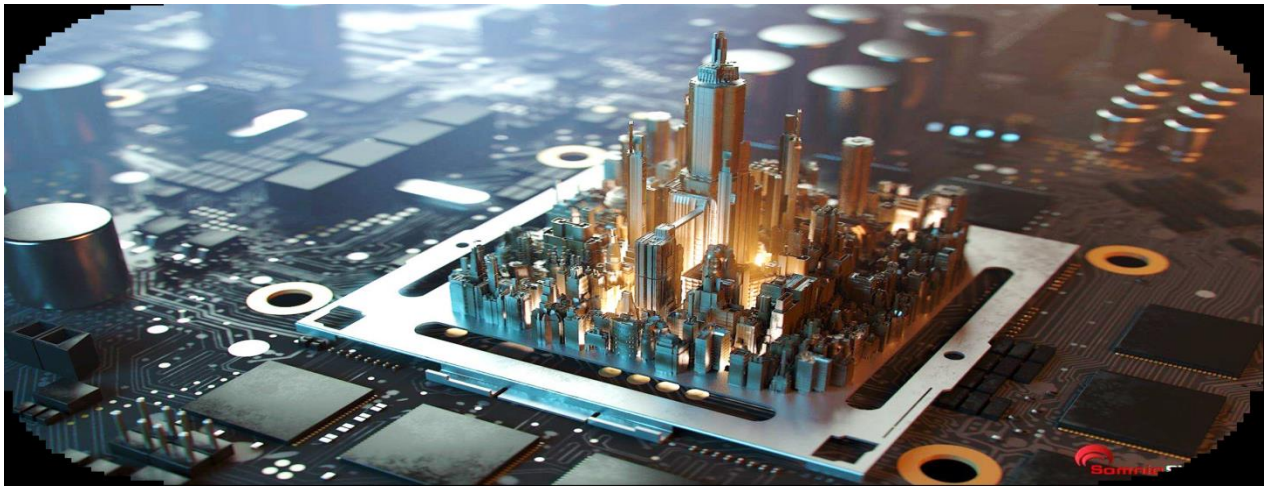
Faculté des sciences exactess
Département d'informatique

Cours 3: Étude de cas « Processeur Intel 8086 »



NIVEAU: INGÉNIEUR I ANNÉE

PRÉSENTÉE PAR: DR. SAAD NARIMANE

2024/2025

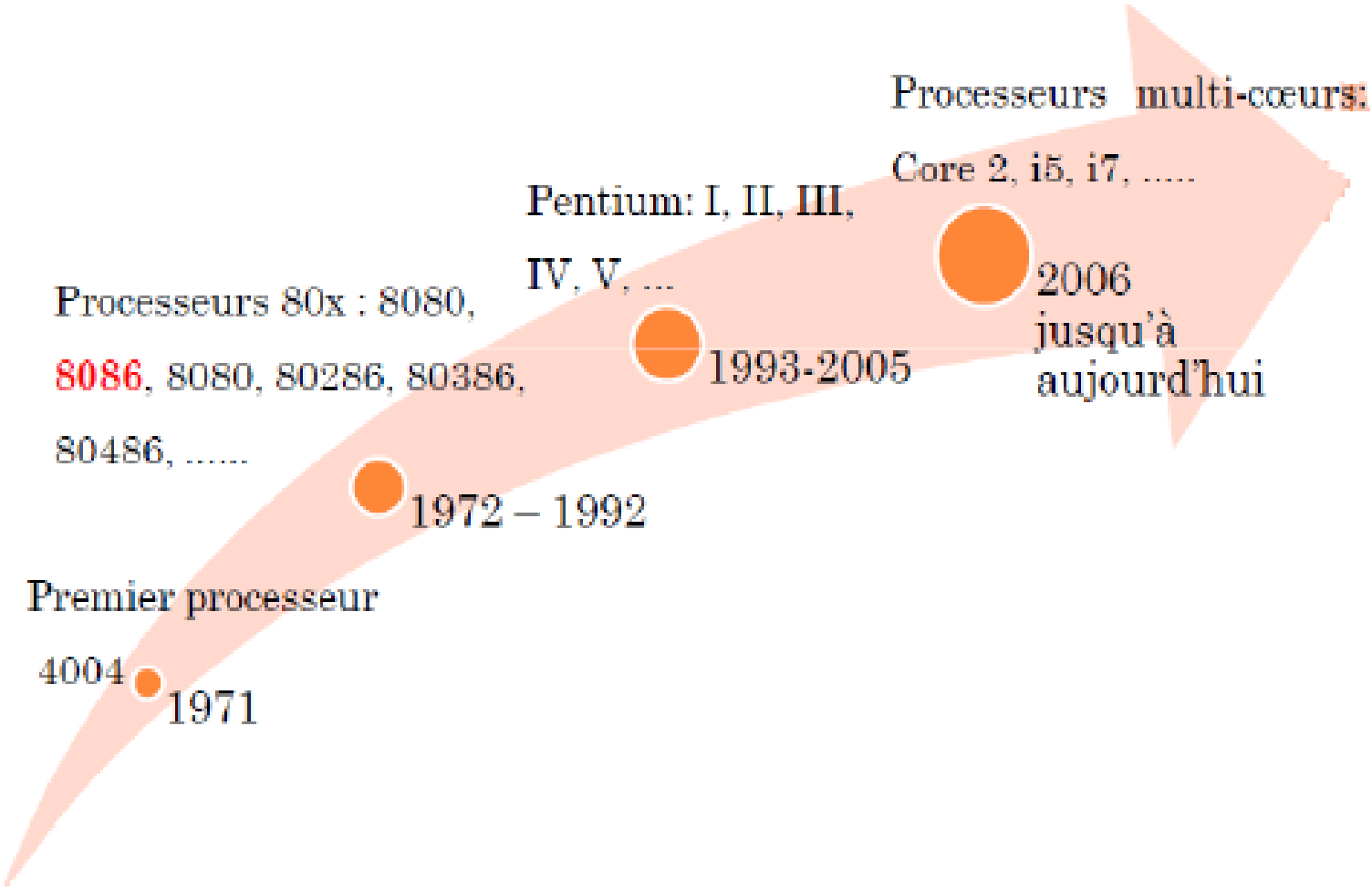
Introduction

- Le 8086 (développé en 1978) d'Intel est à la base des processeurs Pentium,
- Equipé d'un bus de données de 16 bits « traite des données codées sur 16 bits », et un bus d'adresses de 20 bits ,
- 14 registres pour le 8086
- Accéder à une donnée dans un registre, il faut spécifier son nom
- Chaque registre possède un nom qui est une chaine de caractères

Remarque :

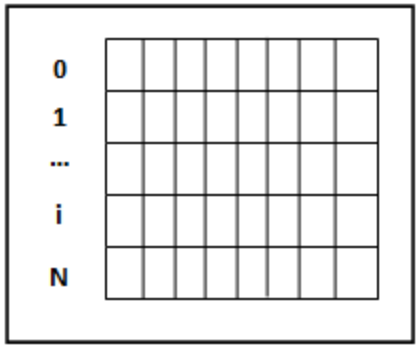
✓**Un bit** est une valeur binaire qui, par convention, peut prendre la valeur 0 ou 1 ;
✓ **Un octet** est une donnée codée sur 8 bits,
✓ **Un mot** est une donnée codée sur 16 bits,
✓ **Un Koctet** est un ensemble de 1024 octets,
✓ **Un Moctet** est un ensemble de 1024 Koctets.

Evolution des Processeurs Intel 8086



I. La gestion de la mémoire « Adressage segmenté »

- La mémoire est une séquence d'octets, numérotés entre 0 et N – 1
- La capacité de la mémoire = N octets. toujours N est une puissance de 2 ($N=2^m$).
- Adresse effective (AE) d'un octet en mémoire = le numéro qui lui est associé



- Pour le 8086, un octet est désigné par un couple (***numéro de segment, déplacement dans le segment***)

- ✓ **Un segment** est un ensemble de **64 Koctets consécutifs.**
- ✓ **Un déplacement (ou offset)** spécifie un octet particulier dans un segment.
- Segment et déplacement sont codés sur **16 bits** et peuvent donc prendre une valeur comprise **entre 0 et 65535**, ou :

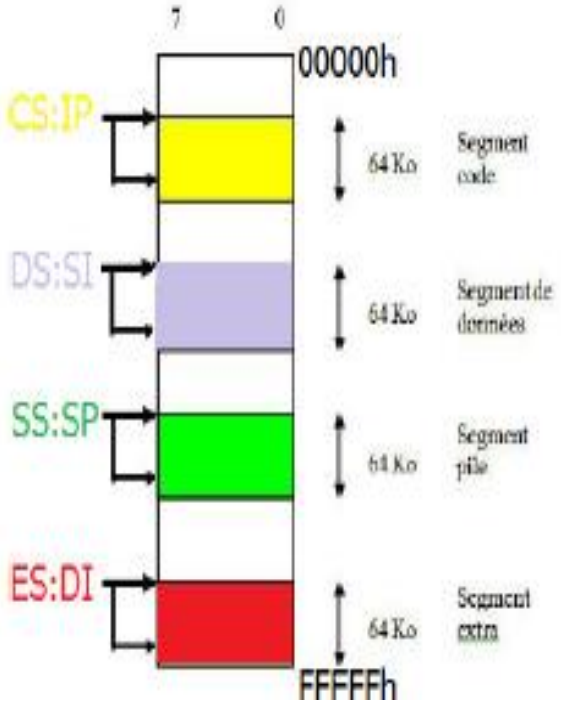
Adresse effective = 16 × segment + déplacement

- Un octet d'adresse effective donnée peut être **accédée de plusieurs manières.**
- Un programme utilise plusieurs segments mémoire:
- **Le segment de code, le segment de données, le segment de pile « Important »**
- Leur attribution en mémoire est réalisée **automatiquement** par le système d'exploitation

- Une adresse se présente sous la forme **segment:offset (Adresse Logique)**
- **Adresse effective = Adresse physique**
- 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.



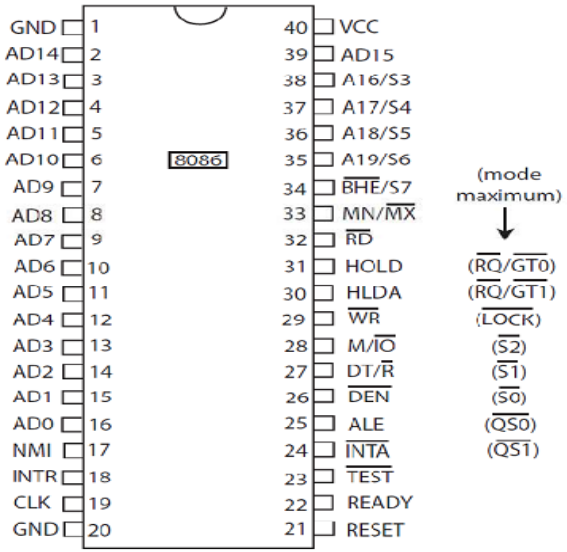
2. L'architecture du 8086

- ***2.1. L'architecture externe du 8086***
- Le microprocesseur 8086 est un circuit intégré:

✓ Il possède un bus multiplexé **adresse/donnée de 20 bits.**

✓ Le **bus de donnée** occupe **16 bits** ce qui permet d'échanger des mots de **2 octets**

