## ARCHITECTURE DES ORDINATEURS

#### 6)- Introduction à l'ASSEMBLEUR 'x86'

```
Biblio :: 1/« Assembly Langage for INTEL-based computers»

[Kip R. IRVINE] – Ed. Prentice Hall, 1999 – ISBN: 0-13-660390-4.

2/« An assembly langage introduction to computer architecture (using the intel pentium)»

[K. MILLER] – Ed. Oxford University Press, 1999 – ISBN: 0-19-512376-X

3/«The Intel microprocessors : Architecture, Programming & Interfacing»

[Barry B. BREY] – Ed. Prentice Hall 2006 – ISBN: 0-13-119506-9.

4/ «The Hardware Bible»

[Winn L. Rosch] – Ed. QUE /McMillan computer Publishing – ISBN: 0-7897-1743-3.

@ web::

1-http://css.csail.mit.edu/6.858/2013/readings/
```

2-http://www.ustudy.in/node/

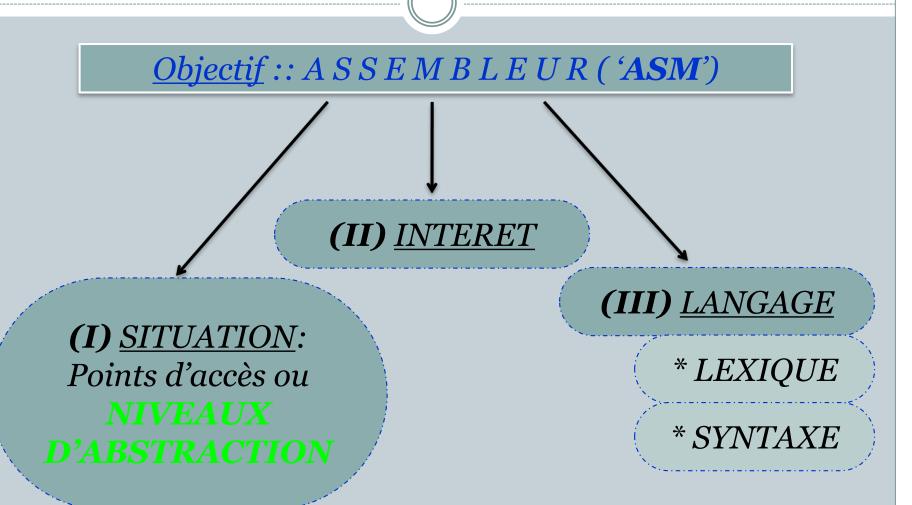
#### ARCHITECTURE/ORDINATEURS

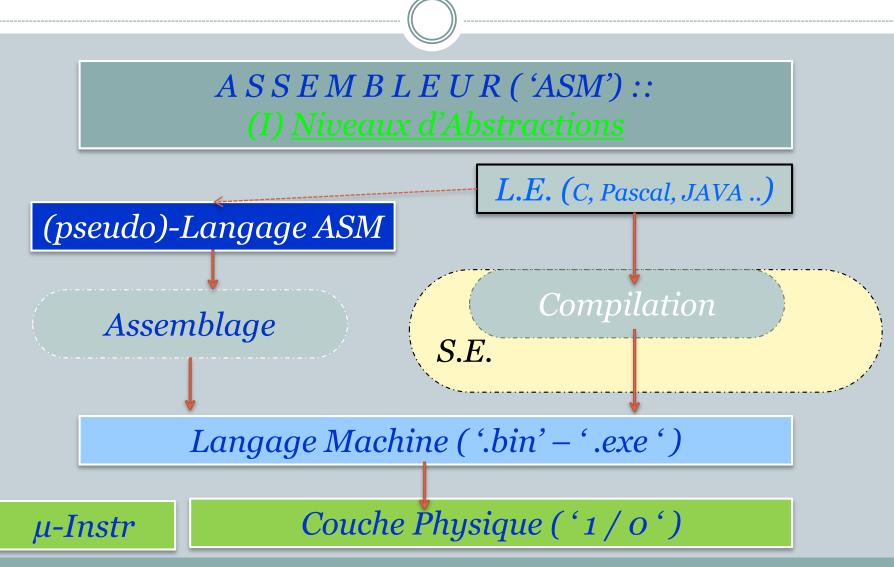
#### (I) <u>RESSOURCES & MECANISMES</u>

- Schémas fonctionnels, modèle V. Neumann, ..
- Registres, adressage, « data & @ » bus, ..
- Concepts avancés : pipeline, fetch, segmentation, Pagination ..

