Université de Bouira
Faculté des Sciences et Sciences Appliquées
Département d'Informatique
Semestre 3 (2ème année)

CHAPITRE 3

Etude de cas : Processeur 80x86

Par : Dr. A. ABBAS

Disponible sur :

https://sites.google.com/a/esi.dz/a-abbas

Objectifs du Chapitre

- Présenter le Processeur 8086 de Intel
- Étudier son jeux d'instruction.
- Apprendre à programmer en assembleur pour comprendre le fonctionnement μP 8086 .

Plan

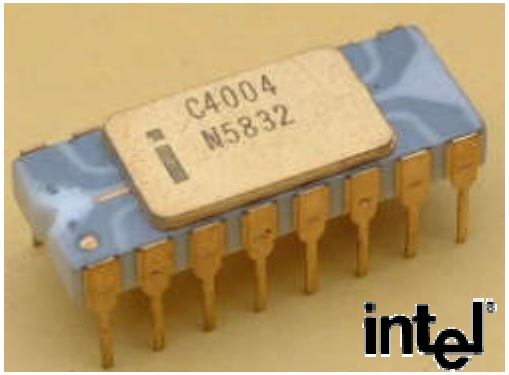
- Introduction
- Architecture générale du µP 8086
- Jeu d'instructions du µP 8086
- Programmation en assembleur

Introduction : Un peu d'histoire

Qui a inventé le microprocesseur ?

quand ?

En 1971



Silicon Gate MOS **4004**

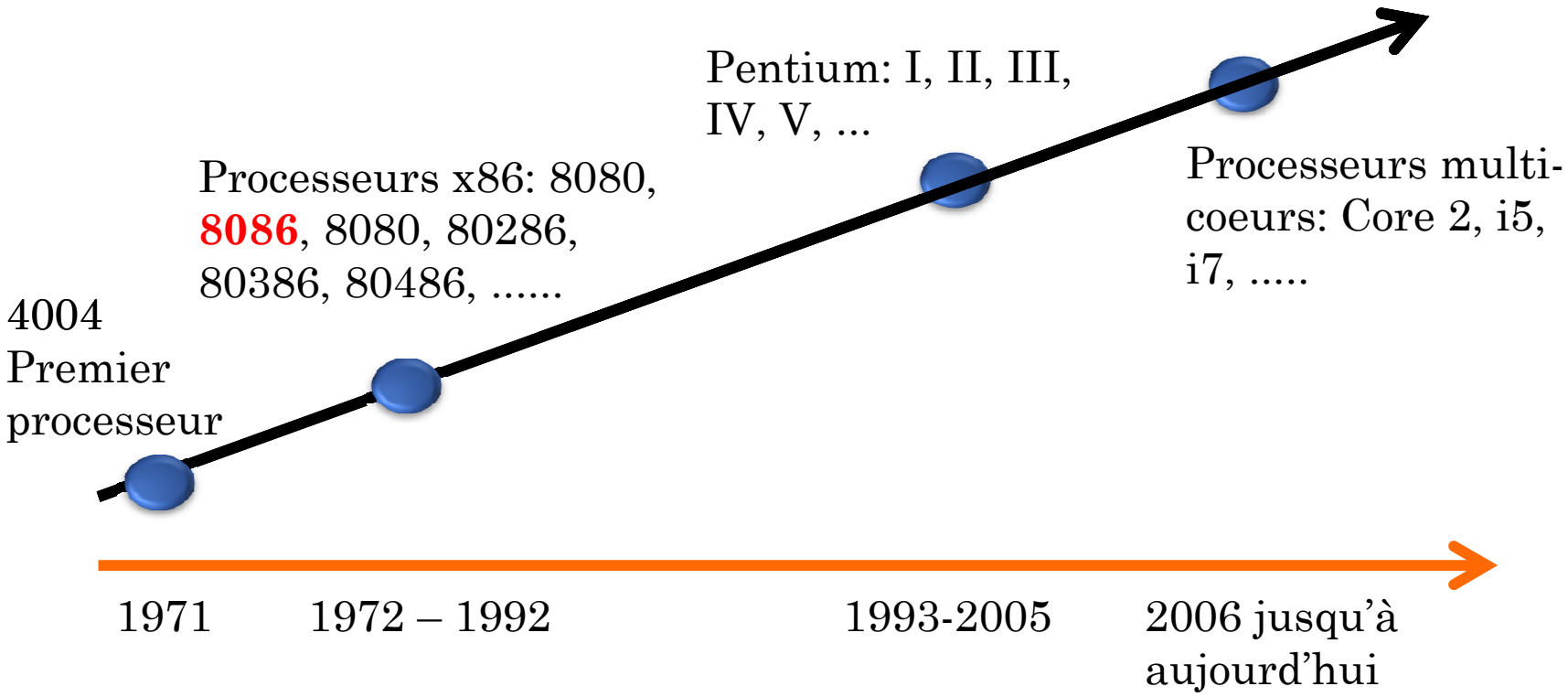
SINGLE CHIP 4-BIT P-CHANNEL MICROPROCESSOR

- 10.8 Microsecond Instruction Cycle
- CPU Directly Compatible With MCS-4 ROMs and RAMs
- Easy Expansion—One CPU can Directly Drive up to 32,768 Bits of ROM and up to 5120 Bits of RAM
- 4-Bit Parallel CPU With 46 Instructions
- Instruction Set Includes Conditional Branching, Jump to Subroutine and Indirect Fetching
- Binary and Decimal Arithmetic Modes

- Composant micro programmé
- 4 bits
- 2300 transistors
- Horloge 100 Khz
- Développé par INTEL pour BUSICOM