- Le **bus d'adresse** occupe **20 bits** ce qui permet d'adresser **1 Mo**



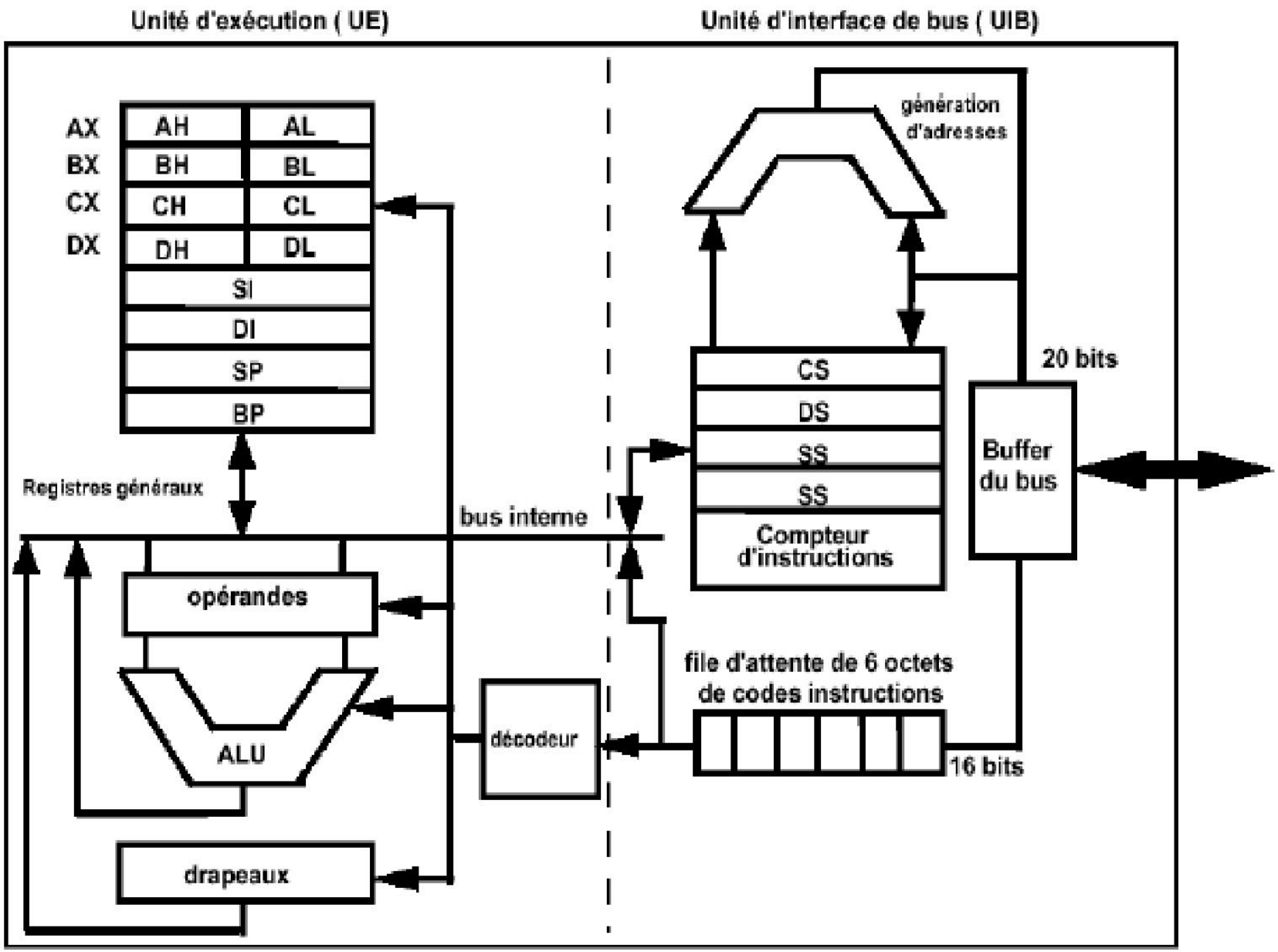
2.2. L'architecture interne du 8086

- Deux unités internes distinctes : **l'UE (Unité d'Exécution)** et **l'UIB (Unité d'Interfaçage avec le Bus).**

✓ **L'UIB** permet de **récupérer** et **stocker les informations à traiter**, (l'interface physique entre le microprocesseur et le monde extérieur)

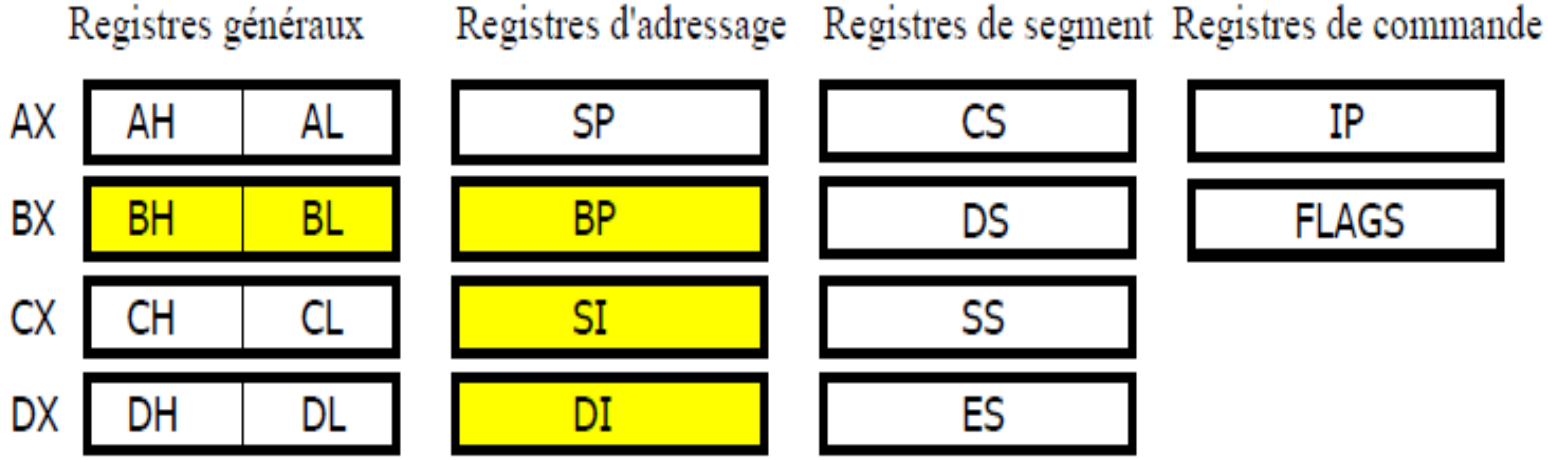
✓ **L'UE exécute les instructions** transmises par l'UIB.

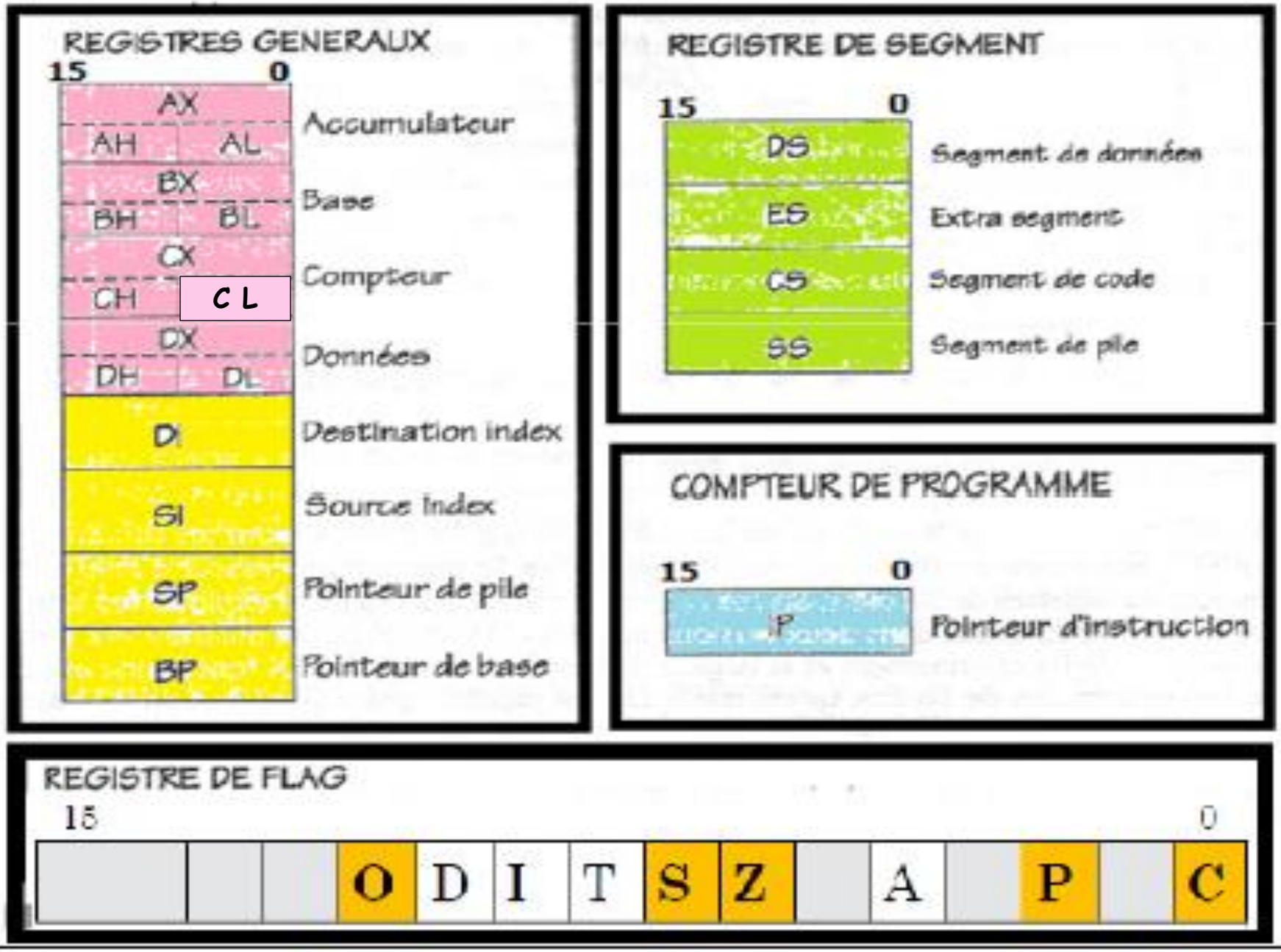
- Comporte essentiellement l'UAL de 16 bits qui manipule les registres généraux de 16 bits aussi.
- Pendant que l'UE exécute les informations qui lui sont transmises, **l'instruction suivante est chargée dans l'UIB**.



L'architecture interne du 8086

- ## *2.3. Les registres du 8086:*
- Tous les registres du 8086 sont structurés en 16 bits.
- Vu de l'utilisateur, le 8086 comprend **3 groupes de 4 registres de 16 bits**, un **registre d'état de 9 bits** et un **compteur programme de 16 bits** non accessible par l'utilisateur.
- **Les données** dont le processeur 8086 a besoin sont stockées dans des registres **(AX, BX, CX, DX, IP, SI, CS, DS, SS, ...)**.
- Les registres **AX, BX, CX et DX** sont les registres les plus utilisés pour **les calculs**.

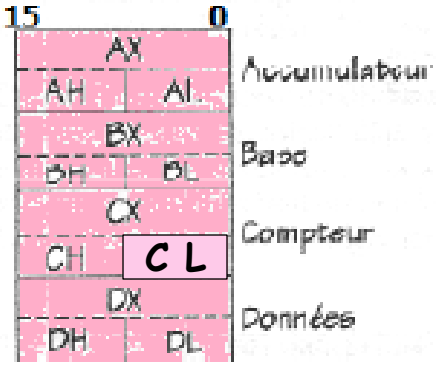



REGISTRES GENERAUX
15
0
AX
AH
AL
Accumulateur
BX
BH
BL
Base
CX
CH
CL
Compteur
DX
DH
DL
Données
DI
Destination index
SI
Source Index
SP
Pointeur de pile
BP
Pointeur de base
REGISTRE DE SEGMENT
15
0
DS
Segment de données
ES
Extra segment
CS
Segment de code
SS
Segment de pile
COMPTEUR DE PROGRAMME
15
0
IP
Pointeur d'instruction
REGISTRE DE FLAG
15
0
O
D
I
T
S
Z
A
P
C

REGISTRES GÉNÉRAUX

(GROUPE DE DONNÉES)

Registres	Usage
AX: Accumulateur	Usage général, Obligatoire pour la multiplication et la division, Ne peut pas servir pour l'adressage
BX : Base	Usage général, Adressage
CX : Comptage et calcul	Usage général, Compteur de répétition. Ne peut pas servir pour l'adressage
DX : Data	Usage général, Extension au registre AX pour contenir un nombre 32 bits Dans la multiplication et la division 16 bits Ne peut pas servir pour l'adressage

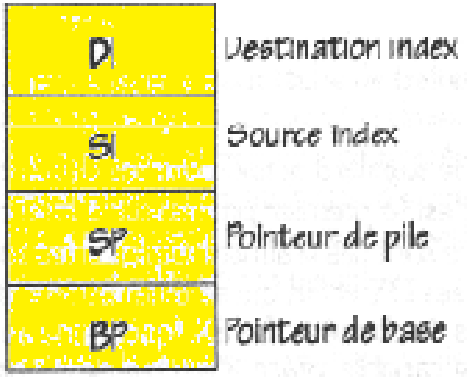


Groupe de données: formé par 4 registres de 16 bits (AX, BX, CX et DX) chaque registre peut être divisé en deux registres de 8 bits : (AH, AL, BH, BL, CH, CL, DH et DL)

REGISTRES GÉNÉRAUX

(D'ADRESSAGE)

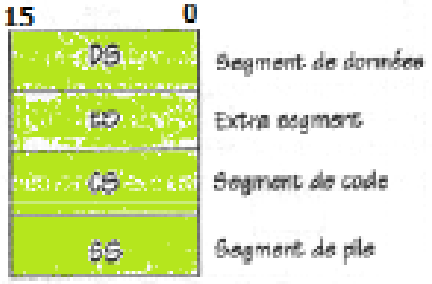
Registres	Usage
DI : Registre d'index (destination)	Usage général Adressage comme registre d'index de l'opérande destination
SI : Registre d'index (source)	Usage général Adressage comme registre d'index de l'opérande source.
SP : Pointeur de Pile	Utilisé pour l'accès à la pile. Pointe sur la tête de la pile
BP : Pointeur de Base	Usage général Adressage comme registre de base



Un groupe de pointeur et indexe: formé de 4 registres de 16 bits : (SI, DI, SP et BP) et font généralement référence à un emplacement en mémoire (segment).

REGISTRES DE SEGMENT

Registres	Usage
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



COMPTEUR D'INSTRUCTION et REGISTRE D'ÉTAT (FLAGS)

Le compteur d'instruction (IP), appelé aussi Compteur Ordinal (C.O.) permet de pointer TOUJOURS le premier octet de l'instruction suivante.