#### (II) <u>EXPLOITATION(RESSOURCES)</u> $\rightarrow$ <u>ASM</u>

- Autres mécanismes & Programmation en ASM, ..
- Outils standards (traitements <u>conditionnels</u>, <u>itératifs</u> ..
- Héritage des outils systèmes : interruptions ..
- Alternative « RISC »





# *ASSEMBLEUR ('ASM') ::*(II) <u>Intérêt</u>

- 1)- Accès aux ressources matérielles 'Low Level'
- 2)- Programmation optimale, orientée 'hardware' & code minimal
- 3)- Exploitation optimale des ressources matérielles
- Domaines d'applications divers : 'systèmes embarqués', 'pilotage process <u>non-standard</u>', etc ..
- Accès à la bibliothèque 'système' : Fonctions DOS & BIOS .. (récupère toute la puissance du SE)

ASSEMBLEUR ('ASM') ::
(III) <u>Le Langage</u>

Dualité

LANGAGE de programmation

ENVIRONNEMENT (programmation)

ASSEMBLEUR ('ASM') ::
(III) <u>Le Langage</u>

<u>LANGAGE</u>

Lexique ::
J.I. orienté 'CPU'

**ENVIRONNEMENT** 

EDI. // 'SE'

ASSEMBLEUR ('ASM') ::
(III) <u>Le Langage</u>

<u>LANGAGE</u>

Lexique ::
J.I. orienté 'CPU'

'Chaque CPU a son propre JI' **ENVIRONNEMENT** 

EDI. // 'SE'

'Exécution orientée S.E.'

ASSEMBLEUR ('ASM') ::
(III) <u>Le Langage</u>

#### **LANGAGE**

Lexique :: J.I. orienté 'CPU'

'Chaque CPU a son propre JI' **ENVIRONNEMENT** 

EDI. // 'SE'

'Exécution orientée S.E.'

N.B.: L'IDE dépend du S.E. mais le J.I. ne dépend que du 'CPU'

## ASSEMBLEUR('ASM'):: (III) Le Langage

#### *LANGAGE*

J.I. orienté 'CPU'

'80286' :: Jeu de 120 instructions :: Transfert, Arithmétique, Logique, Décalage, Condition, ...

#### **ENVIRONNEMENT**

EDI. // 'SE'

DOS /
WINDOWS/UNIX ::
DEBUG, TASM, MASM,
NASM, FASM,
Chrome, Easy code, ...

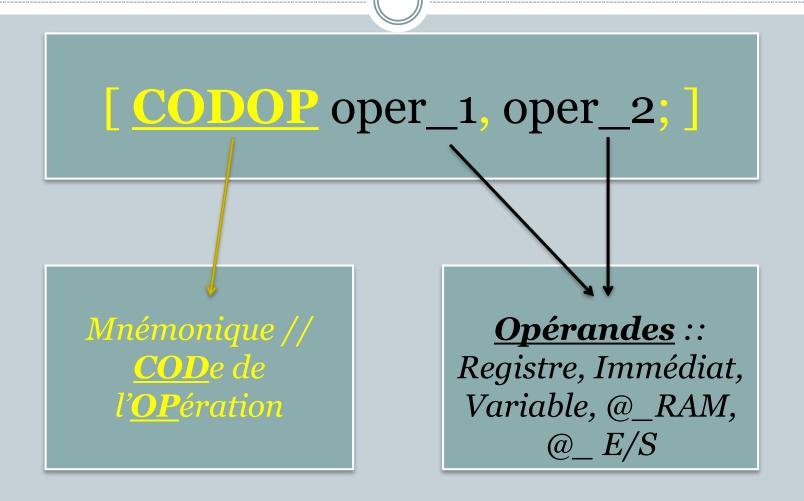
ASSEMBLEUR('ASM'):: (III) Le Langage

<u>LANGAGE</u>

<u>SYNTAXE</u> orientée 'Ressources\_CPU'