Introduction : Un peu d'histoire

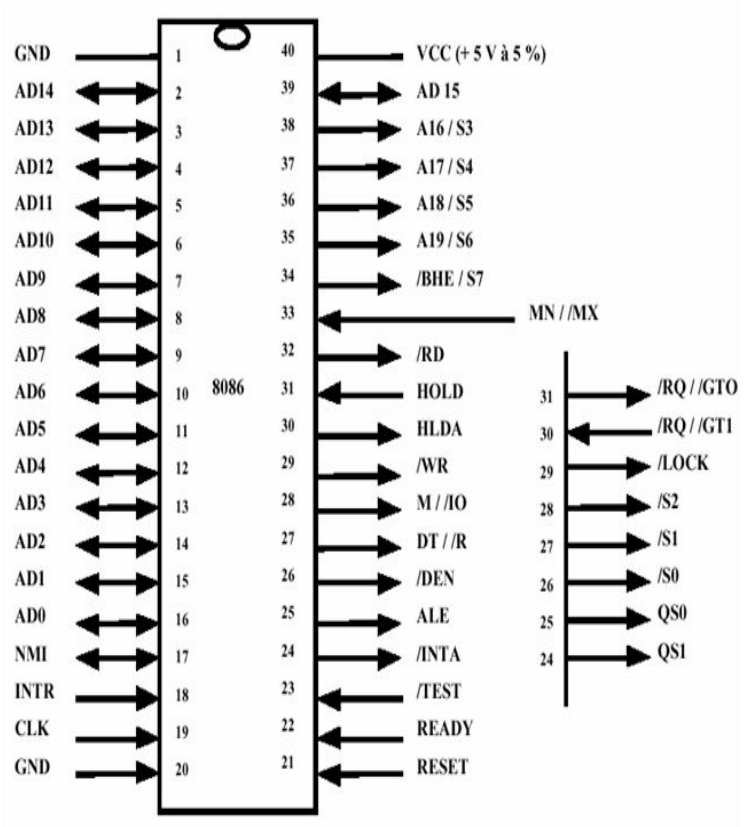
Évolution des processeurs Intel



Le processeur 8086 d'Intel est à la base des processeurs Pentium actuels et ils sont avec le 8086

Introduction: Le processeur 8086

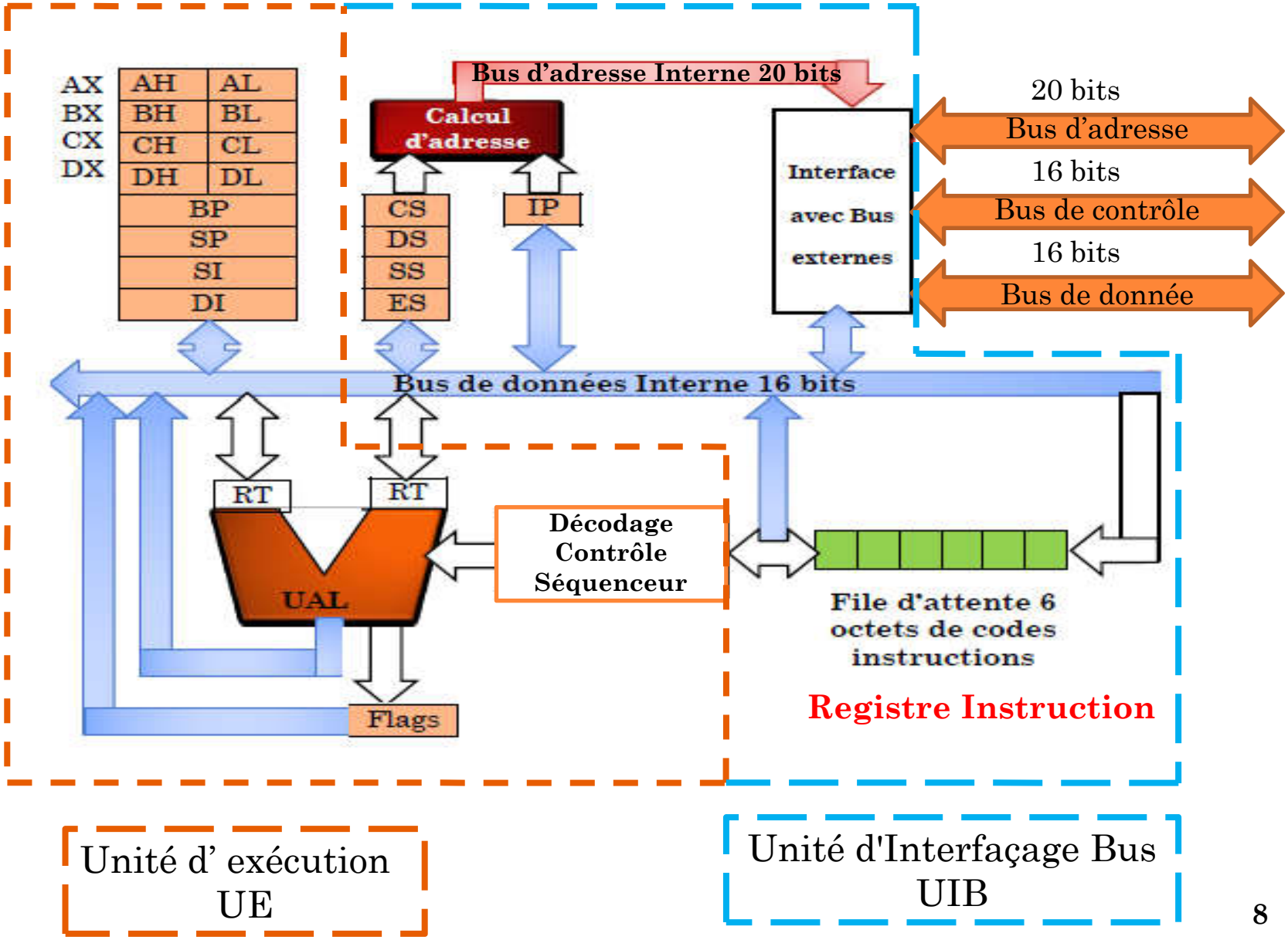
- Le 8086 (développé en 1978) est le premier microprocesseur de type x86
- Le 8086 est le premier processeur 16 bits fabriqué par Intel.
- Il se présente sous forme d'un boîtier de **40 broches** alimenté par une alimentation unique de 5V.



PARTIE 1:

Architecture Générale du µP 8086

Architecture Générale du µP 8086



Architecture interne du 8086 :

Le 8086 comporte deux unités fonctionnelles qui peuvent travailler en parallèle : unité d' exécution (**UE**) et Unité d'Interfaçage Bus (**UIB**)

UE comporte:

- **UAL :** peut être charger par **2 registre temporaires** (TR) et effectuer des opérations sur des mots de 8 bits ou de 16 bits.
- **Quatre registres d'intérêt général** (AX, BX, CX et DX),
- **Deux pointeurs de pile** (SP et BP),
- **Deux registres d'index** (SI et DI).

UIB comporte:

- **Quatre registres de segments** (**DS**, **CS**, **SS** et **ES**),
- **Un compteur d'instruction** (**IP** - Instruction Pointer),
- **Une pile** (6 octets) pour stocker les instruction et données en attente.