9

CF : indicateur de retenue (carry) ;
PF : indicateur de parité;
AF : indicateur de retenue auxiliaire ;
ZF : indicateur de zéro ;
SF : indicateur de signe ;
TF : indicateur d'exécution pas à pas (trap) ;
IF : indicateur d'autorisation d'interruption ;
DF : indicateur de décrémentation ;
OF : indicateur de dépassement (overflow).

FLAGS

3. LES MODES D'ADRESSAGE PROPRES AU 8086

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

1. Adressage implicite : L'instruction n'utilise aucun opérande ou concerne des registres implicites.

```
Exemple : RET        ; retour d'un sous-programme
          MOVSB      ; copie l'octet d'adresse SI dans celui d'adresse DI
```

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)
          MOV AL, 3Ah    ; copie dans AL la valeur 3A hexa
```

3. Adressage Direct

- Un opérande est donné en mode direct s'il est désigné par son adresse effective dans la mémoire.
- ***Exemple 1:*** MOV [1200h],5Ah
- La valeur 5Ah sera stockée à l'adresse effective ***1200h***.

- ***Exemple 2:*** MOV [1200h], 4D59h
- 4D59h est une donnée sur 16 bits, elle occupera les deux octets d'adresse effective ***1200h*** et ***1201h***.
- Si DS = 720h, la répercussion de cette instruction sur la mémoire sera:

0720:1200	***59h***
0720:1201	***4Dh***

- Rappelons que *0720:1200* est la représentation *(Segment : Offset)* de l'adresse physique 08400h.

4 *Adressage de registre*

l'opérande se trouve dans un registre de 8 ou 16 bits.

Exemple :

```
MOV DX, DI        ; copie dans DX la valeur contenue dans DI
MOV AL, BH        ; le contenu de BH est copié dans AL (8 bits)
```

5 *Adressage de la mémoire* **(Indirect)**

Une seule des éventuelles opérandes d'une instruction peut être stockée dans la mémoire. Son offset est calculé par la formule générale suivante :

Offset = Base + Index + déplacement

Base = { BX, BP }

Index= { DI, SI }

déplacement = { Aucun, valeur 8 bit, valeur 16 bits)

➔ Possibilité d'adressage très variés

INDIRECT

Formule de L'offset	Mode d'adressage	Combinaisons possibles
Déplacement	Adressage direct	[d8] [d16]
Base	Adressage basé	[BX] [BP]
Base +déplacement		[BX]+[d8] [BX]+[d16] [BP]+[d8] [BP]+[d16]
Index	Adressage indexé	[DI] [SI]
Index+ déplacement		[DI]+[d8] [SI]+[d16] [DI]+[d16] [SI]+[d8]
Base+Index	Adressage basé indexé	[BX + SI] [BX + DI] [BP + SI] [BP + DI]
Base+déplacement +Index		[BX + SI] + [d8] [BX + DI] + [d8] [BP + SI] + [d8] [BP + DI] + [d8] [BX + SI] + [d16] [BX + DI] + [d16] [BP + SI] + [d16] [BP + DI] + [d16]

5. Adressage indirect

Adressage Relatif: ajouter un déplacement pour un adressage basé ou indexé. Ex: Mov [BX]+12 , Mov [DI]+32

Exemples d'adressage indirect

Adressage basé

```
MOV DX, [BX]          ; copie dans DX le mot d'adresse DS:BX
MOV DX, TABLE[BX]     ; copie dans DX le mot d'adresse S:(BX+TABLE)
MOV 5[BP], AL         ; AL est copié dans l'octet d'adresse SS:(BP+5)
MOV DH, [BX + VEC]
```

Adressage indexé

