Chapitre 4:

Le Microprocesseur 80x86

Filière : **2ème Année préparatoire**

Présenter par : MILI MANEL

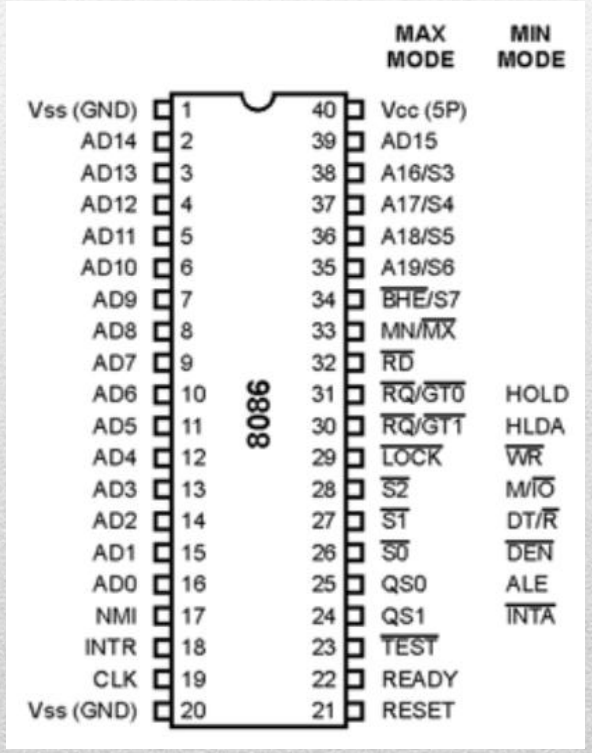
Email: manel.mili@issatso.u-sousse.tn

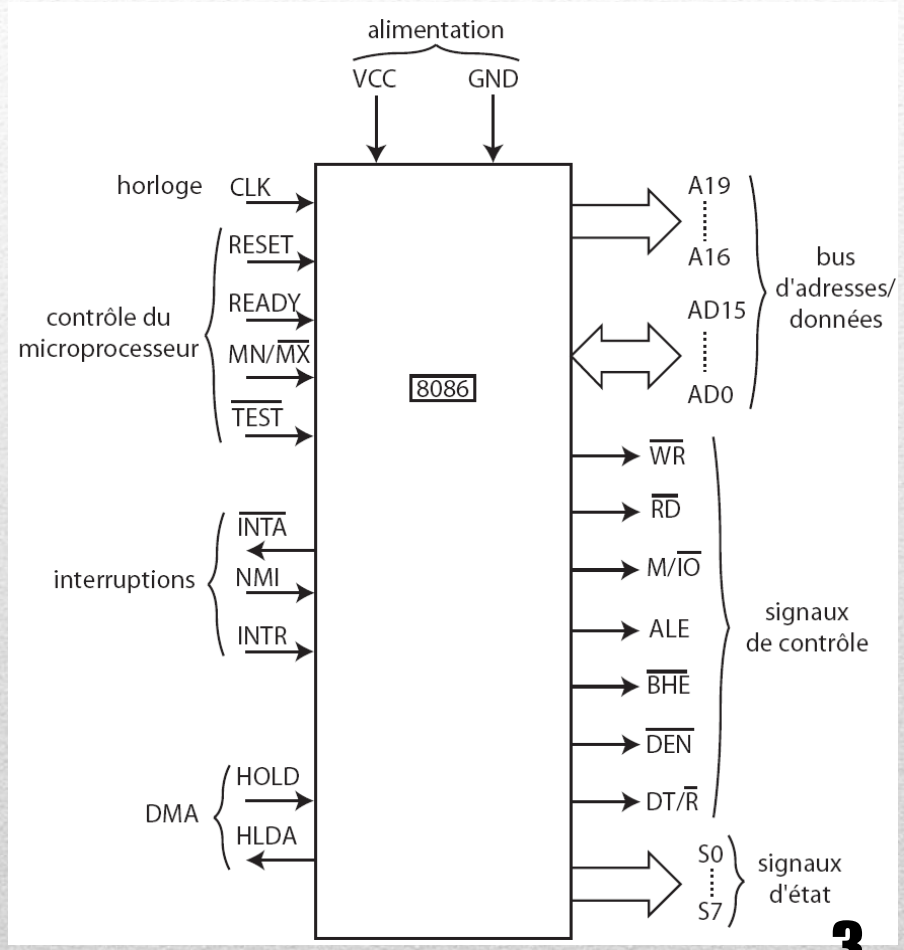
Chapitre 4: le Microprocesseur 80x86

1. Description 80x86
2. Gestion et segmentation de la Mémoire
3. Les Registres du 80x86
4. Jeux d'instructions 80x86
5. Les sous programmes