Note : Tous les registres sont d'une largeur de 16 bits,

Architecture interne du 8086 :

La segmentation de la mémoire

- La taille du bus d'adresse égale à **20 bits ⇒ La mémoire totale adressable égale 2^{20} octets = 1 Mo**

- La taille des registres est **16 bits ⇒ on peut adresser seulement 2^{16} octets = 64 ko.**

- La mémoire est donc fractionnée en **pages de 64 ko appelés segments.**

- Pour adresser une case mémoire donnée, on utilise alors deux registres :

  - Un registre pour adresser le segment, appelé **registre segment: CS, DS, SS, ES**
  - Un registre pour adresser à l'intérieur du segment, appelé **registre offset: IP, SP, BP, SI, DI.**

- Une adresse se présente sous la forme **segment:offset**

Le segment localise le début d'une zone mémoire de 64Ko
L'offset précise l'adresse relative par rapport au début de segment.

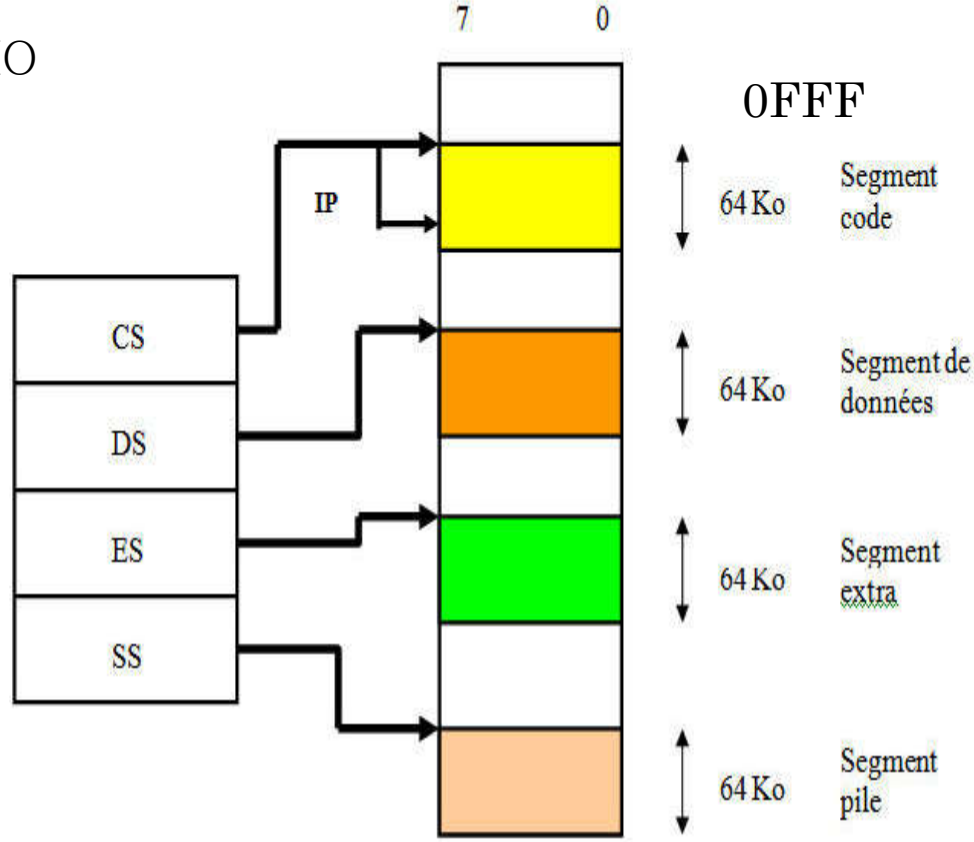
Architecture interne du 8086 :

Gestion de la mémoire :

L'espace mémoire adressable (1 MO = 2^{20} , 20 bits du bus d'adresse)

Le compteur de programme (IP) est de 16 bits donc la possibilité d'adressage est de 2^{16} = 64 Ko

Mémoire est divisé en quatre segment logiques allant jusqu'à 64 KO chacun.



Chaque segment débute à l'endroit spécifié par le registre segment

Le déplacement (offset) à l'intérieur du segment se fait par un registre de décalage qui permet de trouver une information à l'intérieur du segment.

Exemple: la paire de registre CS:IP : pointe sur le code d'une instruction (CS: registre segment et IP: Déplacement)

Architecture interne du 8086 :

Gestion de la mémoire : <u>Adresse physique</u>

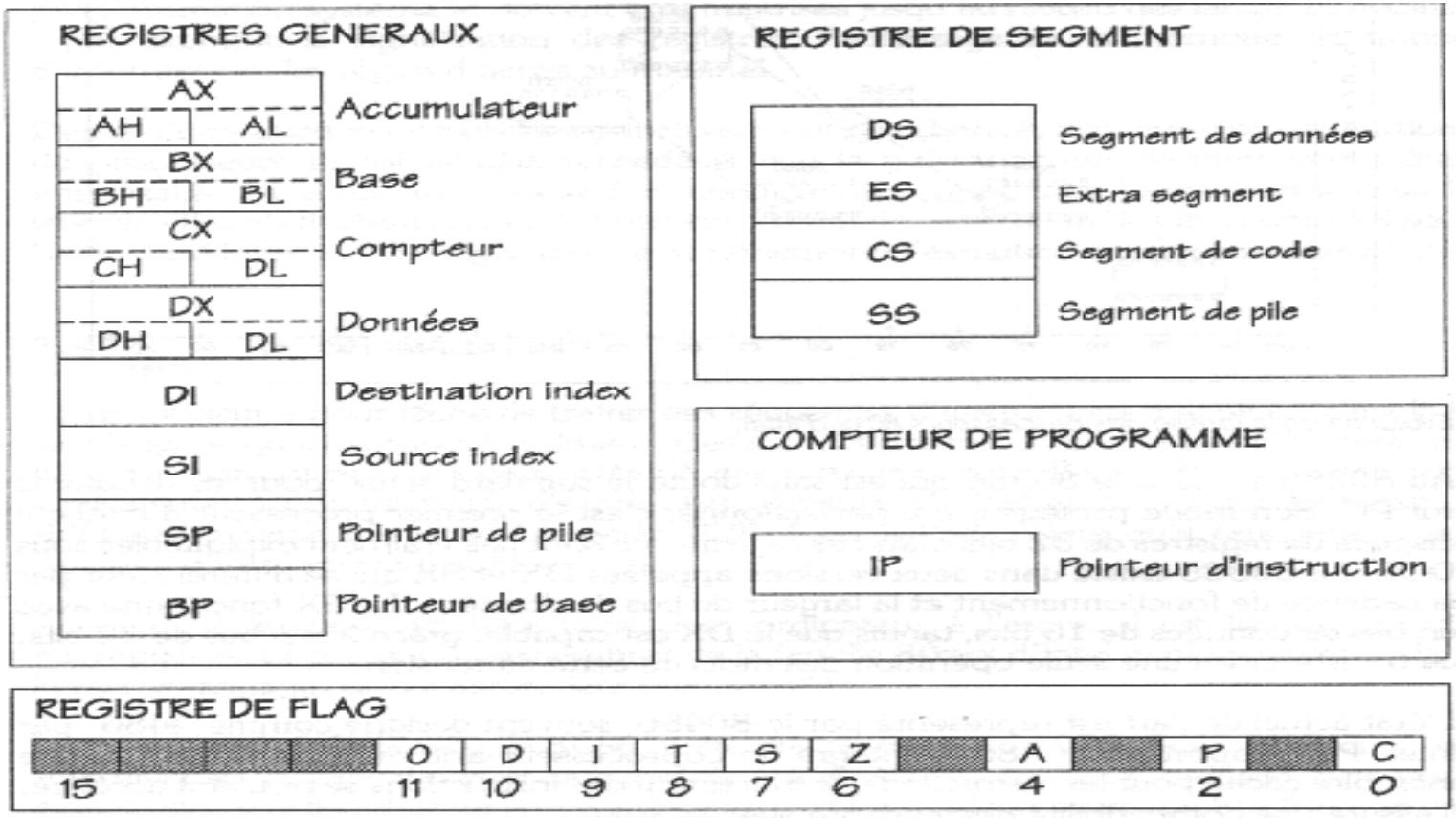
L'adresse de 20 bits est formée par la juxtaposition d'un registre segment (16 bits de poids fort) et d'un déplacement *(offse*t, 16 bits de poids faible).

