

# 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/>

# 6)- Introduction à l'ASM 'x86'



## ARCHITECTURE/ORDINATEURS

### (I) RESSOURCES & MECANISMES

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

### (II) EXPLOITATION(RESSOURCES) → ASM

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

## 6)- Introduction à l'ASM 'x86'

Objectif :: *ASSEMBLEUR* ('ASM')

(II) INTERET

(III) LANGAGE

\* *LEXIQUE*

\* *SYNTAXE*

(I) SITUATION:  
Points d'accès ou  
**NIVEAUX**  
**D'ABSTRACTION**

## 6)- Introduction à l'ASM 'x86'

*ASSEMBLEUR ('ASM') ::  
(I) Niveaux d'Abstractions*

*L.E. (C, Pascal, JAVA ..)*

*(pseudo)-Langage ASM*

*Assemblage*

*Compilation*

*S.E.*

*Langage Machine ( '.bin' – '.exe ' )*

*μ-Instr*

*Couche Physique ( ' 1 / 0 ' )*

## 6)- Introduction à l'ASM 'x86'



### *ASSEMBLEUR ('ASM') :: (II) Intérêt*

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

## 6)- Introduction à l'ASM 'x86'



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

*Dualité*

*LANGAGE de  
programmation*

*ENVIRONNEMENT  
( programmation )*

## 6)- Introduction à l'ASM 'x86'



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



*LANGAGE*

*Lexique ::  
J.I. orienté 'CPU'*



*ENVIRONNEMENT*

*EDI. // 'SE'*

## 6)- Introduction à l'ASM 'x86'



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



*LANGAGE*

*Lexique ::  
J.I. orienté 'CPU'*

*'Chaque CPU a son  
propre JI'*



*ENVIRONNEMENT*

*EDI. // 'SE'*

*'Exécution orientée S.E.'*



## 6)- Introduction à l'ASM 'x86'

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

*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'*

## 6)- Introduction à l'ASM 'x86'



### *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, ...*

## 6)- Introduction à l'ASM 'x86'



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



*LANGAGE*

*SYNTAXE orientée*  
*'Ressources\_CPU'*

## 6)- Introduction à l'ASM 'x86'



***LANGUAGE // SYNTAXE ::***  
*‘Modèle générique d’1 instruction quelconque’*

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

## 6)- Introduction à l'ASM 'x86'

[ CODOP oper\_1, oper\_2; ]

*Mnémonique //*  
*CODE de*  
*l'OPération*

*Opérandes ::*  
*Registre, Immédiat,*  
*Variable, @\_RAM,*  
*@\_ E/S*

## 6)- Introduction à l'ASM 'x86'

[ **CODOP** oper\_1, oper\_2; ]  
(Destination & 'Source') (Source)

Mnémonique //  
CODE de  
l'OPération

**Opérandes** ::  
Registre, Immédiat,  
Variable, @\_RAM,  
@\_E/S

Règle PERMANENTE

Oper\_1 : 'Destination' & 'Source'  
Oper\_2 : 'Source'

## 6)- Introduction à l'ASM 'x86'



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

‡ illustration: “ Type d'opérandes ”

CLC ‡	“Mnémonique” / <u>‘sans op’</u>
NOP ‡	“Mnémonique” / <u>‘sans op’</u>
INC AX ‡	“Mnémonique” + <u>‘1 opérande’</u>
ADD AX, 2 ‡	“Mnémonique” + <u>‘2 ops (Reg, Im)’</u>
ADD AX, BX ‡	“Mnémonique” + <u>‘2 ops (Reg, Reg)’</u>

## 6)- Introduction à l'ASM 'x86'



*EXEMPLE/ILLUSTRATION...*

*Instructions élémentaires::*

- (1) Transfert*
- (2) Arithmétique*
- (3) Logique*



## 6)- Introduction à l'ASM 'x86'



; exemple 1= instructions & typologie

MOV AX, 10 ; (1)

SHL AX, 2 ; (2)

MOV BX, AX ; (3)

ADD AX, 2 ; (4)

SUB AX, 0011b ; (5)

MUL AX, 0Ah ; (6)

AND AX, 00FFh ; (7)

OR AX, 0081h ; (8)

## 6)- Introduction à l'ASM 'x86'



; exemple 1= instructions & typologie

<b>MOV</b> AX, 10 ;	(1) TRANSFERT
<b>SHL</b> AX, 2 ;	(2) DECALAGE / ARITHM.
<b>MOV</b> BX, AX ;	(3) TRANSFERT
<b>ADD</b> AX, 2 ;	(4) ARITHMETIQUE
<b>SUB</b> AX, 0011b ;	(5) ARITHMETIQUE
<b>MUL</b> AX, 0Ah ;	(6) ARITHMETIQUE
<b>AND</b> AX, 00FFh ;	(7) LOGIQUE
<b>OR</b> AX, 0081h ;	(8) LOGIQUE

## 6)- Introduction à l'ASM 'x86'



; exemple 2= manipulation Reg d'adressage

MOV AX, 10 ; (1)

MOV SI, AX ; (2)

MOV BX, AX ; (3)

MOV DI, SI+100 ; (4)

ADD AX, [BX] ; (5)

MOV [DI], [SI+100] ; (6)

## 6)- Introduction à l'ASM 'x86'

; exemple 2=  
; manipulation Reg  
; d'adresse

MOV AX, 10 ; (1)

MOV SI, AX ; (2)

MOV BX, AX ; (3)

MOV DI, SI+100 ; (4)

ADD AX, [BX] ; (5)

MOV [DI], [SI]+100 ; (6)

ACTION

$AX \leftarrow 10$

$SI \leftarrow AX$

$BX \leftarrow AX$

$DI \leftarrow SI+100$

$AX \leftarrow [BX]$

$@[DI] \leftarrow [SI]+100$

RESULTS

$AX = 10$

$SI = 10$

$BX = 10$

$DI = 110$

$AX = [ @ = 10 ]$

$@[DI] \leftarrow [ @ = 110 ]$   
 $= [ @ = 10 ] + 100$

## 6)- Introduction à l'ASM 'x86'



### 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 ) => Décodage (donc exécution) rapide
- Autonomie (*Instr\_ASM*) vis-à-vis du SE

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

Structure GÉNÉRIQUE 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 TYPE: *transfert, arithmétique, logique, décalage, branchement (avec/sans condition)*...