```
MOV BX, [DI]          ; copie le mot d'adresse DS:DI dans le registre BX
MOV AL, 13[SI]        ; copie dans AL l'octet d'adresse DS:(SI+13)
MOV [VCT + SI], AX
MOV [SI], AH
```

Adressage basé indexé

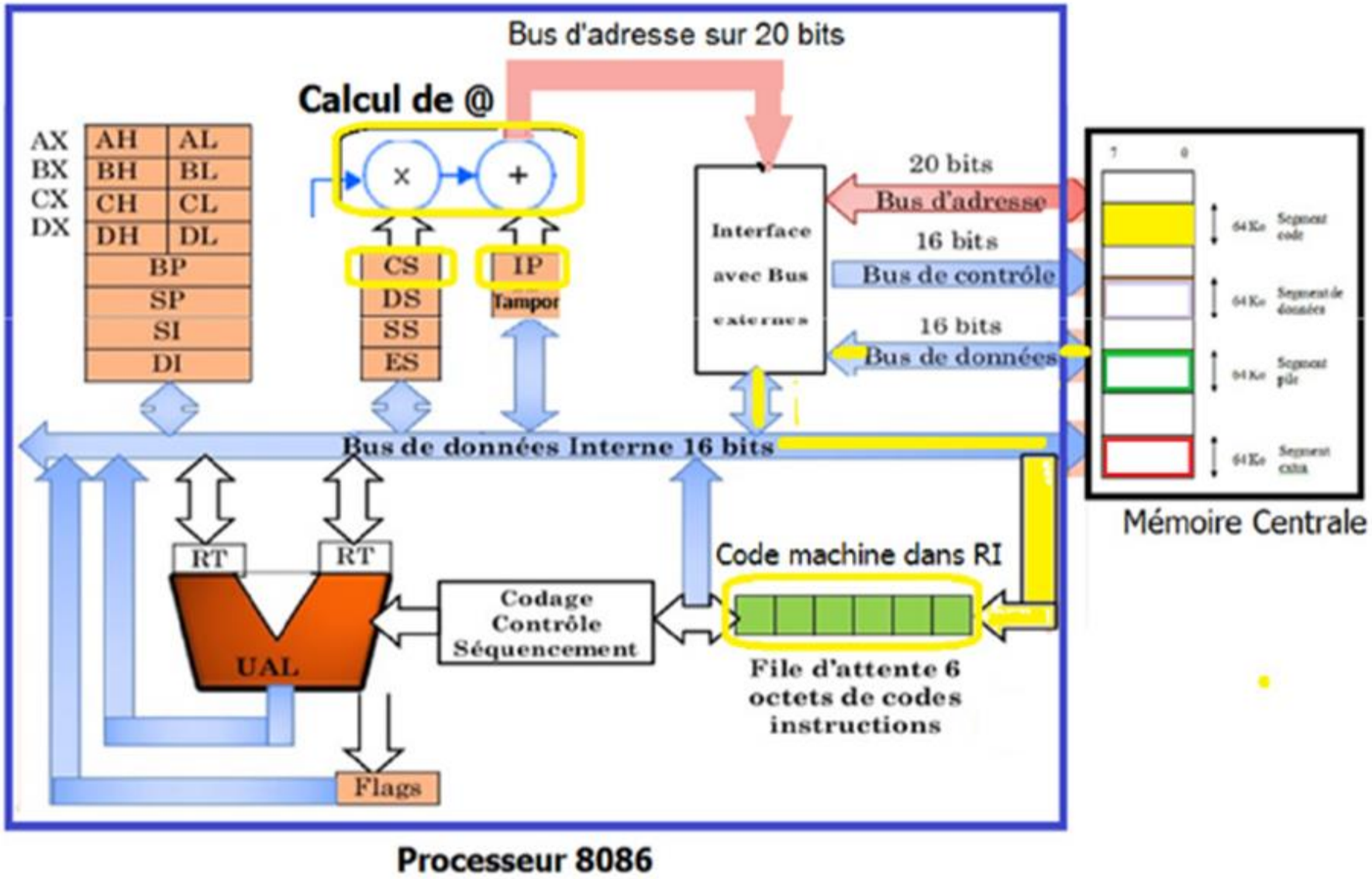
```
MOV AL, [BX + SI + 2]  ; copie l'octet d'adresse DS:(BX+SI+2) dans AL
MOV TABLE[BP][DI], AX
```

4. Étapes de fonctionnement d'une instruction dans le 8086

- Le langage assembleur est **très proche** du **langage machine** (des informations en **binaire**)
- Il dépend donc du type de processeur
- Langage assembleur consiste à écrire sous forme **symbolique** la succession d'instructions
- Ces instructions sont stockées dans un fichier texte **(le fichier source)** qui, grâce à un programme spécifique **(assembleur)** sera traduit en langage machine.
- L'avantage de l'assembleur est de **générer** des programmes efficaces et rapides à l'exécution.

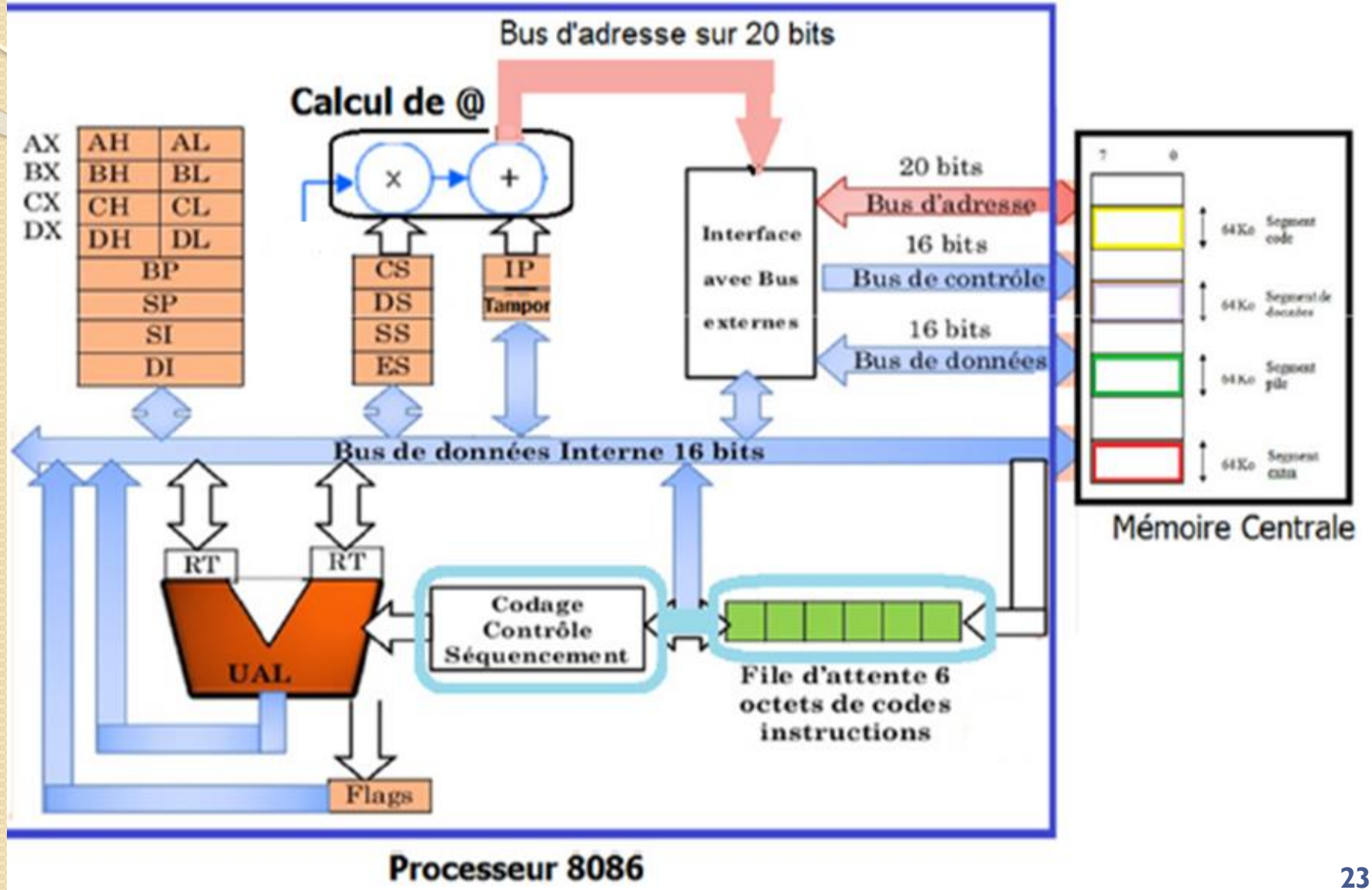
❶ Recherche d'instruction

RI ← [CS:IP]



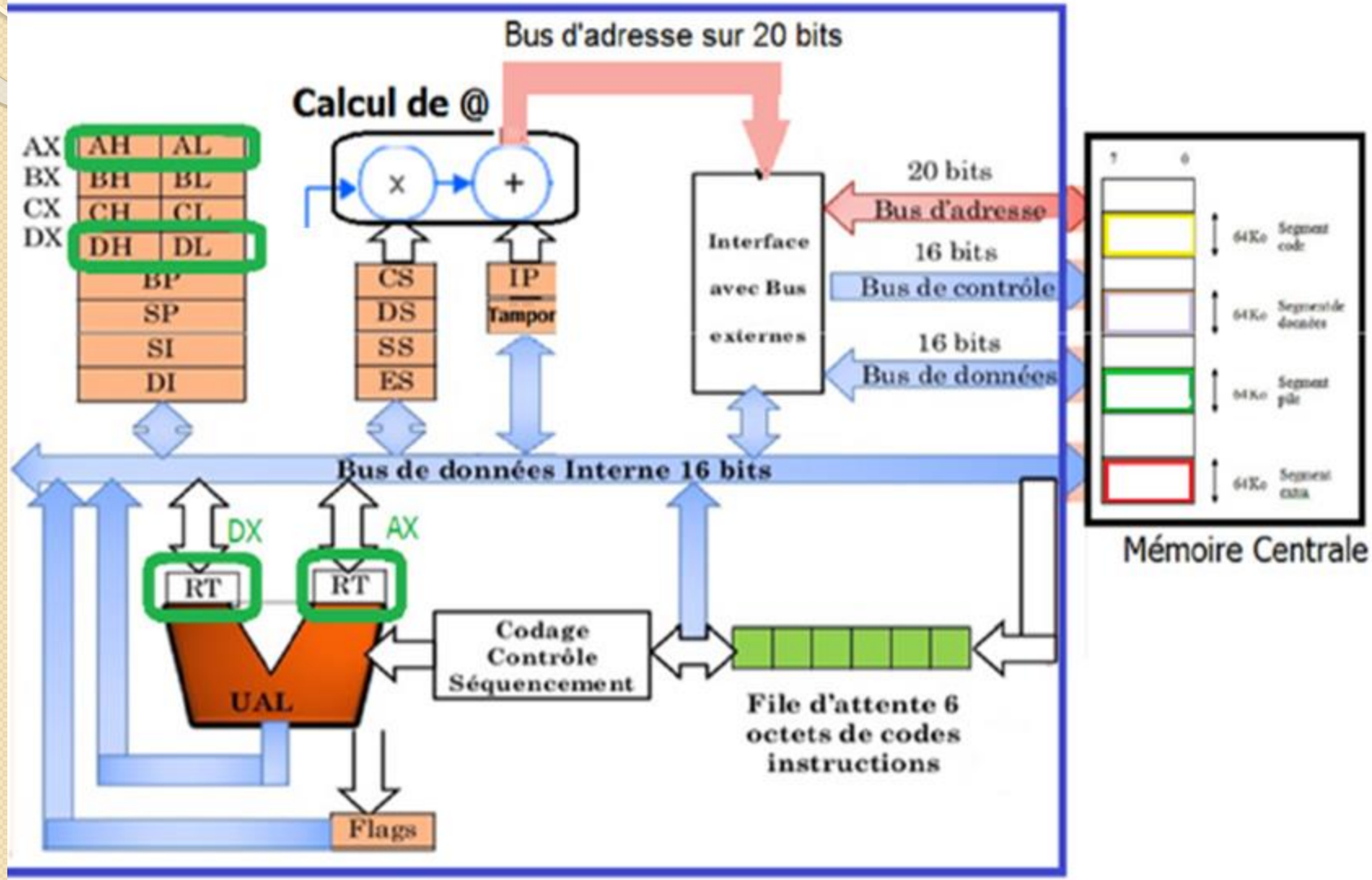
❷ Décoder l'instruction :

Dec ← RI



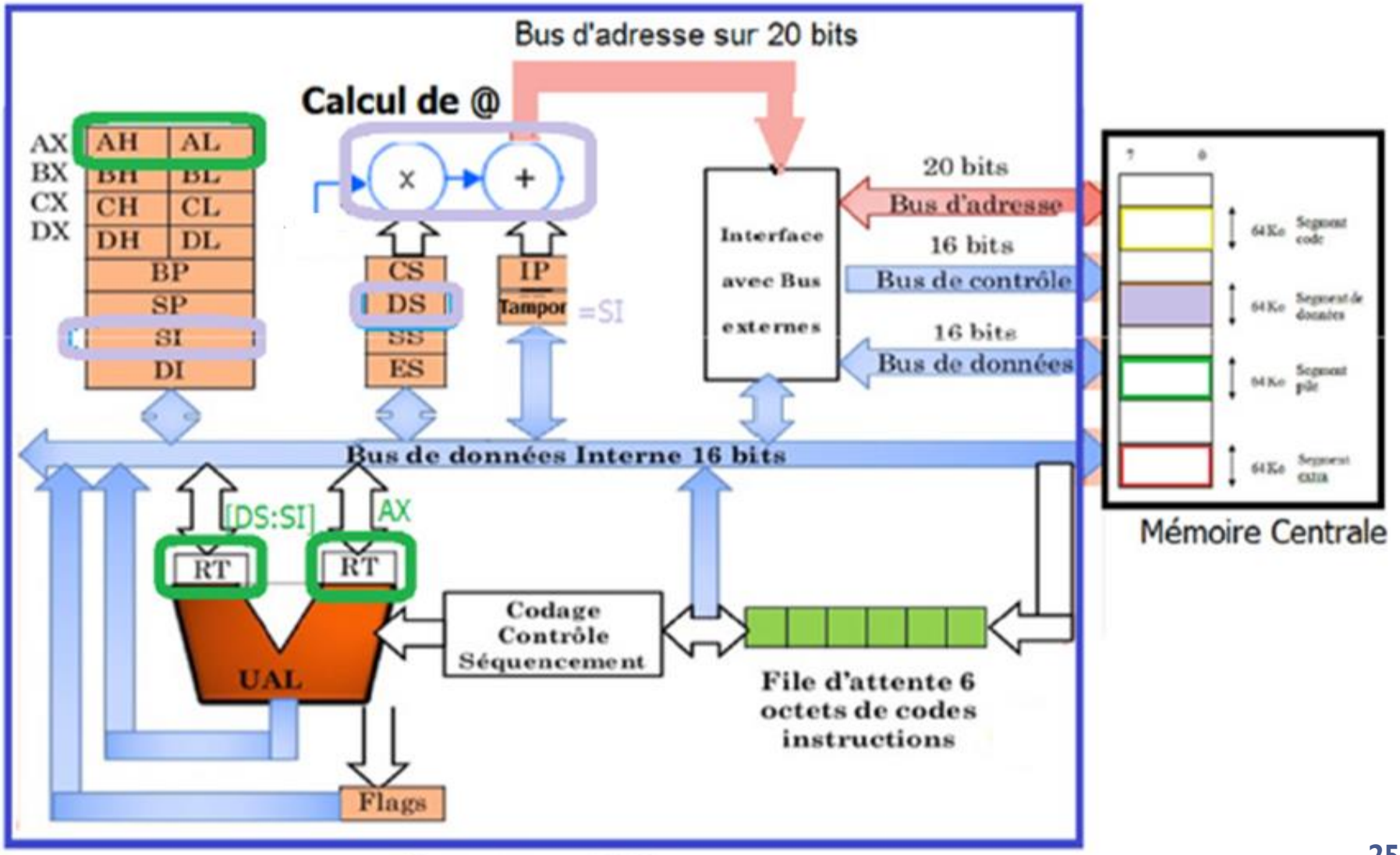
❸ Charger l'Opérande

instruction Reg1, Reg2: RT1 ←Reg1 et RT2 ←Reg2



Charger l'Opérande

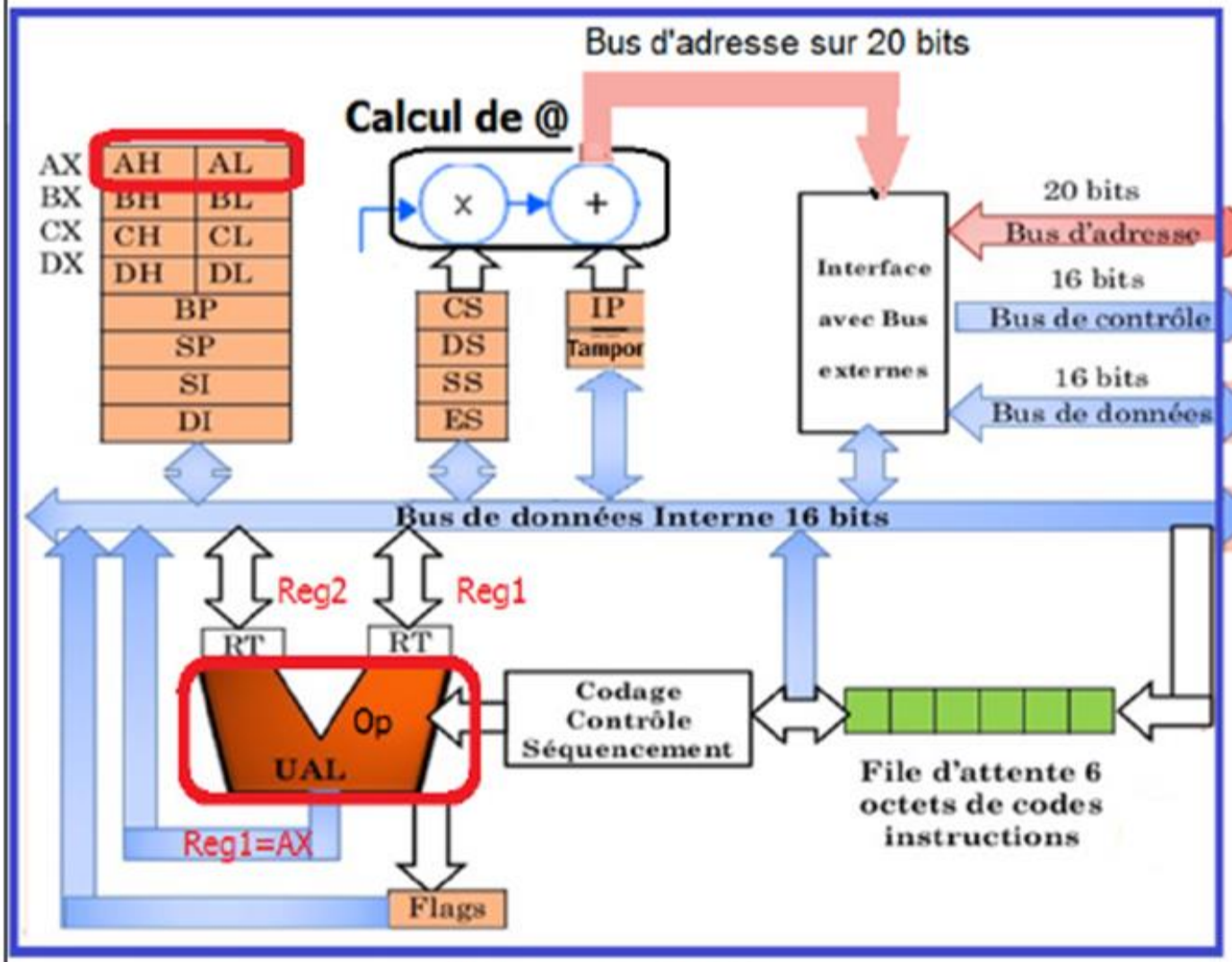
❸ instruction Reg1, mem(imm) ou mem(imm), Reg1:

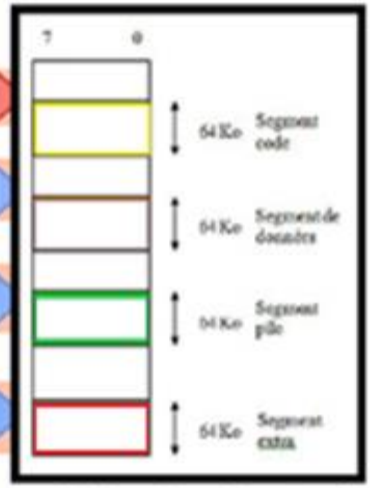


❹ Exécuter l'Instruction

instruction Reg1, Reg2:

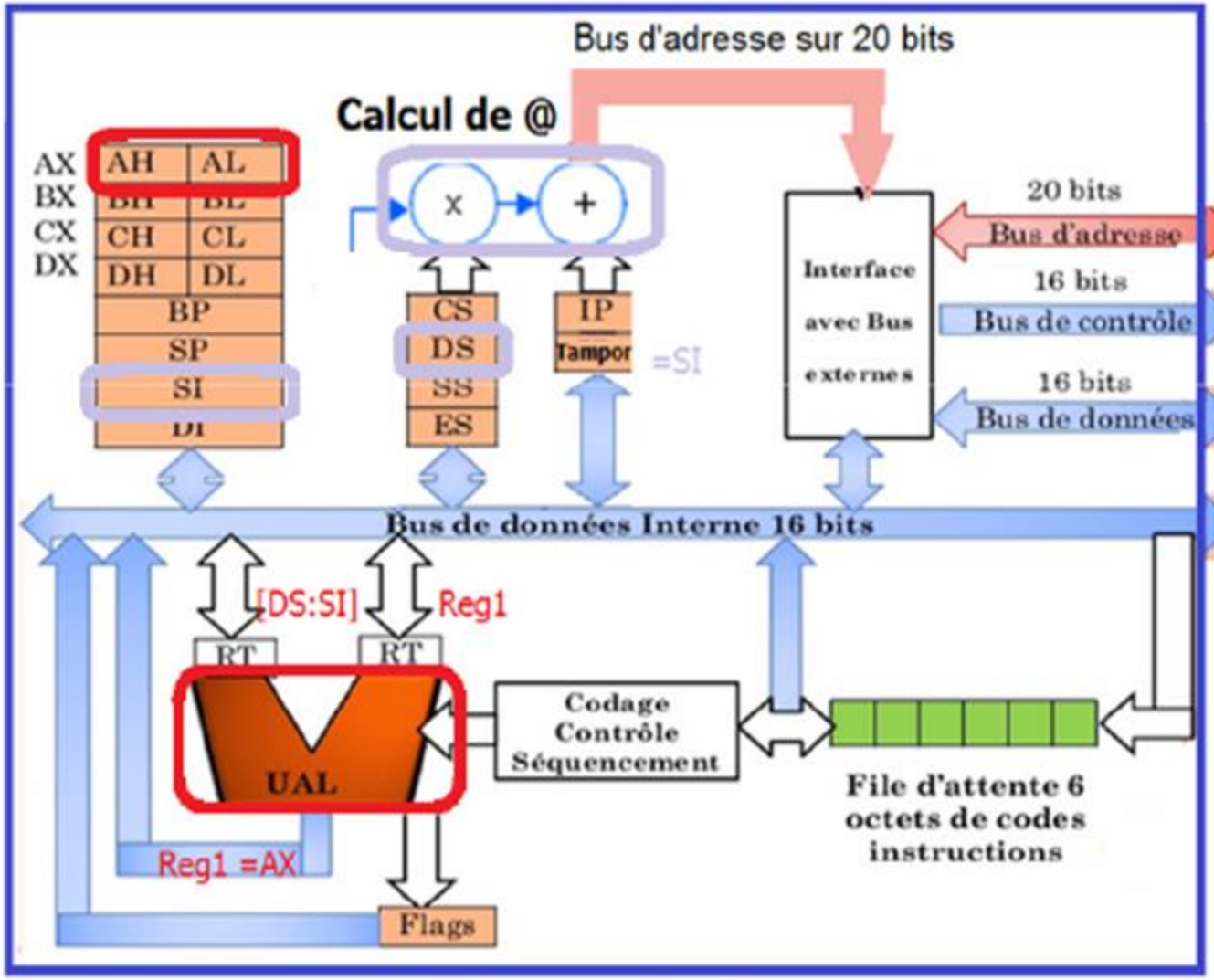
Reg1 ← RT1 Op RT2

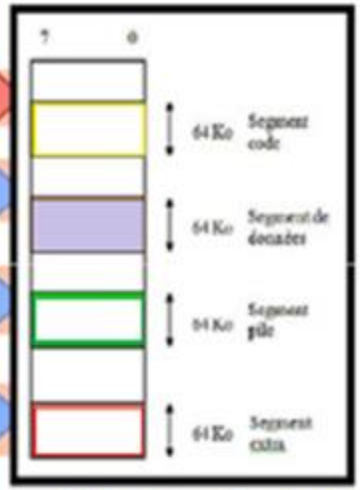




❹ Exécuter l'Instruction

instruction Reg1, mem(imm):
Reg1 ← RT1 Op RT2



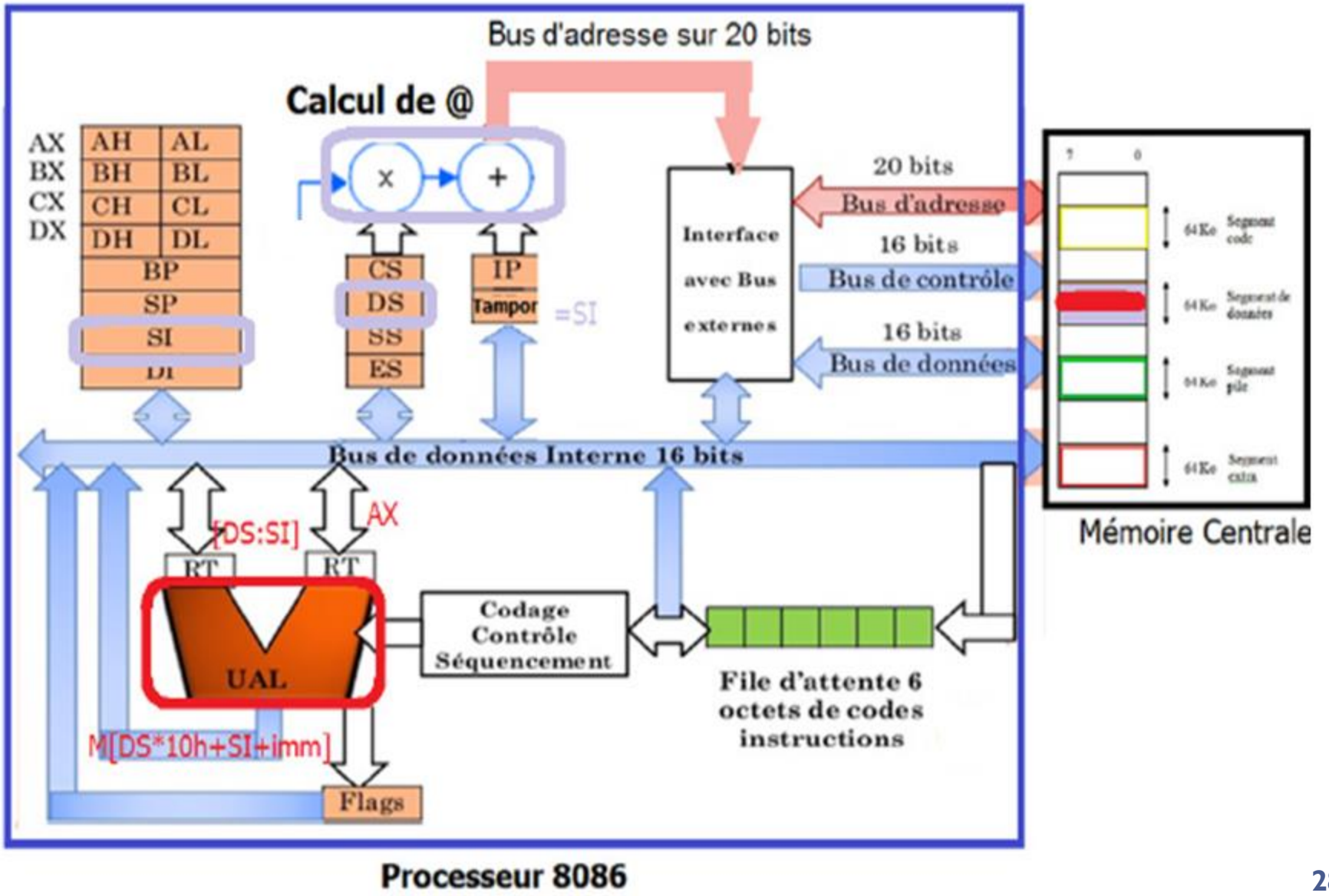


Mémoire Centrale

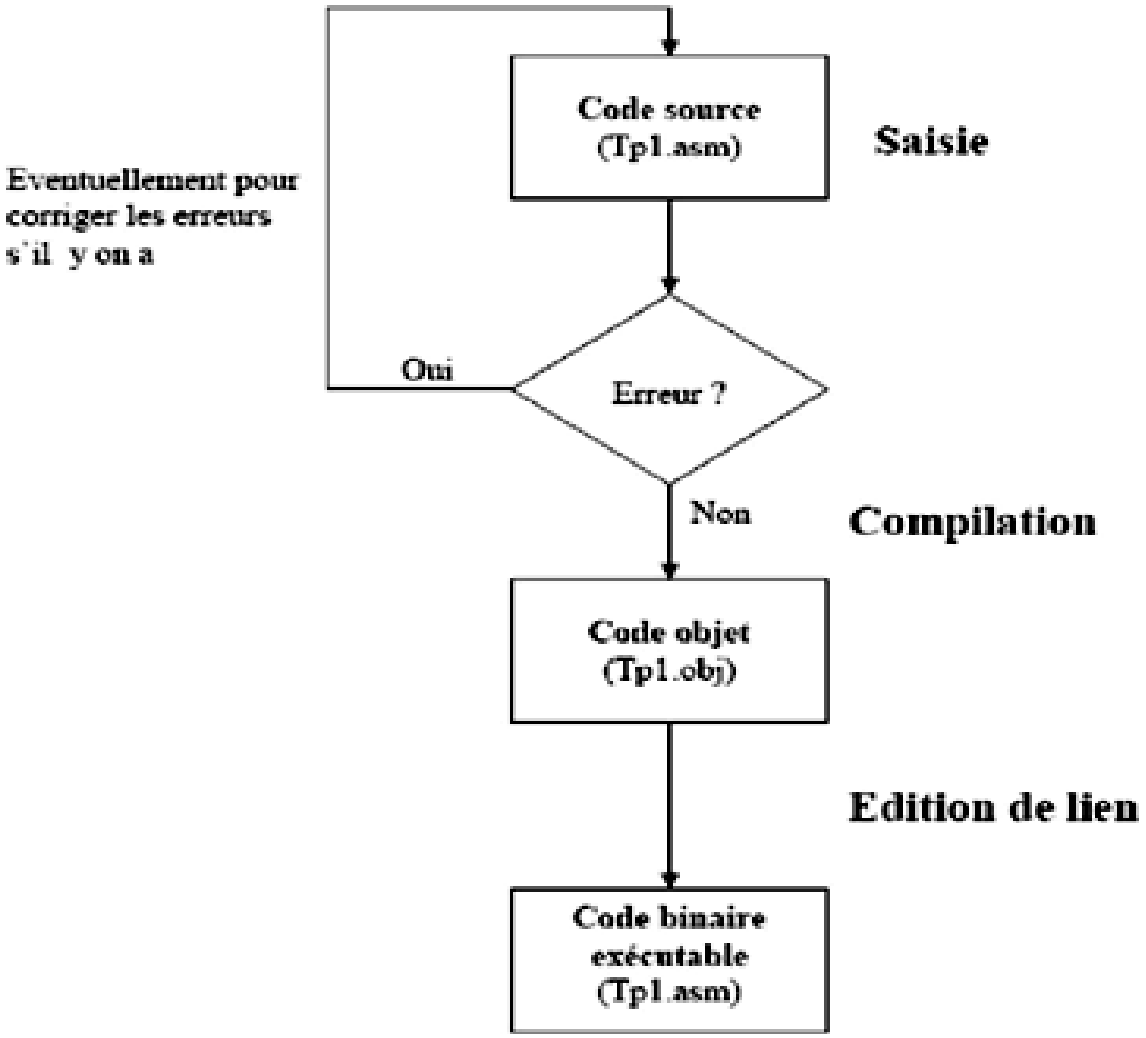