Le schéma de la figure suivante illustre la formation d'une adresse 20 bits à partir du segment et du déplacement sur 16 bits :

Bits	20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
	Base	0 0 0 0
+	0 0 0 0	Offset
=	Adresse physique	

Architecture interne du 8086 :

Les registres



Architecture interne du 8086 :

Les registres généraux

Registres	Usage général et
AX: Accumulateur	Obligatoire pour la multiplication et la division et ne peut pas servir pour l'adressage
BX : Base	Adressage
CX : Comptage et calcul	Compteur de répétition.
DX : Data	Extension au registre AX pour contenir un nombre 32 bits dans la multiplication et la division 16 bits
SP : Pointeur de Pile	Utilisé **uniquement** pour l'accès à la pile. Pointe sur la tête de la pile
BP : Pointeur de Base	Adressage comme registre de base (BX)
SI : Registre d'index (source)	Adressage comme registre d'index de l'opérande source.
DI : Registre d'index (destination)	Adressage comme registre d'index de l'opérande destination

Architecture interne du 8086 :

Les registres de segment

Registres	Usage général et
CS : Code Segment	▪ Définit le début de la mémoire programme dans laquelle sont stockées les instructions du programme. ▪ Les adresses des différentes instructions du programme sont relatives à CS
DS : Data Segment	▪ Définit le début de la mémoire de données dans laquelle sont stockées toutes les données traitées par le programme.
SS : Stack Segment	▪ Définit le début de la pile. ▪ SP permet de gérer l'empilement et le dépilement.
ES : Extra Segment	▪ Définit le début d'un segment auxiliaire pour données

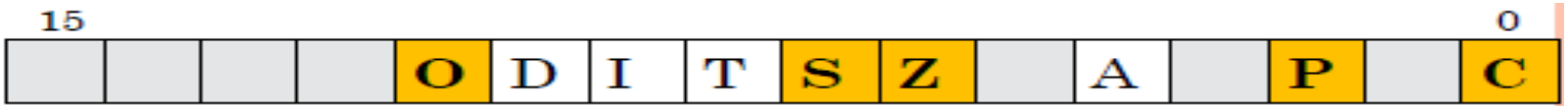
Le pointeur d'instruction

Le pointeur d'instruction (IP), appelé aussi Compteur Ordinal (C.O.)

permet de pointer TOUJOURS le premier octet de l'instruction suivante.

Architecture interne du 8086 :

Registre d'état (Flags)

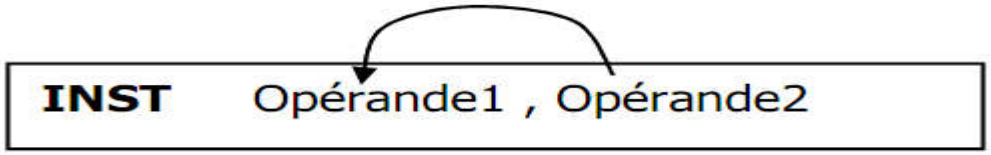


- **C (Carry) :** indique qu'il y a une retenue du résultat à 8 bits ou 16 bits.
- **P (Parité) :** indique que le nombre de 1 est un nombre pair.
- **Z (Zéro) :** indique que le résultat est nul.
- **S (Signe) :** indique le signe du résultat
- **O (Overflow):** indique un dépassement de capacité

Architecture interne du 8086 :

Format d'instruction

- La structure la plus générale d'une instruction est la suivante :



L'opération est réalisée entre les 2 opérandes et le résultat est toujours récupéré dans l'opérande de gauche.

Exemple : CMP, ADD, MOV, LEA, XCHG, AND

- Il y a aussi des instructions qui agissent sur un seul opérande

INST Opérande

Exemple : INC, DEC, NEG, NOT ....;

- Il y a aussi des instructions sans opérande

INST *Exemple : NOP, STI, CLI, PUSHF, CBW...;*

- Les opérandes peuvent être des registres, des constantes ou le contenu de cases mémoire.

Architecture interne du 8086 :

Modes d'adressage

1- Adressage registre

L'opération se fait sur un ou 2 registres

INST R , R

INST R

Exemples :

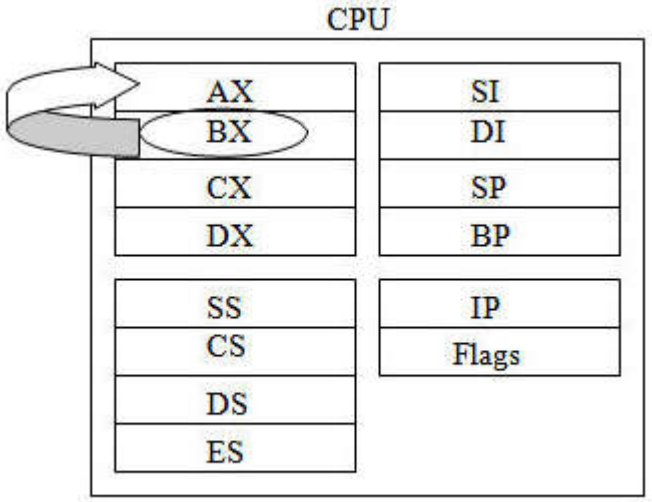
INC AX : incrémenter le registre AX (AX++)

