou du début de la pile.

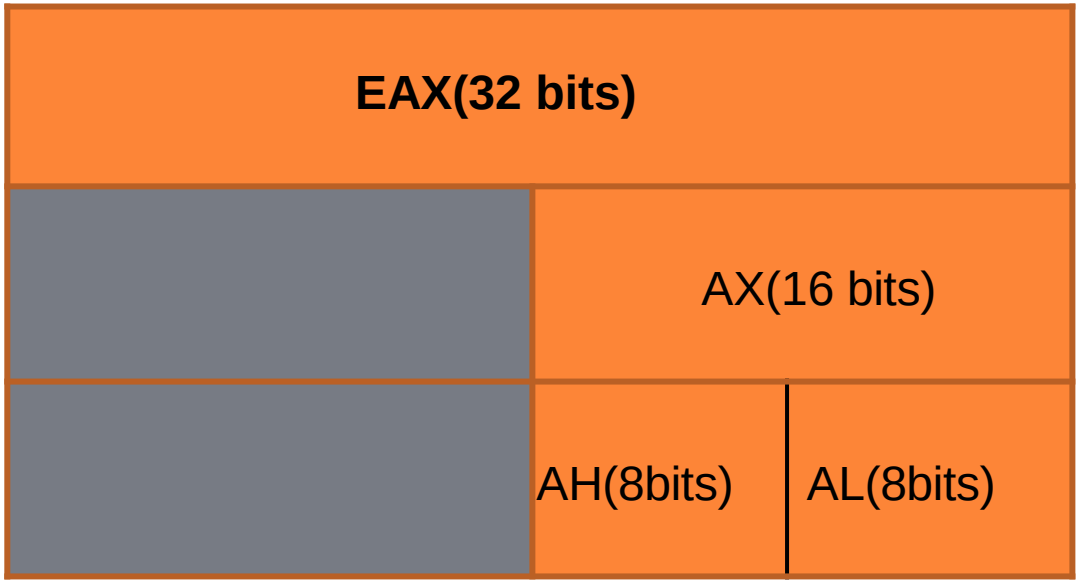
- **Un registre de flag**
  Il contient des bits qui ont chacun un rôle indicateur.

II.3 Les registres généraux

- **EAX** -- **accumulateur** -- sert à effectuer des calculs arithmétiques ou à envoyer un paramètre à une interruption
- **EBX** -- **registre auxiliaire de base** -- sert à effectuer des calculs arithmétiques ou bien des calculs sur les adresses
- **ECX** -- **registre auxiliaire (compteur)** -- sert généralement comme compteur dans des boucles
- **EDX** -- **registre auxiliaire de données** -- sert à stocker des données destinées à des fonctions

Ceci est leur utilisation théorique, mais dans la pratique ils peuvent être utilisés à d'autres usages.

II.3 Registre EAX



II.4 Jeu d'instructions d'un CPU

Décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter.

- Transfert de données: charger ou sauver en mémoire (mov , …)
- Opérations arithmétiques (add ,mul ,div , …)
- Opérations logiques(and ,or ,…)
- Contrôle de séquence :
  - Branchement (jmp, jnz…) « c'est le saute »
  - Test (cmp,…) « c'est la comparaison »

III- LANGAGE MASM32

BTS
DSI

III.1 DÉCLARATION DE VARIABLES

Les variables se déclarent de la manière suivante:

- en 8bits : **nom_Variable DB** valeur
- en 16bits : **nom_Variable DW** valeur
- en 32bits : **nom_Variable DD** valeur

De manière générale:

DB (**D**eclare **B**yte) : 1 byte (8 bits)
DW (**D**eclare **W**ord) : 1 word (16 bits)
DD (**D**eclare **D**ouble) : 2 words (32 bits)

Les valeurs 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*

Exemple:

```
var1 db 6 ; var1 est un byte initialisé à 6 en décimal
var2 db 0FFh ; var2 est un byte initialisé à FF en hexadécimal
var3 dw 67h ; var3 est un word initialisé à 67 (16 bits)
var4 dd 67A3h ; var3 est un double word initialisé à 67A3 (32 bits)
```