Processeur 8086

❹ Exécuter l'Instruction

instruction mem(imm),Reg1:



- **Cycle d'exécution d'un programme en langage assembleur:**



- ***4.1. Constitution d'un programme***
- Un programme assembleur = ensemble de déclarations **(directives +d'instructions)**

✓ **Les directives**

- **Les directives de données: (EQU; DB; DW ; DD)**

  Exemples : Pi **EQU** 3.14 ; valeur constante

- **Déclaration des variables :**
- ***DB (define byte) ;*** Exemple : *Nb_max DB 123*
- ***DW (define word);*** réserve des mots à deux octets
- ***DD (define double);*** réserve des mots à quatre octets
- ***Dup :*** Lorsque l'on veut déclarer un tableau de **n** cases, toutes initialisées à la **même valeur**

  Exemples : tab DB 100 dup (15) ; 100 octets valant 15

- **Les directives Word PTR et Byte PTR**

On doit utiliser une directive spécifiant la taille de la donnée à transférer :

Exemples : MOV byte ptr [BX], val ; concerne 1 octet

MOV word ptr [BX], val ; concerne 1 mot de 2 octets

✓ Les directives de segment et de procédure (SEGMENT ;ASSUME ; PROG/END PROG)

- ***SEGMENT*** : Elle permet de contrôler la **relation** entre la génération du code objet et la gestion des segments logiques ainsi générés.

✓ *Syntax de la directive:*

nom_seg SEGMENT [align][combine]['class']

- ***ASSUME :*** La directive ASSUME permet d'indiquer à l'assembleur quel est le segment de données **et** celui de codes; il s'agit d'initialiser le segment de données DS (même chose pour : ES et SS)
  Exemple: MOV AX, nom_du_segment_de_données;
- ***PROG / END PROG :*** Les directives permettent d'indiquer le début et la fin du programme.