2

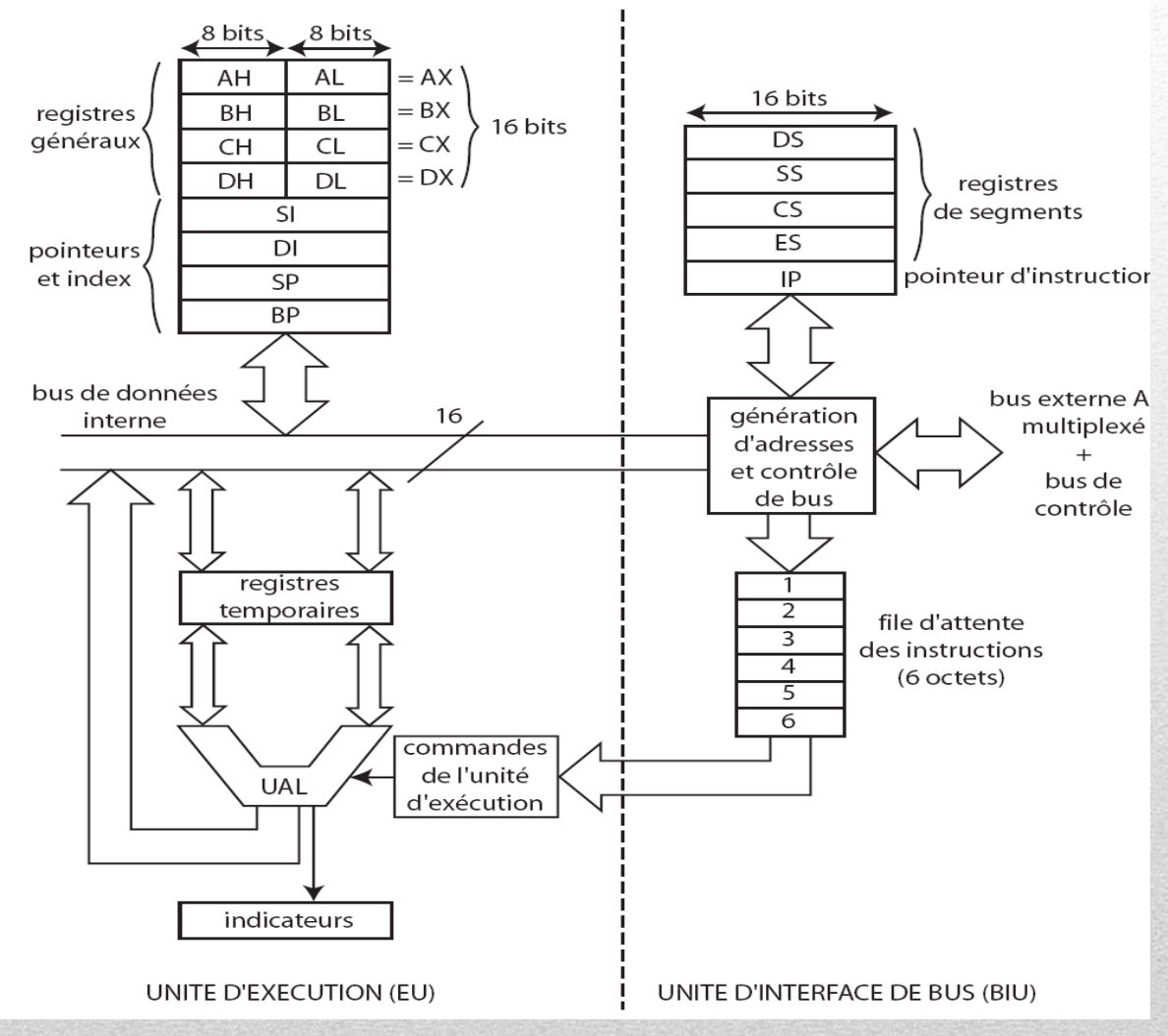

D8086
12290019
INTEL '78 '79


MAX MODE
MIN MODE
8086
Vss (GND) 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
Vss (GND) 20
40 Vcc (5P)
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GT0 HOLD
30 RQ/GT1 HLDA
29 LOCK WR
28 S2 M/IO
27 S1 DT/R
26 S0 DEN
25 QS0 ALE
24 QS1 INTA
23 TEST
22 READY
21 RESET