MOV AX, BX : Copier le contenu de BX

dans AX (AX←BX); opérandes 16 bit

ADD CH, DL : Copier le contenu de DL

dans CH (CH←DL) ; opérandes 8 bits



Architecture interne du 8086 :

Modes d'adressage

2- Adressage Immédiat

L'opérande est une constante (valeur) qui fait partie de l'instruction :

INST R , IM

Exemples :

MOV AX, 243 : charger le registre AX par le nombre décimal 243 (AX←243)

MOV AL, 'A' : Charger le registre AL par le code ASCII du caractère 'A' (65)

MOV AX,'AB' : Charger AH par le code ASCII du caractère 'A' (65) et AL par le code ASCII du caractère 'B' (66)

***MOV AX*, 0A1EBH** ; *AX*:= valeur hexa A1EB

Architecture interne du 8086 :

Modes d'adressage

3- Adressage direct

Un des deux opérandes se trouve en mémoire. L'adresse de la case mémoire est précisé directement dans l'instruction.

INST R , [adr] INST [adr] , R INST taille [adr] , IM

L'adresse doit être placée entre **[Rseg:Roffset].**

Si le segment (Rseg) n'est pas précisé, DS (début de la mémoire de données) est pris par défaut comme Rseg .

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

MOV AX, [SS:243] : Copier le contenu de la mémoire SS:243 dans AX

MOV BYTE [adresse],4Ah, On écrit 4A dans la position adresse

MOV WORD [adresse],4Ah ; On écrit 004A, 4A → adresse, et 00 → adresse+1

Architecture interne du 8086 :

Modes d'adressage

4- Adressage Indirect

Un des deux opérandes se trouve en mémoire. L'adresse se trouve dans l'un de ces 4 registres BX, BP, SI ou DI.

INST R , [Rseg : Roffset]

INST [Rseg : Roffset] , R

INST taille [Rseg : Roffset] , im

Si Rseg n'est pas spécifié, le segment par défaut sera utilisé comme suit.

Registre Roffset	BX	BP	SI	DI
Segment par défaut	DS	SS	DS	DS

Exemples :

MOV AX, [BX]; Charger AX par le contenu de la mémoire d'adresse DS:BX

Architecture interne du 8086 :

Modes d'adressage

4- Adressage Indirect

L'adressage indirect est divisé en 3 catégories selon le registre d'offset utilisé: l'adressage **Basé**, l'adressage **indexé** et l'adressage **basé indexé**.

Mode Adressage	Basé	Indexé	Basé Indexé
Registres utilisés	BX BP	DI SI	BX, DI BX, SI BP, DI BP, SI

Architecture interne du 8086 :

Modes d'adressage

4- Adressage Indirect

A. Adressage Basé

L'offset se trouve dans l'un des deux registres de base BX ou BP. On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset,

INST R , [Rseg : Rb+dep]

INST [Rseg : Rb+dep] , R

INST taille [Rseg : Rb+dep] , im

Exemples :

MOV AX, [BX] : Charger AX par le contenu de la mémoire d'adresse DS:BX

MOV AX, [BX+5] : Charger AX par le contenu de la mémoire d'adresse DS:BX+5

Architecture interne du 8086 :

Modes d'adressage

4- Adressage Indirect

B. Adressage Indexé

L'offset se trouve dans l'un des deux registres d'index SI ou DI. On peut préciser un déplacement qui sera ajouté au contenu de RI pour déterminer l'offset.

INST R , [Rseg : RI+DEP]

INST [Rseg : RI+DEP] , R

INST taille [Rseg : RI+DEP] , im

Exemples :

MOV AX, [SI]; Charger AX par le contenu de la mémoire d'adresse DS:SI

MOV AX, [SI-500]; Charger AX par la mémoire d'adresse DS:SI-500

Architecture interne du 8086 :

Modes d'adressage

4- Adressage Indirect

A. Adressage Basé Indexé

L'offset de l'adresse de l'opérande est la somme d'un registre de base, d'un registre d'index et d'un déplacement optionnel. Si Rseg n'est pas spécifié, le segment par défaut du registre de base est utilisé :

INST R , [Rseg : Rb+Ri+dep]

INST [Rseg : Rb+Ri+dep] , R

INST taille [Rseg : Rb+Ri+dep] , im

Exemples :

MOV AX, [BX+DI+5]; AX est chargé par la mémoire d'adresse DS:BX+DI+5

MOV AX, [SI-500]; Charger AX par la mémoire d'adresse DS:SI-500

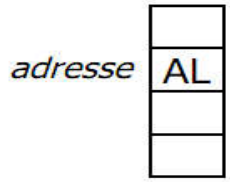
Architecture interne du 8086 :

Taille des échanges avec la mémoire

- La mémoire est organisée en octets.

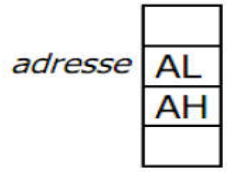
- Quand on fait une instruction entre un registre et une donnée qui se trouve en mémoire, c'est le registre qui détermine la taille de l'opération:
  - Si le registre est sur 8 bits, l'opération se fera avec une seule case mémoire.

MOV [adresse], AL donne →



  - Si le registre est sur 2 octets, l'opération se fera avec deux cases mémoires

MOV [adresse], AX donne →



PARTIE 2:

Jeu d'instructions du μP 8086

Jeu d'instructions du µP 8086

Plusieurs types d'instructions, notamment:

- Instructions de Transfert (MOV, XCHG, ...)
- Instructions Arithmétiques (ADD, SUB, MUL, DIV, ...)
- Instructions Logiques (NOT, AND, OR, XOR, ...)
- Instructions de Décalage (SHR, SHL, ROL, ROR, ...)
- Instruction de Branchement (JMP, Jxx, LOOP....)

A suivre...

Jeu d'instructions du µP 8086

MOV

MOV Od,Os: Copie l'opérande Source (Os) dans l'opérande Destination (Od) *[Od ←OS]*

MOV R1 , R2	R1←R2
MOV R , [adr]	R←[adr]
MOV [adr] , R	[adr]←R
MOV R , im	R←im
MOV taille [adr] , im	copier une constante dans une case mémoire *taille = BYTE ou WORD*

✗ **MOV [adr], [adr]**
MOV R_{seg}, R_{seg}

Jeu d'instructions du μP 8086

Instructions arithmétiques

- Le 8086 permet d'effectuer les quatre opérations arithmétiques de base: l'**addition**, la **soustraction**, la **multiplication** et la **division**.

- Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signés ou non signés.

Jeu d'instructions du µP 8086

ADD

ADD Od,Os: additionne l'opérande source et l'opérande destination et met le résultat dans l'opérande destination *[Od←Od+Os]*.

ADD R1 , R2	R1 ← R1 + R2
ADD R , [adr]	R1 ← R + [adr]
ADD [adr] , R	[adr] ← [adr] + R
ADD R , im	R ← R + im
ADD taille [adr] , im	[adr] ← [adr] + im

ADD [adr], [adr]

Jeu d'instructions du μP 8086

Exemples :

add ah, [1100H] : ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH (**adressage direct) ;**

add ah,[bx] : ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH (**adressage base) ;**

add byte [1200H], 05H : ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (**adressage immediat).**

Jeu d'instructions du μP 8086

Soustraction :

SUB operande1, operande2

L'opération effectuée est :

opérande1←opérande1 - opérande2.

Exemples :

mov ax, 39h

sub ax, 18h

ax←ax-18h

Jeu d'instructions du µP 8086

Multiplication :

MUL operande,

où opérande est un registre ou une case mémoire.

AX←AL x Op8 DX:AX←AX x Op16

Cette instruction effectue la multiplication du contenu de AL par un opérande sur 1 octet ou du contenu de AX par un opérande sur 2 octets.

Le résultat est placé dans AX si les données à multiplier sont sur 1 octet (résultat sur 16 bits), dans (DX,AX) si elles sont sur 2 octets (résultat sur 32 bits).

Les drapeaux CF et OF sont positionnés si la partie haute du résultat est non nulle. La partie haute est AH pour la multiplication 8 bits et DX pour la multiplication 16 bits

Jeu d'instructions du μP 8086

Exemples :

```
mov al,51
mov bl,32
mul bl
```

AX = 51 × 32

```
mov ax,4253
mov bx,1689
mul bx
```

(DX, AX) = 4253 × 1689

Jeu d'instructions du μP 8086

Division :

DIV operande,

où opérande est un registre ou une case mémoire.

Cette instruction effectue la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX,AX) par un opérande sur 2 octets.

Résultat :

si l'opérande est sur 1 octet, alors AL = quotient et AH =reste ;

si l'opérande est sur 2 octets, alors AX = quotient et DX =reste.

DIV Op8 ; AX/Op8 , Quotient=AL , Reste =AH

DIV Seg16 ; DX:AX/Seg16 , Quotient =AX , Reste=DX

Jeu d'instructions du μP 8086

Exemples :

```
mov ax,35
mov bl,10
div bl
```

AL = 3 (quotient) et AH = 5 (reste)

```
mov dx,0
mov ax,1234
mov bx,10
div bx
```

AX = 123 (quotient) et DX = 4 (reste)

Jeu d'instructions du μP 8086

Autres instructions arithmétiques :

INC Op :	**incrémentation d'une unité ;**	**Op + 1→ Op**
DEC Op:	**décrémentation d'une unité ;**	**Op - 1 → Op**
IMUL :	**multiplication signée ;**	
IDIV :	**division signée.**	

Remarque :

Attention, l'indicateur C n'est pas positionné quand il y a débordement, C'est l'indicateur Z qui permet de détecter le débordement.

Pour incrémenter une case mémoire, il faut préciser la taille :

INC byte [adr] **INC word [adr]**

Jeu d'instructions du μP 8086

Autres instructions arithmétiques :

NEG Op	remplace Op par son négatif (Complément à 2 de Op) :
CMP OD, OS	Compare (soustrait) **OS** et **OD** et positionne les flags selon résultat.
	L'opérande Od n'est pas modifié

Jeu d'instructions du μP 8086

Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont :

- ET;
- OU;
- OU exclusif ;
- complément à 1;
- complément à 2;
- décalages et rotations.

Jeu d'instructions du μP 8086

Les instructions logiques

ET logique : **AND opérande1, opérande2**

L'opération effectuée est :

opérande1 ← opérande1 ET opérande2

Exemple :

mov al, 10010110B

mov bl, 11001101B

and al, bl

→

AL =	1	0	0	1	0	1	1	0
BL =	1	1	0	0	1	1	0	1
AL =	1	0	0	0	0	1	0	0

Jeu d'instructions du µP 8086

Instruction PUSH : PUSH Op

Empiler l'opérande Op (Op doit être un opérande de 16 bits)

- Décrémente SP de 2
- Copie Op dans la mémoire pointée par SP

PUSH R16 ; R16=Registre sur 16 bits

PUSH word [adr]

~~PUSH Val_Immédiate~~ ~~PUSH R8~~

Instruction POP : POP Op

Dépiler dans l'opérande Op (Op doit être un opérande 16 bits)

- Copie les deux cases mémoire pointée par SP dans l'opérande Op
- Incrémente SP de 2