```
PROGRAM  Exemple1
Pile SEGMENT STACK
        ; On met les directives pour réserver de l'espace mémoire.
PILE ENDS
Data SEGMENT
        ; On met les directives de données pour réserver de la mémoire
        ; Pour les variables qui seront utilisées dans le programme.
Data ENDS
Extra SEGMENT
        ; On met les directives pour déclarer les variables (les chaînes de
        ; Caractères).
Extra ENDS
Code SEGMENT
    ASSUME  cs : code, ds, data, es : pile :ss :pile
PROG
    Mov AX,Data
    Mov DS,AX
    Mov AX,Extra
    Mov ES,AX
    Mov AX,pile
    Mov SS,AX
        ; mettre les instructions du programme
Code ENDS
  END PROG
```

4.2. Les instructions

- Chaque microprocesseur possède un ensemble d'instructions fixé par le constructeur
- Le 8086 possède 92 types d'instructions de base qu'on peut les diviser en groupes suivants:
- **Instructions de transfert de données.**
- **Instructions arithmétiques**
- **Instructions logiques.**
- **Instructions de sauts de programme.**
- **Instructions de boucles de répétition**

➢ Instructions de transfert de données :

- Elles permettent de déplacer des données d'une source vers une destination :
  - ✓ ***Registre vers mémoire ;***
  - ✓ ***Registre vers registre ;***
  - ✓ ***Mémoire vers registre.***
- ***Remarque :*** le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire

(Pour ce faire, il faut passer par un **registre intermédiaire**)

Usage	Nom	Fonction
Général	MOV	Transfert d'octets ou de mots
	PUSH	Chargement de la pile
	POP	Déchargement de la pile
	PUSHA	Chargement de tous les registres dans la pile
	POPA	Déchargement de tous les registres dans la pile
	XCHG	Echange d'octet ou de mot
	XLAT	Translation d'octet
Entrées-sorties	IN	Entrée de mot ou d'octet
	OUT	Sortie de mot ou d'octet
Adresses	LEA	Chargement de l'adresse effective
	LDS	Chargement du pointeur avec DS
	LES	Chargement du pointeur avec ES
Indicateurs	LAHF	Transfert des indicateurs dans AH
	SAHF	Rangement de AH dans les indicateurs
	PUSHP	Chargement des indicateurs dans la pile
	POPF	Déchargement des indicateurs de la pile.

- **MOV :** Elle permet de transférer les données (un octet ou un mot) d'un registre à un autre registre ou d'un registre à une case mémoire, sa syntaxe est comme suit :

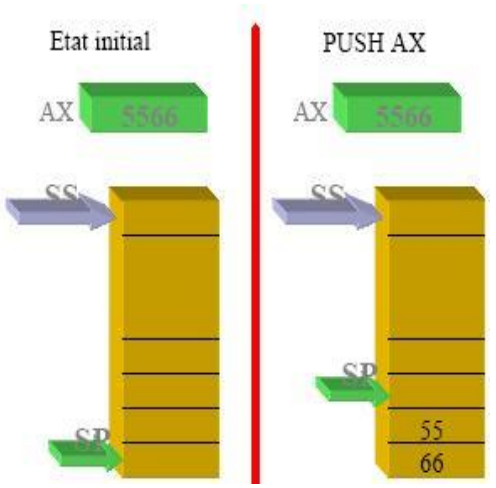
MOV destination, source

Exemples :

- MOV AX, BX : Transfert d'un registre de 16 bits vers un registre de 16 bits.
- MOV AH, CL : Transfert d'un registre de 8 bits vers un registre de 8 bits.
- MOV AX, [Val1] ; Transfert du contenu d'une case mémoire 16 bits vers AX.
- MOV [Val2], AL ; Transfert du contenu du AL vers une case mémoire d'adresse Val2.

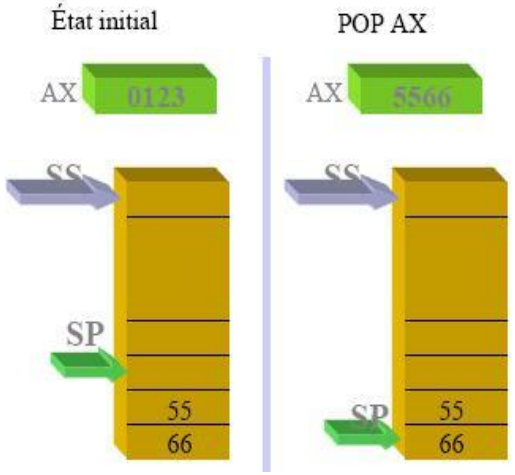
- **PUSH :** Elle permet d'empiler les registres du CPU sur le haut de la pile.

Syntaxe : **PUSH source**

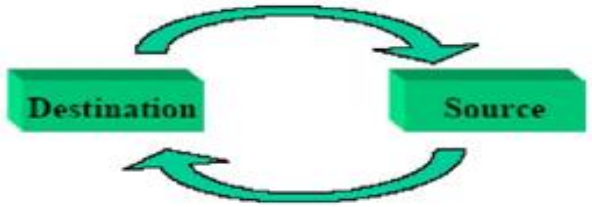


- **POP :** Elle permet de dépiler les registres du CPU sur le haut de la pile.

Syntaxe : **POP destination**



- **XCHG :** Elle permet **de permuter** la source avec la destination



Exemple :

XCHG AX, BX : elle permute entre AX et BX.

Si AX=8976 et BX=5678,

Alors après l'exécution de l'instruction on aura AX=5678 et BX=8976

- **IN / OUT : Envoyer ou de recevoir un octet on utilise l'accumulateur AL**, ou s'il s'agit **d'envoyer ou de recevoir un mot on utilise l'accumulateur AX**.

Syntaxe : **IN** ACCUMULATEUR, DX

OUT DX, ACCUMULATEUR

Instructions arithmétiques

Usage	Nom	Fonction
Addition	ADD	Addition sur un octet ou un mot
	ADC	Addition sur un octet ou un mot avec retenue
	INC	Incrémentation de 1
	AAA	Ajustement ASCII
	DAA	Ajustement décimal
Soustraction	SUB	Soustraction sur un octet ou un mot
	SBB	Soustraction sur un octet (mot) avec retenue
	DEC	Décrémentation de 1
	NEG	Mètre un octet ou un mot en négatif
	CMP	Comparaison d'octet ou mot
	AAS	Ajustement ASCII
	DAS	Ajustement décimal

Usage	Nom	Fonction
Multiplication	MUL	Multiplication d'octet ou de mot **non signée**
	IMUL	Multiplication d'octet ou de mot **signée**
	AAM	Ajustement ASCII
Division	DIV	Division d'octet ou de mot **non signée**
	IDIV	Division d'octet ou de mot **signée**
	AAD	Ajustement ASCII
	CBW	Conversion d'un octet en un mot
	CWD	Conversion d'un mot en double mots

- **ADD (Addition) :** Elle permet d'additionner le contenu de la source (octet ou un mot) avec celui de la destination le résultat est mis dans la destination.

  Syntaxe : ADD Destination, source

Exemples :

ADD AX, BX ;AX = AX + BX (addition sur 16 bits).

ADD AL, [SI] ;AL = AL + le contenu de la case mémoire pointé par SI.

ADD [DI], AL ; le contenu de la case mémoire pointé par DI est additionnée avec AL, le résultat est mis dans la case mémoire pointé par DI.

- **ADC (Addition avec retenue) :** Elle permet d'additionner le contenu de la source (octet ou un mot) avec celui de la destination et la retenue (CF); le résultat est mis dans la destination.

Syntaxe : ADC Destination, source

Exemples :

ADC AX, BX ;AX = AX + BX + CF (addition sur 16 bits)

ADC AL, BH ;AL = AL + BH + CF (addition sur 8 bits)

ADC AL, [SI] ;AL = AL + le contenu de la case mémoire pointé par SI + CF

- **SUB (Soustraction) :** Elle permet de soustraire la destination de la source (octet ou un mot) le résultat est mis dans la destination.

Syntaxe : SUB Destination, source

Exemples :

SUB AX, BX ;AX = AX - BX (Soustraction sur 16 bits

- SUB AL, BH ;AL = AL - BH (Soustraction sur 8 bits).
- SUB AL, [SI] ; AL = AL - le contenu de la case mémoire pointé par SI.
- SUB [DI], AL ; le contenu de la case mémoire pointé par DI= le contenu de la case mémoire pointé par DI - AL .

- **SBB (Soustraction avec retenue) :** Elle permet de **soustraire** la **destination de la source et la retenue** (octet ou un mot) le résultat est mis dans la destination.

Syntaxe : SBB Destination, source

Exemples :

- SBB AX, BX ;AX = AX – BX - CF (Soustraction sur 16 bits).
- SBB AL, [SI] ;AL = AL - le contenu de la case mémoire pointé par SI – CF.

✓ **INC (Incrémentation):** Elle permet d'incrémenter le contenu de la destination.

Syntaxe : INC Destination

Exemples :

INC AX ;AX = AX + 1 (incrémentation sur 16 bits).

INC AL ;AL = AL +1 (incrémentation sur 8 bits).

INC [SI] ;[SI] = [SI] + 1 Le contenu de la case mémoire pointé par SI sera incrémenter.

✓ **DEC (Décrémentation) :** elle permet de décrémenter le contenu de la destination.

Syntaxe : DEC Destination

- Exemples :
- DEC AX ;AX = AX - 1 (décrémentation sur 16 bits).
- DEC [SI] ; [SI] = [SI] – 1 ; le contenu de la case mémoire pointé par SI sera décrémenté.

- **NEG (Négatif) :** elle **soustrait** l'opérande **destination** (octet ou mot) **de 0** le résultat est stocker dans la destination. Donc cette opération réalise le **complément à deux** d'un nombre.

Syntaxe : **NEG** Destination

Exemples :

- NEG AX ;AX = 0 – AX.
- NEG AL ;AL = 0 – AL.
- NEG [SI] ; [SI] = 0 - [SI].

- ✓ **CMP (Comparaison):** Elle **soustrait la source de la destination**, qui peut être un octet ou un mot, le résultat n'est pas mis dans la destination, en effet ***cette instruction touche uniquement les indicateurs*** pour être tester ***avec un autre instruction ultérieure de saut conditionnel***.
- Les indicateurs susceptibles d'être touché sont : AF, CF, OF, PF, SF, ZF. Cette instruction va nous **permettre de comparer deux nombres**.

  Syntaxe : **CMP** Destination , Source

- ✓ **MUL :** Elle effectue une multiplication **non signée** de **l'opérande source avec l'accumulateur**.

  Syntaxe : **MUL** Source

- Si la source est **un octet** alors elle sera **multipliée par l'accumulateur AL** le **résultat sur 16 bits** sera **stocké dans le registre AX**.
- Si la source est **un mot** alors elle **sera multipliée avec l'accumulateur AX** le **résultat de 32 bits** sera **stocké dans la paire des registres AX et DX**. Cette multiplication **traite les données en tant que nombres non signés**.

- ✓ **DIV :** Elle effectue **une division non signée** de **l'accumulateur** par l'opérande source.

  Syntaxe : **DIV** Source

- Si **l'opérande est un octet**, alors **on récupère le quotient** dans le registre **AL** et **le reste dans le registre AH**.
- Si **l'opérande est un mot**, alors **on récupère le quotient** dans le registre **AX** et **le reste dans le registre DX**.

Instructions logiques

Usage	Nom	Fonction
Logique	NOT	Inversion logique sur un octet ou un mot
	AND	Et logique
	OR	Ou logique
	XOR	Ou exclusif
	TEST	Et logique sans résultat, affecte uniquement les indicateurs du registre des flags.
Décalages	SHL	Décalage logique à gauche
	SAL	Décalage arithmétique à gauche
	SHR	Décalage logique à droite
	SAR	Décalage arithmétique à droite
Rotation	ROL	Rotation à gauche
	ROR	Rotation à droite
	RCL	Rotation à gauche à travers le bit de retenue
	RCR	Rotation à droite à travers le bit de retenue

- ✓ **NOT (Négation) :** Elle réalise la **complémentation à 1** d'un nombre.

Syntaxe : **NOT** Destination

Exemple :

- MOV AX, 500H ;AX = 0000 0101 0000 0000
- NOT AX ;AX = 1111 1010 1111 1111

- ✓ **AND :** Elle permet de faire un **ET logique** entre la destination et la source (octet ou mot) le résultat est mis dans la destination.

Syntaxe : **AND** Destination, source

Exemples :

- MOV AX , 503H ;AX = 0000 0101 0000 0011
- AND AL, BH ;AL = AL AND BH (ET logique sur 8 bits)
- AND AL, [SI] ;AL = AL AND le contenu de la case mémoire pointé par SI.

- **OR :** Elle permet de faire un **OU logique** entre la destination et la source (octet ou mot) le résultat est mis dans la destination.

Syntaxe : **OR** Destination, source

Exemples :

- MOV AX , 503H ;AX = 0000 0101 0000 0011
- OR AX , 0201H ; 0000010100000011 OR 0000000100000001 = 0000011100000011

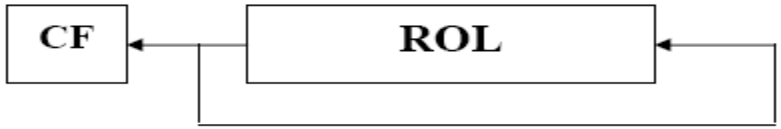
✓ **SHL :** Elle permet de faire **le décalage logique à gauche** de la destination.

Syntaxe : **SHL** destination, compteur

Exemples :

- SHL AL,1 ; Pour faire un seul décalage.
- MOV CL, 3
- SHL AL, CL } Pour faire trois décalages de suite

✓ **ROL :** Elle permet de faire la rotation à gauche de la destination.



Syntaxe : **ROL** destination, compteur

Exemples :

- ROL AL,1 ;Pour faire une seule rotation
- MOV CL,4
- ROL AL,CL } ;Pour faire quatre rotations de suite

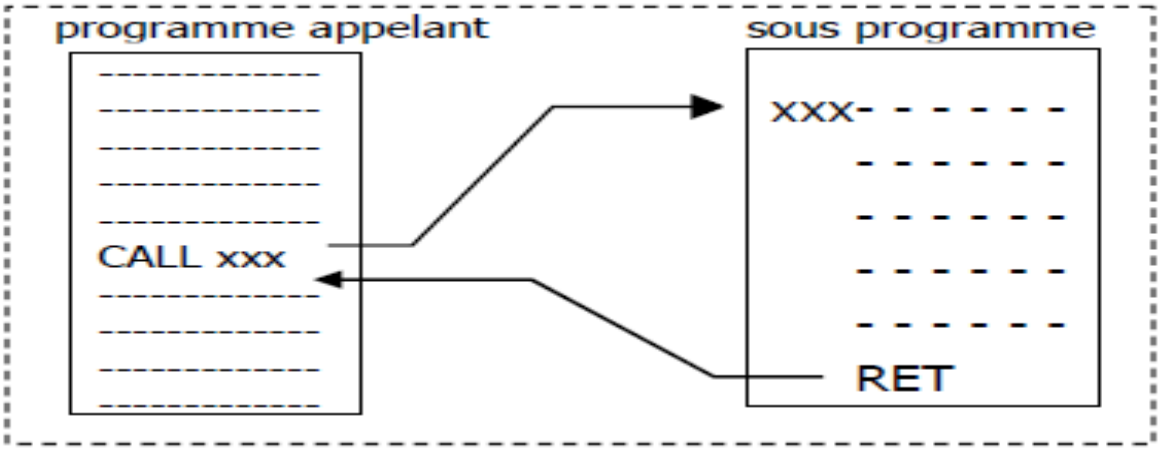
Instructions de sauts et de branchement

Type	Nom	Fonction
Branchements inconditionnels	CALL	Appel à un sous programme
	RET	Retour d'un sous programme
	JMP	Saut
Branchements conditionnels (arithmétique non signée)	JA/JNBE	Si supérieur / Si non inférieur ou non égal
	JAE/JNB	Si supérieur ou égal/ Si non inférieur
	JB/JNAE	Si inférieur/si non supérieur ni égal
	JBE/JNA	Si inférieur ou égal/si non supérieur.
Branchements conditionnels (arithmétique signée)	JG/JNLE	Si plus grand/si pas inférieur ni égal
	JGE/JNL	Si plus grand ou égal/Si pas inférieur
	JL/JNGE	Si moins que/Si pas plus grand ni égal
	JLE/JNG	Si moins que ou égal/Si pas plus grand

Type	Nom	Fonction
Branchement conditionnels (flags)	JC	Si retenue
	JE/JZ	Si égal/Si zéro
	JNC	Si pas de retenue
	JNE/JNZ	Si non égal / Non zéro
	JNO	Si pas de débordement
	JNP/JPO	Si pas de parité/ Si parité impaire
	JNS	Si pas de signe
	JO	Si débordement
	JP/JPE	Si parité / Si parité paire
	JS	Si signe (négatif)

Branchements inconditionnels

- **CALL label :** Appel d'une procédure (sous programme) qui commence à la ligne « label ». La position de l'instruction suivant le CALL est empilée pour assurer une poursuite correcte après l'exécution du sous programme.
- **RET Retour de sous programme**. L'exécution du programme continue à la position récupérée dans la pile.



- **JMP (Saut inconditionnel) :** Elle **transfert, sans condition**, la commande à l'emplacement de destination. L'opérande Cible peut être obtenu à partir de l'instruction elle même (JMP direct) ou à partir de la mémoire ou à partir d'un registre indiqué par l'instruction.

  Syntaxe : **JMP** cible

- **INT n :** appel à **l'interruption** logicielle n°: n

✓ Branchements conditionnels

- Les branchements conditionnels sont conditionnés par l'état des indicateurs (drapeaux « Flags ») qui sont eux même positionnés par les instructions inconditionnels.
- Dans la suite nous allons utiliser la terminologie :

➢ **supérieur** ou **inférieur** pour les nombres non signés

➢ **plus petit** ou **plus grand** pour les nombres signés

➢ **- +** pour l'opérateur logique OU

- **JE/JZ label (Jump if Equal or Zero)** Aller à la ligne label si résultat **nul** ou si **égalité**. C'est-à-dire si Z=1
- **JNE/JNZ :(Jump Si non égal / Non zéro)** Elle: **transfert, avec condition**, la commande à l'emplacement de destination.
  Si ZF=0 alors IP = IP + déplacement.

  Syntaxe : **JNZ** cible

Instructions boucles de répétition

Type	Nom	Fonction
Boucles	LOOP	Boucle
	LOOPE/LOOPZ	Boucle si égal/Si zéro
	LOOPNE/LOOPNZ	Boucle si différent/si diff 0
	JCXZ	Branchement si CX=0

- **LOOP :** L'instruction loop 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

Syntaxe: **LOOP** label

- **LOOPE** (whileEqual)**:** on reboucle tant que le **compteur CX différent de 0 et flag Z =1**
- **LOOPZ** (Loop While Zero) Décrémente le registre CX (aucun flag n'est positionné) on reboucle tant que **CX** est **différent de zéro** et le **flag Z** est égal à **1**. La condition supplémentaire sur Z, donne la possibilité de quitter une boucle avant que CX ne soit égal à zéro.
- **LOOPNE** (Loop While Not Equal)**, LOOPNZ** (Loop While Not Zero) Décrémente le registre CX et reboucle tant que **CX** est **différent de zéro** et **le flag Z est égal à 0**. Fonctionne de la même façon que loopz.
- **JCXZ label :** branchement à la ligne **label** si CX = 0 indépendamment de l'indicateur Z.

5.Génération du code machine

5.1 Etapes de la réalisation d'un programme :

1) Définir le problème à résoudre : que faut-il faire exactement ?

2) Déterminer des algorithmes, des organigrammes comment faire ? Par quoi commencer, puis poursuivre ?

3) Rédiger le programme (code source) :

- ✓ utilisation du jeu d'instructions (mnémoniques) ;
- ✓ création de documents explicatifs (documentation).

4) Tester le programme en réel ;

5) Corriger les erreurs (bugs) éventuelles : déboguer le programme puis refaire des tests jusqu'à obtention d'un programme fonctionnant de manière satisfaisante.

➢ **Langage machine et assembleur :**

✓ **Langage machine :** ***codes binaires*** correspondant aux instructions ;

✓ **Assembleur :** ***logiciel de traduction*** du code source écrit en langage assembleur (***mnémoniques***).

5.2. Réalisation pratique d'un programme :

1) **Rédaction du code source en assembleur à l'aide d'un éditeur** (logiciel de traitement de texte ASCII) :
   - ✓ Edit sous MS-DOS,
   - ✓ Notepad (bloc-note) sous Windows,
2) **Assemblage du code source** (traduction des instructions en codes binaires) avec un assembleur :
   - ✓ MASM de Microsoft,
   - ✓ **Emu 8086**, ...
3) Pour obtenir le **code objet** : code machine exécutable par le microprocesseur ;
   - ✓ Chargement en mémoire centrale et exécution : rôle du système d'exploitation (ordinateur)
4) **Pour la mise au point (débogage) du programme**, on peut utiliser un programme d'aide à la mise au point (comme DEBUG sous MS-DOS) permettant :

- ✓ l'exécution pas à pas;
- ✓ la visualisation du contenu des registres et de la mémoire ;
- ✓ la pose de points d'arrêt ...

5.3. Structure d'un fichier source en assembleur :

- ✓ Pour faciliter la lisibilité du code source en assembleur, on le rédige sous la forme suivante :

Labels instructions commentaires

label1 : mov ax, 60H ;ceci est un commentaire ...

...

...

sous prog1 **proc** near ; sous-programme

...

...

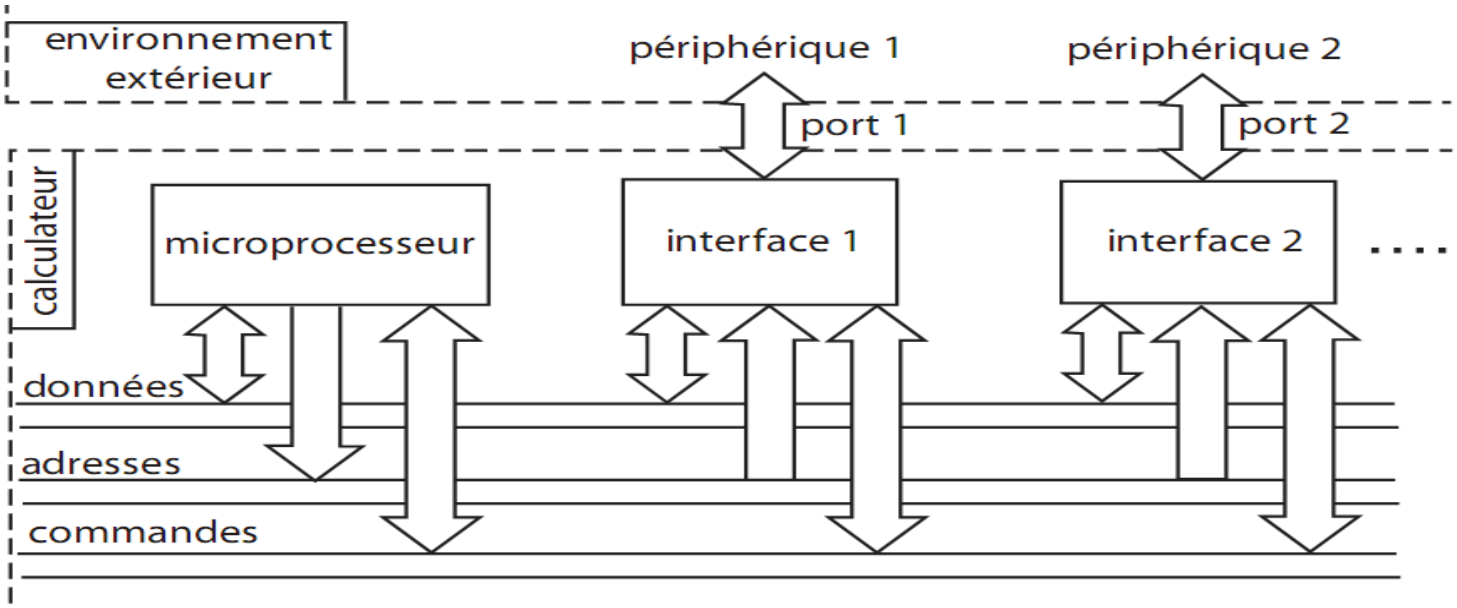
sous prog1 **endp**

...

...

6. Gestion des entrées/sorties par le 8086

- Une **interface d'entrées/sorties** est un **circuit intégré** permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante, modem, disques, processus industriel, ...
- Les interfaces d'E/S sont **connectées** au microprocesseur à travers **les bus d'adresses**, de **données et de commandes**. Les jonctions entre les interfaces et les périphériques sont appelés **port**.
- **Le 8086** dispose d'un espace mémoire de **1 Mo (adresse d'une case mémoire sur 20 bits)** et d'un espace d'E/S de **64 Ko (adresse d'un port d'E/S sur 16 bits)**.



- Exemple :

interface	port	exemple de périphérique
interface parallèle	port parallèle	imprimante
interface série	port série	modem

6.1.Instructions d'accès aux ports d'E/S

- Les instructions de lecture et d'écriture d'un port d'E/S sont respectivement les instructions **IN** et **OUT**.

➢ **Lecture d'un port d'E/S :**

❑ Si l'adresse du port d'E/S est sur **un octet** :

✓ IN al, adresse : lecture d'un port sur 8 bits

✓ IN ax, adresse : lecture d'un port sur 16 bits

❑ Si l'adresse du port d'E/S est sur **deux octets** :

✓ IN al, dx: lecture d'un port sur 8 bits

✓ IN ax, dx: lecture d'un port sur 16 bits

- Où le registre DX contient du port d'E/S à lire.

Exemple:

Lecture d'un port d'E/S sur 8 bits à l'adresse 4221h :