alimentation
VCC
GND
horloge
CLK
RESET
READY
MN/MX
TEST
contrôle du microprocesseur
INTA
NMI
INTR
interruptions
HOLD
HLDA
DMA
8086
A19
A16
AD15
AD0
bus d'adresses/ données
WR
RD
M/IO
ALE
BHE
DEN
DT/R
signaux de contrôle
S0
S7
signaux d'état

3

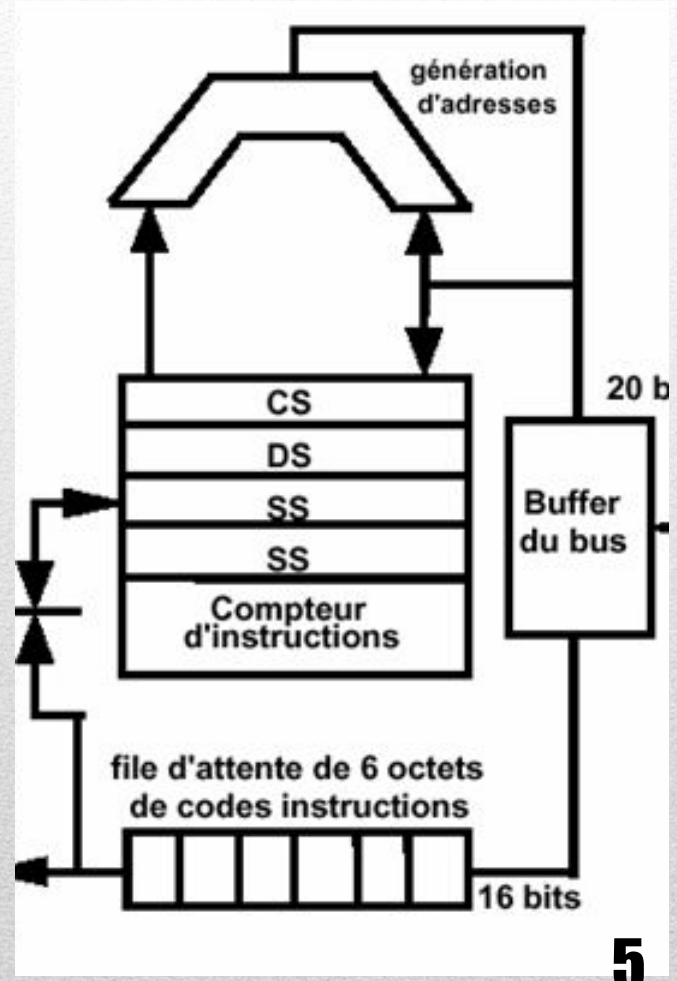
Le 8086 est constitué de deux unités fonctionnant en parallèle :