III.2 STRUCTURE DE CODE MASM32

```
; entête de code ***************************
.386 ; processeur = Pentium
.model flat, stdcall ; un seul segment de 4Go
; déclaration des variables*****************
.data
var1 dd ...
...
; code masm32*****************************
.code
start:
... ; les instructions de base masm32
end start
```

III.3 INSTRUCTION MOV

mov destination, source

L'instruction la plus utilisée est l'instruction mov, qui copie la valeur d'un opérande source dans un opérande destination. La syntaxe est la suivante :

- mov reg, reg (registre à registre)
- mov reg, mem (mémoire à registre)
- mov mem, reg (registre à mémoire)
- mov reg, imed (registre à valeur)
- mov mem, imed (mémoire à valeur)

NOTE: Pas de transfert de mémoire à mémoire

III.4 INSTRUCTIONS DE BASE **ARITHMÉTIQUE**

- Incrémentation

```
INC EAX              ; EAX <- EAX + 1
INC ma_variable
```

- Décrémentation

```
DEC EAX
DEC ma_variable
```

- Addition

```
ADD EAX, 5           ; EAX <- EAX + 5
ADD BH, var          ; BH <- BH + var
ADD var, ECX         ; var <- var + ECX
```

- Soustraction

```
SUB  EAX,  5         ; EAX <- EAX – 5
SUB   BH,var         ; BH <- BH – var
SUB var, CX          ; var <- var – CX
```

- Multiplication

```
MUL EBX              ; EAX <- EBX * EAX
```

- Division

```
DIV EBX              ; EAX <- EAX / EBX
                     ;Il faut que EDX=0
```

EXEMPLE1:

Programme masm32 qui calcule somme de 6A+B5 et stock le résultat dans EAX

```
.386
.model flat, stdcall
.data ; variables globales initialisées
x dd 6Ah
y dd 0B5h
.code
start:
 mov EAX, x
 add EAX, y
end start
```

III.5 INSTRUCTIONS DE BASE **LOGIQUE**

- AND bit à bit

```
MOV AL, 0101b          ; AL <- 5
MOV BL, 1001b          ; BL <- 9
AND AL, BL     ; AL <- AL AND BL; AL vaut 0001b, soit 1
```

- OR bit à bit

```
MOV AH, 0101b          ; AH <- 5
MOV BH, 1001b          ; BH <- 9
Or  AH, BH     ; AH <- AH OR BH; AH vaut 1101b , soit 13
```

III.6 Appel d'une fonction

invoke fonction a, b, c ; appelle fonction(a, b, c)

;Le résultat d'une fonction est toujours dans **eax**

Remarque: pour appeller une fonction qui utilise des bibliothèques, il faut les importer par le mot clé include ou bien includelib

include pour inclure les biblios .inc

includelib pour inclure les biblios .lib

III.6.2- fonction prédéfini d'affichage à l'écran

Syntaxe :

- invoke **StdOut** , ADDR msg

Avec msg : le texte à afficher à l'écran

Remarque: pour utiliser la fonction **StdOut ,** il faut importer les bibliotheques :

include \masm32\include\kernel32.inc

include \masm32\include\masm32.inc

includelib \masm32\lib\kernel32.lib

includelib \masm32\lib\masm32.lib

EXEMPLE2:

Code masm32 qui afficher un message à l'écran

```
.386 ; processeur = Pentium
.model flat, stdcall    ; un seul segment, appel standard
;-----------------------------------------------------------------------
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
;-----------------------------------------------------------------------
.data
msg db "Bonjour   : ",0
.code
start:
invoke   StdOut , ADDR msg
ici:
 jmp ici
end start
```

III.6.3 fonction de la conversion de hexadécimal to ascii

Syntaxe :

invoke **dwtoa** , eax, ADDR z

La valeur hexadécimal de eax sera converti en ascii

Exemple: eax = F -> z =15

EXEMPLE3 :

Code masm32 qui calcule et affiche la somme de 2 nombres

```
.data ; variables globales initialisées
x dd 10
y dd 11
.data?
z dd ? ; variable globale non initialisée
.code
start:
 mov eax, x
 add eax, y
 invoke dwtoa , eax, ADDR z
 invoke StdOut ,ADDR z
 ici: jmp ici
end start
```