```
mov dx, 4221h
IN al, dx
```

➢ Ecriture d'un port d'E/S :

- ❑ Si l'adresse du port d'E/S est sur **un octet** :
- ✓ OUT adresse, al : Ecriture d'un port sur 8 bits
- ✓ OUT adresse, ax : Ecriture d'un port sur 16 bits
- ❑ Si l'adresse du port d'E/S est sur **deux octets** :
- ✓ OUT dx, al: Ecriture d'un port sur 8 bits
- ✓ OUT dx, ax : Ecriture d'un port sur 16 bits
- Où le registre DX contient du port d'E/S à lire

- **Exemple:**
- Ecriture de la valeur AF37 H dans le port d'E/S sur 16 bits à l'adresse 52 h :

```
Mov ax, AF37H
OUT 52 h, ax
```

7. Gestion de la pile

- Une pile est une zone mémoire servant à stocker **temporairement** des valeurs.
- On ne peut stocker qu'une information à la fois et l'élément dépilé à un moment donné est celui qui a été empilé en dernier: c'est la structure **LIFO (Last In, First Out).**

- Les opérations ne se font que sur **16bits**.
- La pile commence au **segment SS** et elle finit dans le même segment mais à **l'offset SP**.
- A noter que cette pile est remplie à l'envers: *le premier élément est situé à une adresse plus haute que le dernier élément*
- Les données **ne sont pas effacées** après un POP
- Un PUSH **écrase** les données précédemment dans la pile

- ***7.1. L'instruction d'empilement (PUSH)***
- L'empilement d'une donnée est réalisé par l'instruction: **PUSH donnée**
- Où donnée est un registre **16bits** ou un emplacement mémoire désigné par adressage *direct* ou *indirect*.
- **L'empilement** s'effectue en deux étapes:

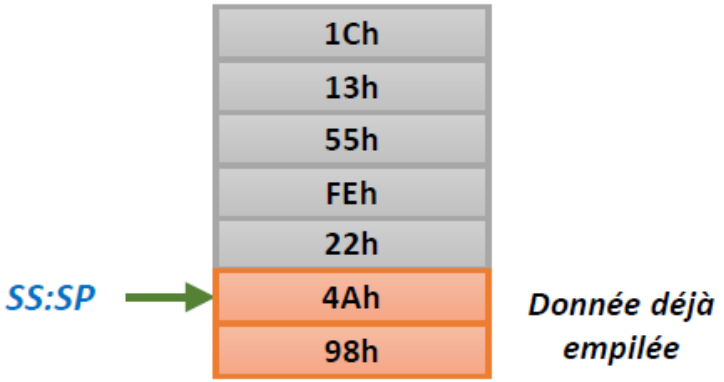
✓ *SP←SP- 2*: on ***fait reculer SP de deux cases*** (donnée sur16bits) ,pour qu'il pointe sur le nouveau sommet susceptible de recevoir la ***donnée à empiler***.

- *[SP]←donnée:* ***le sommet de la pile*** reçoit la donnée.

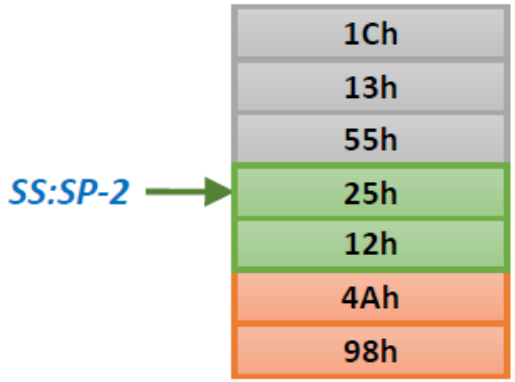
- Exemple:

Soit AX=1225h et soit l'instruction: PUSH AX

- *Avant l'instruction :*



- *Après l'instruction :*



7.2. L'instruction de dépilement (POP)

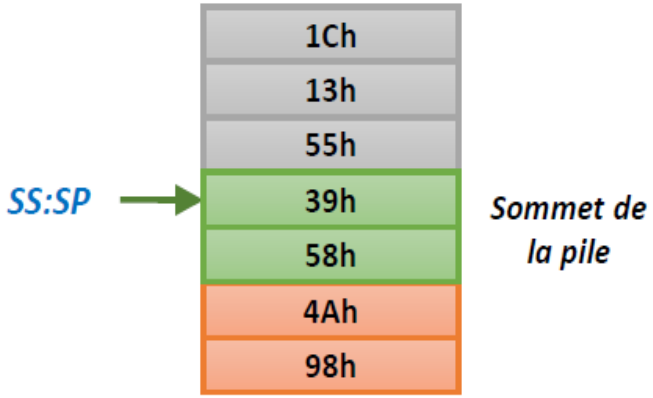
- Le dépilement est l'opération qui permet de **retirer le sommet** de la pile et le **stocker dans une destination.**
- Il est réalisé par l'instruction: **POP destination**
- Destination est un registre **16 bits** ou un emplacement mémoire désigné par adressage direct ou indirect.
- **Le dépilement** engendre deux opérations:

✓ destination←[SP]: ***le sommet*** de la pile ***est copié dans destination***.

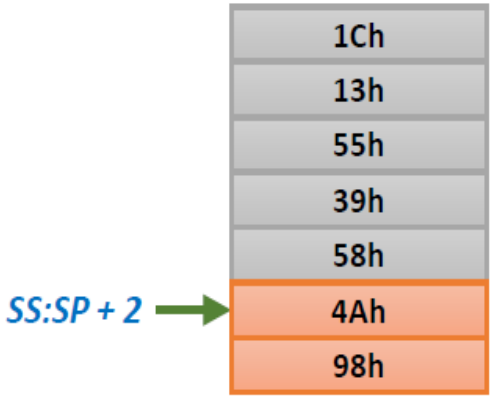
- SP←SP+2: ***la donnée du sommet étant retirée***, on fait ***avancer SP*** pour ***pointer sur la donnée suivante.***

- Exemple:
  - Considérons l'instruction: **POP DX**

- *Avant l'instruction :*



- *Après l'instruction :*



- *Le registre DX est chargé par la valeur 5839h*

7.3. Autres instructions de pile

- ✓ **PUSHA**(PushAll): Empile les 8 registres généraux:
  - Les registres sont empilés dans l'ordre AX, CX, DX, BX, SP, BP, SI, DI.
  - La valeur empilée de SP est celle avant le début de l'instruction.
- ✓ **POPA** (PopAll): Dépile **les16 octets** du sommet de la pile **vers** les 8 registres généraux:
  - Les registres sont dépilés dans l'ordre DI, SI, BP, SP, BX, DX, CX, AX.
  - En fait, le registre SP pose un cas particulier, puisqu'il n'est pas mis à jour avec la valeur dépilée qui n'est pas prise en compte.
- ✓ **PUSHF** (Push Flags): Empile le registre des indicateurs.
- ✓ **POPF** (Pop Flags): Dépile vers le registre des indicateurs.

8. Fonctions et sous-programmes

- Un **Sous-Programme (procédure):** une séquence d'instructions qui possède un nom unique et fait l'appel à l'intérieur d'un **programme principal** pour éviter la répétition des instructions et engendre un accès à la pile **(mémorisation de l'adresse de retour)**, le branchement se fait lors de l'appel.
- **Instructions de sauvegarde** : si le Sous Programme utilise des registres du processeur, il est important de sauvegarder ces registres sur la pile au début du **SP (Optimisation de la place mémoire)**

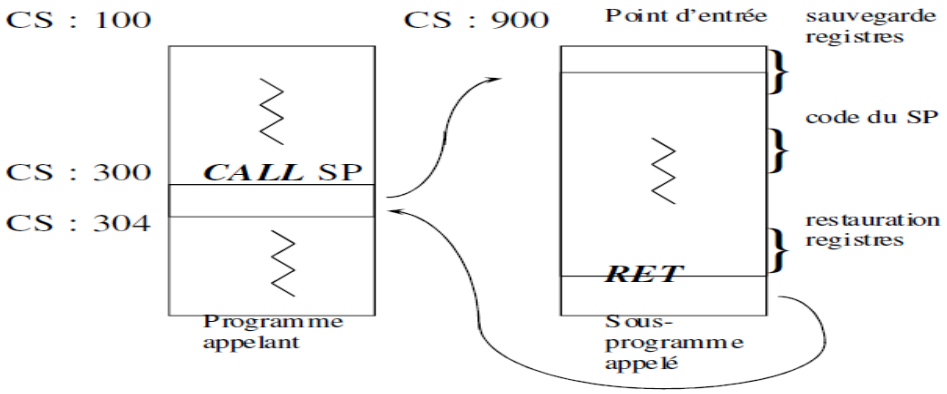
Ecriture d'un sous-programme :