4

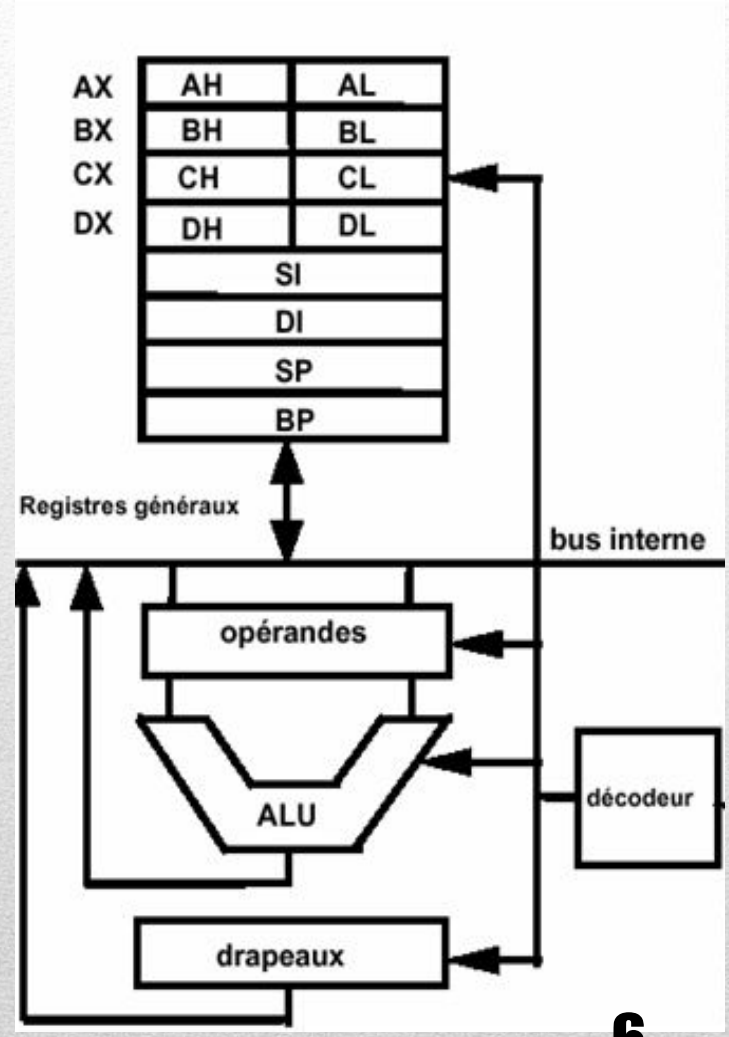
L’unité UIB (Unité d'Interface de Bus) comporte essentiellement :

- Une file d’attente d’instructions générée en FIFO (First In First Out)
- Les registres de segments
- Compteur d’instructions (IP) ou bien PC



L’ ’UE (unité d’exécution comporte) :

- Les registres généraux et les registres d’adressages
- Le registre d’état (Flags)
- Registre de données (accumulateurs)
- L’UAL
- Décodeur d’instructions