POP R16 **POP word [adr]**

~~**POP R8**~~

Jeu d'instructions du µP 8086

Exemple Illustratif : Instruction PUSH , POP

MOV AX 10Ah

MOV BX 20Bh

MOV CX 30Ch

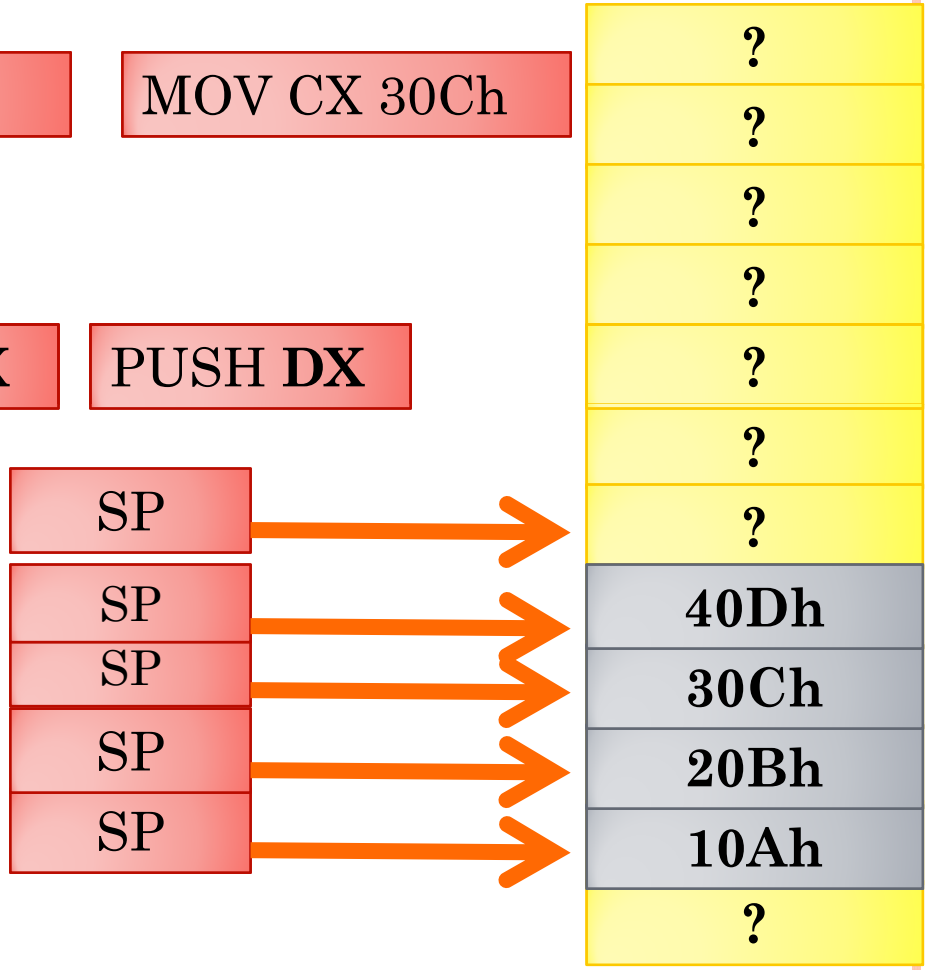
MOV DX 40Dh

PUSH **AX**

PUSH **BX**

PUSH **CX**

PUSH **DX**



Jeu d'instructions du μP 8086

Instruction PUSHA : PUSHA

Empile tous les registres généraux et d'adressage dans l'ordre suivant :

AX, CX, DX, BX, SP, BP, SI, DI

Instruction POPA : POPA

Dépile tous les registres généraux et d'adressage dans l'ordre inverse de PUSHA afin que chaque registre retrouve sa valeur. La valeur dépilée de SP est ignorée.

Remarque :

Les deux instructions PUSHA et POPA ne sont pas reconnues par le 8086. Elles ont été introduites à partir du 80186.

Jeu d'instructions du μP 8086

Instruction XCHG : XCHG OD , OS

Échange l'Opérande Source avec l'Opérande Destination.

Impossible sur segment (entre cases mémoire).

XCHG Reg1 , Reg2

XCHG R , [adr]

XCHG [adr] , Reg

~~XCHG [adr] , [adr]~~

Jeu d'instructions du μP 8086

Instructions de branchement

 Les instructions de branchement (ou saut) permettent de modifier l'ordre d'exécution des instructions du programme en fonction de certaines conditions.

 L'instruction de branchement est une instruction à un opérande « **INST Label ».** Un label (ou une étiquette) est une représentation symbolique d'une instruction en mémoire.

 Le mode d'adressage des instructions de branchement est **immédiat.**

Jeu d'instructions du µP 8086

Instructions de branchement

On distingue 3 types d'instructions:

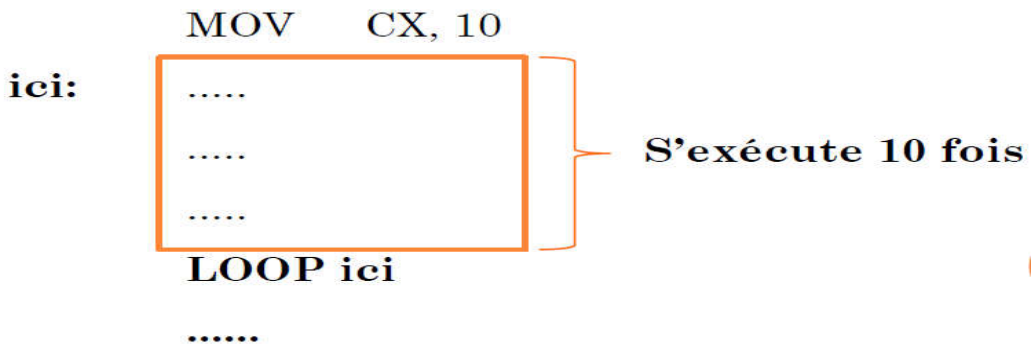
- Contrôle de Boucle : ***LOOP X***
- Branchement inconditionnel: ***JMP***
- Branchement conditionnel: ***Jcondition***

Jeu d'instructions du µP 8086

Instructions de Contrôle de Boucle : *LOOP*

LOOP Label:

fonctionne automatiquement avec le registre CX (compteur). Quant le processeur rencontre une instruction LOOP, il décrémente le registre CX. Si le résultat n'est pas encore nul, il reboucle à la ligne portant l'étiquette label, sinon il continue le programme à la ligne suivante



Jeu d'instructions du µP 8086

Instructions de Contrôle de Boucle : *LOOP*

- **Exemple: Que fait ce programme?**

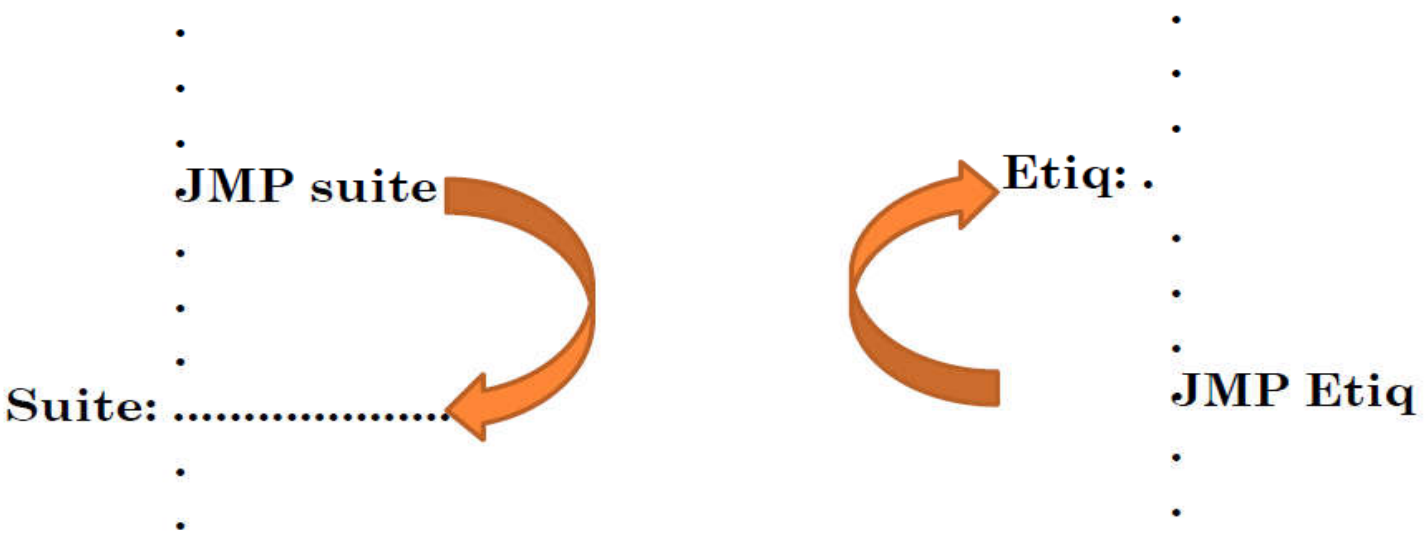
```
        MOV DX, 0
        MOV CX, 5
ETIQ:   MOV BX, CX
        ADD  DL, [BX+1100]
        LOOP ETIQ
```