```
nom_sp    PROC
            :   ← instructions du sous-programme
          ret   ← instruction de retour au programme principal
nom_sp    ENDP
```

Appel d'un sous-programme par le programme principal : CALL procédure

```
            :   ← instructions précédant l'appel au sous-programme
  call nom_sp   ← appel au sous-programme
            :   ← instructions exécutées après le retour au programme principal
```

- ✓ Lors de l'exécution de l'instruction **CALL**, le pointeur d'instruction **IP** chargé avec l'adresse de la **première instruction** du sous-programme.
- ✓ Lors du retour au programme appelant, l'instruction suivant le **CALL** doit être exécutée, c'est-à-dire que **IP** doit être rechargé avec l'adresse de cette instruction (l'instruction suivante).
- ✓ Avant de charger **IP** avec l'adresse du sous-programme, **l'adresse de retour** au programme principal, c'est-`a-dire le **contenu de IP**, est **sauvegardée** dans une zone mémoire particulière appelée **pile**.
- ✓ Lors de l'exécution de l'instruction **RET**, cette adresse est **récupérée** à partir de la **pile** et **rechargée dans IP**, ainsi le programme appelant peut se poursuivre.



- ✓ **Remarque :** la **valeur de SP** doit être **initialisée par le programme principal** avant de pouvoir utiliser la pile

- **Classification des paramètres :** Les paramètres peuvent être **en entrée, en sortie ou en entrée-sortie**, ils sont classés : **par valeur, par référence**
- **Passage de paramètres par valeur :**
  - *Recopie* de *la valeur* à transmettre dans une *zone mémoire* connue du sous-programme
  - Travaille sur *une copie de la variable* => la variable du Programme principal **ne varie pas** (protection)

- Exemple

Prog. Principal

```
• • •
MOV   AX, 4 }
MOV   BX, 5 }  Paramètres
call Somme  → Appel
• • •
-> Résultat dans AX
```

Sous-Prog.

```
Somme PROC NEAR
      ADD   AX, BX
      RET
Somme ENDP
```

✓ Passage par référence : transmission de l'adresse de la variable

- Plus rapide
- Le sous programme **travaille sur les variables du Programme principal** => ces variables ont été modifiées par le sous programme.
- Les variables **ne seront pas protégés** dans ce cas.

- Exemple

Prog. Principal

```
•••
MOV   CX, 10 ; taille de TAB
LEA   BX, TAB  ; BX<- @TAB
call Affiche
•••
```

Sous-Prog.

```
Affiche PROC NEAR
        MOV DI, 0
boucle :
        MOV AL,[BX+DI]
        call COUT
        INC DI
LOOP boucle
        RET
Affiche ENDP
```

9. Interruptions

- Pour faire des **entrées/sorties** (essentiellement avec l'écran et le clavier), on passe par des interruptions du **BIOS** ou du **DOS**. Nous n'allons voir ici que ce dont nous avons besoin.

9.1 L'interruption 10h du BIOS

- Le BIOS est de relativement bas niveau et **dépend fortement de la machine**.
- **L'interruption 10h** peut effectuer beaucoup de fonctions différentes, le numéro de la fonction désirée doit être placé dans **AH** avant l'appel de l'interruption. Nous ne parlerons ici que de quelques fonctions :

✓ Fonction 00

- Cette fonction permet de choisir un mode texte ou un mode graphique. En changeant de mode, on peut effacer l'écran, ce qu fait que l'on peut appeler cette fonction pour effacer l'écran et rester dans le même mode. Paramètres : AH = 00

✓ Fonction 09

- Cette fonction permet d'écrire un caractère : Permet les répétitions, Gère la couleur en mode texte et en mode graphique. Paramètres : AH = 09h

✓ Fonction 0Eh

- Cette fonction permet d'écrire un caractère : Fonctionne en mode graphique, Gère le curseur, Gère la couleur seulement e mode graphique. Seule la couleur du caractère est gérée, la couleur du fond n'est pas gérée. Paramètres : AH = 0Eh

✓ Fonction 05

- Cette fonction permet de sélectionner la page active de l'affichage.

Paramètres : AH = 05h

AL = numéro de la page

9.2. L'interruption 21h du DOS

- Appel au BIOS qui fonctionne à un niveau plus proche de la machine. **L'interruption 21h** peut réaliser plusieurs fonctions différentes. Nous ne citerons ici que celles que nous utiliserons :

✓ **Fonction 02**

- Cette fonction permet d'écrire un caractère. Le caractère est envoyé vers la sortie standard, l'écriture peut donc être redirigée dans un fichier.
- Paramètres : AH = 02h DL = Caractère à écrire

✓ **Fonction 09**

- Cette fonction permet en un seul appel, d'écrire une suite de caractères. Paramètres : AH = 09h

  DX = Adresse de la chaîne de caractères ; La chaîne doit être terminée par le caractère $

✓ **Fonction 0Ah**

- Permet de saisir une chaîne de caractère au clavier. La saisie s'arrête quand on tape la touche de retour, le caractère CR (13) est mémorisé avec la chaîne

Quelques interruptions									
		AH	AL	BH	BL	CX	DH	DL	commentaire
INT 10h	Mode écran	00	mode						
	Ecrit char	09	car	page	coul t/g	rep			!Curs !CarSpec
	Ecrt char	0A	car	page		rep			!Curs !CarSpec
	Ecrt char	0E	car		Coul g				Curs CarSpec
	Pos Curs	02		page			ligne	col	
	Choisir page	05	page						
	Allumer pixel	0Ch	couleur	page		x	y		
	50 lignes	11h	12h		30h				
INT 21h	Ecrt char	02						car	^C^B -> int 23h
	Ecrit chaîne	09					adresse		'$'= fin chaîne
	Lit car !écho	07	Car lu						
	Lit chaine	0Ah					adresse		Voir cours
	kbhit	0B	0 : non FF : oui						