LANGAGE // SYNTAXE ::
'Modèle générique d'1 instruction quelconque'

[ CODOP oper\_1, oper\_2; ]





(Destination & 'Source') (Source)

Mnémonique //
CODe de
l'OPération

#### **Opérandes** ::

Oper\_1: '<u>Destination</u>' & '<u>Source</u>'

ANENTE Oper\_2: 'Source'

Règle PERMANENTE

*MODELE*...
[CODOP oper\_1, oper\_2;]

#### i <u>illustration:</u> "Type d'opérandes "

```
CLC i"Mnémonique" / 'sans op'NOP i"Mnémonique" / 'sans op'INC AX i"Mnémonique" + '1 opérande'ADD AX i"Mnémonique" + '2 ops (Reg, Im)'ADD AX iBX i"Mnémonique" + '2 ops (Reg, Reg)'
```

#### EXEMPLE/ILLUSTRATION...

Instructions élémentaires::
(1) Transfert
(2) Arithmétique
(3) Logique

#### i exemple l= instructions & typologie

```
(1)
MOV AX 10 ;
                     (2)
SHL AX, 2;
                     (3)
MOV BX AX 5
                     (4)
ADD AX 2 3
                     (5)
SUB AX, OOLlb;
                     (6)
MUL AX, DAh;
                     (7)
AND AX DOFFh ;
                     (8)
OR AX, DOBIH ;
```

#### i exemple l= instructions & typologie

```
MOV AX 10 i
SHL AX 2 i
MOV BX AX i
ADD AX 2 i
SUB AX 00116 i
MUL AX 0Ah i
AND AX 0061h i
```

- (1) TRANSFERT
- (2) DECALAGE / ARITHM.
- (3) TRANSFERT
- (4) ARITHMETIQUE
- (5) ARITHMETIQUE
- (6) ARITHMETIQUE
- (7) LOGIQUE
- (8) LOGIQUE

```
i exemple 2= manipulation Reg d'adressage
```

```
      MOV AX 1 10 1
      (1)

      MOV SI 1 AX 1
      (2)

      MOV BX 1 AX 1
      (3)

      MOV DI 1 SI+100 1
      (4)

      ADD AX 1 EBX 1 1
      (5)

      MOV EDI 1 ESI+1001 (6)
```

# i exemple 2= imanipulation Reg id'adressage

```
MOV AX, 10;
                     (1)
                     (2)
MOV
   F XA rIZ
                     (3)
MOV
   BX AX ;
   DI 3 SI+100 ;
                     (4)
MOV
                     (5)
ADD AX BX
   EDII- ESII+100;
                     (6)
MOV
```

#### **ACTION**

$$AX \leftarrow 10$$
  
 $SI \leftarrow AX$   
 $BX \leftarrow AX$   
 $DI \leftarrow SI+100$   
 $AX \leftarrow [BX]$   
 $@(DI) \leftarrow [SI]+100$ 

#### RESULTS

AX = 10

#### A RETENIR . . .

≠ entre langage ASM, Environnement (EDI) de ASM et fonction assemblage#compil

#### **ASM** vs **LE** (Langage Evolué):

- Taille\_Instruction (proche de la  $\mu$ -Instr ) =>  $\underline{\text{Décodage}}$  (donc  $\underline{\text{exécution}}$ ) rapide
- Autonomie (*Instr\_ASM*) vis-à-vis du SE

Chaque CPU a son J.I./ASM (qui ne dépend aucunement du SE)

Structure GENERIQUE d'une instruction ASM s/s INTEL:

« CODOP oper 1, oper 2;»

et propriétés respectives de Oper\_1 & Oper\_2.

Classement Instr\_ASM par Taille: implicite, mono-opérande, opérande-double.

Classement Instr\_ASM par <u>TYPE</u>: transfert, arithmétique, logique, décalage, branchement (avec/sans condition)...