Addition les 5 valeurs se trouvant à l'adresse 1101-1105

Jeu d'instructions du µP 8086

Instructions de Branchement inconditionnel

JMP label: effectue un saut (jump) vers le label spécifié.



Jeu d'instructions du μP 8086

Instructions de branchement conditionnel

- Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite, sinon l'exécution se poursuit séquentiellement à l'instruction suivante.
- La condition du saut porte sur l'état de l'un (ou plusieurs) des indicateurs d'état (flags) du microprocesseur qui sont positionnés en fonction du résultat de la dernière opération.

Jeu d'instructions du µP 8086

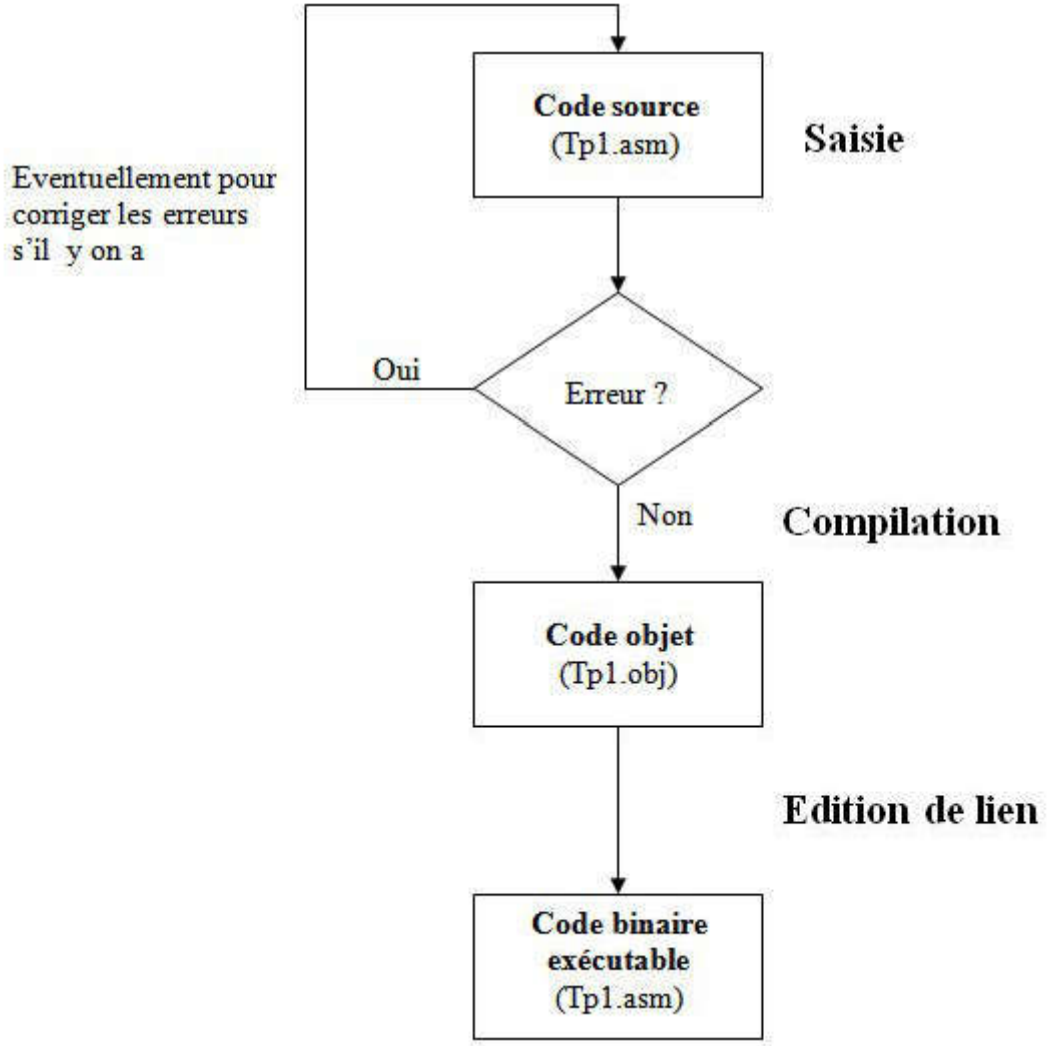
Instructions de branchement conditionnel

Instruction	Nom	Condition
JZ	**Jump if Zero**	Saut si Z = 1
JNZ	**Jump if Not Zero**	Saut si Z = 0
JC	**Jump if Carry**	Saut si C = 1
JNC	**Jump if Not Carry**	Saut si C = 0
JS	**Jump if Sign**	Saut si S = 1
JNS	**Jump if Not Sign**	Saut si S = 0
JO	**Jump if Overflow**	Saut si O = 1
JNO	**Jump if Not Overflow**	Saut si O = 0
JP	**Jump if Parity**	Saut si P = 1
JNP	**Jump if Not Parity**	Saut si P = 0

PROGRAMMATION EN ASSEMBLEUR

Plusieurs logiciels permettent le passage entre ces trois phases présentée comme:

- **MASM** (Microsoft Assembler : avec LINK comme éditeur de lien),
- **TASM** (Turbo assembler : avec TLINK comme éditeur de lien) **et**
- **NASM, etc ...**



PROGRAMMATION EN ASSEMBLEUR

POURQUOI?

- Quelques fois, le code écrit en assembleur peut être plus rapide et plus compact que le code généré par un compilateur.
- L'assembleur permet **l'accès à des fonctionnalités matérielles** du système directement qu'il pourrait être difficile ou impossible à utiliser depuis un langage de plus haut niveau (C, C++, C #, Java,).
- Acquérir une compréhension plus profonde de la façon dont fonctionne un ordinateur (par exemple, comment les compilateurs et les langage de haut niveau fonctionnent)