6

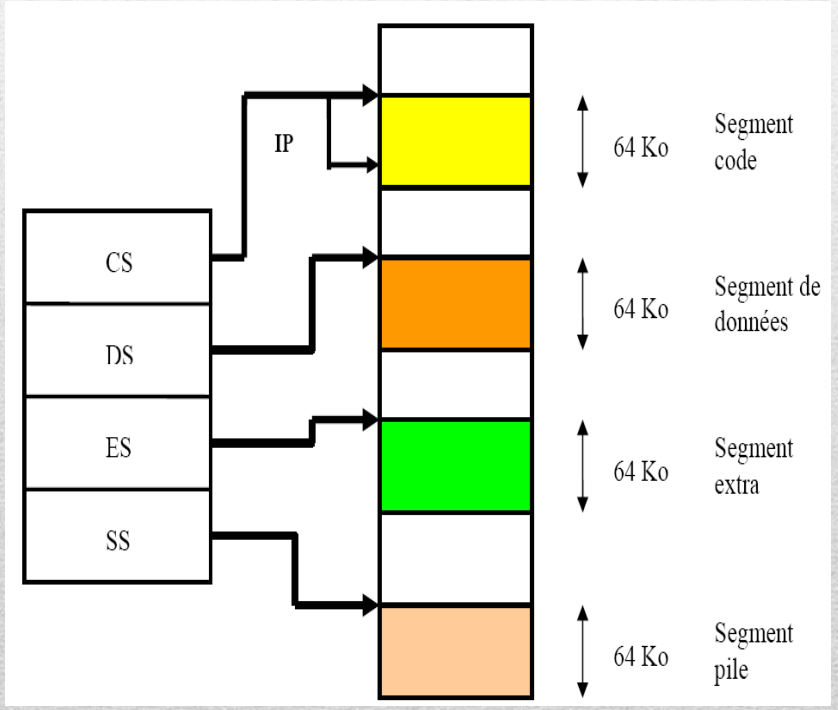
Chapitre 4: le Microprocesseur 80x86

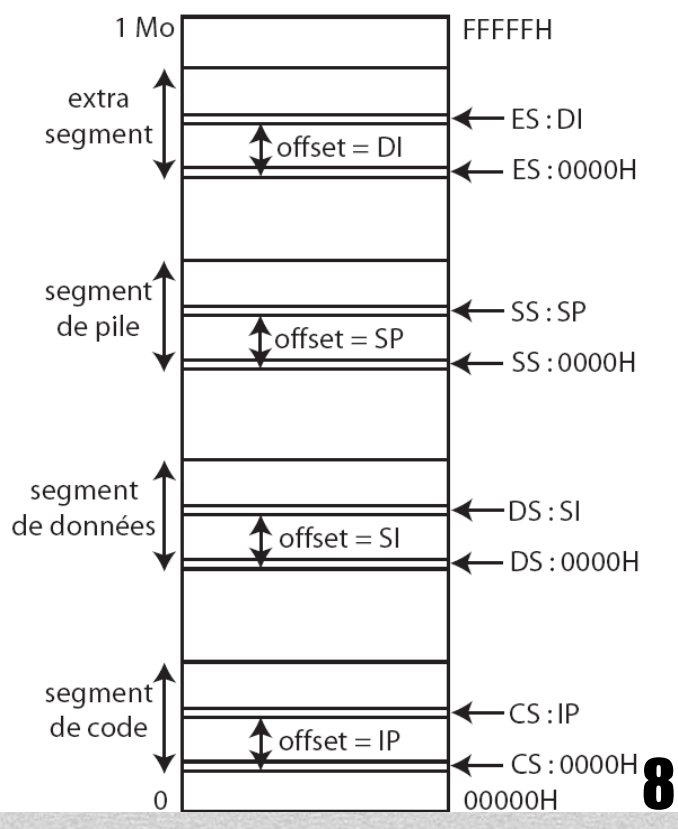
1. Description 80x86
2. Gestion et segmentation de la Mémoire
3. Les Registres du 80x86
4. Jeux d'instructions 80x86
5. Les sous programmes

7

Une case mémoire est repérée par le 8086 au moyen de deux registres:

- **L'adresse de base** d'un segment
- **Un déplacement ou offset** (appelé aussi adresse effective) dans ce segment.





- une **adresse logique** : donnée par le couple (segment, offset) notée sous la forme

segment : offset

- Une **adresse physique** : L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexadécimaux). Elle correspond à la valeur envoyée réellement sur le bus d'adresses.

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**

=> Adresse physique = (16 × segment) + offset

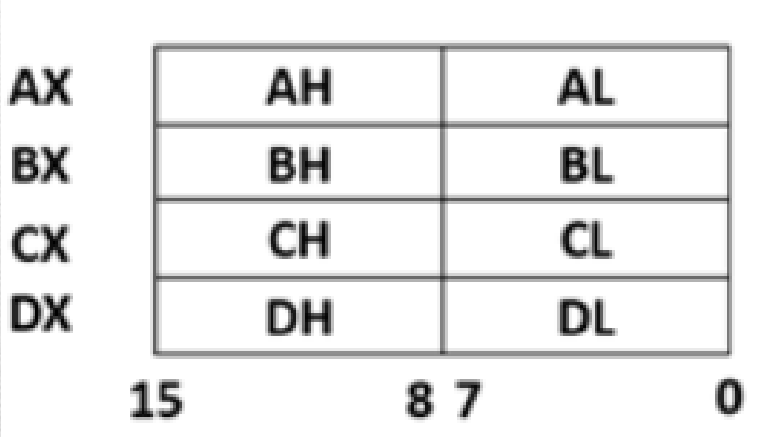
9

Chapitre 4: le Microprocesseur 80x86

1. Description 80x86
2. Gestion et segmentation de la Mémoire
3. Les Registres du 80x86
4. Jeux d'instructions 80x86
5. Les sous programmes

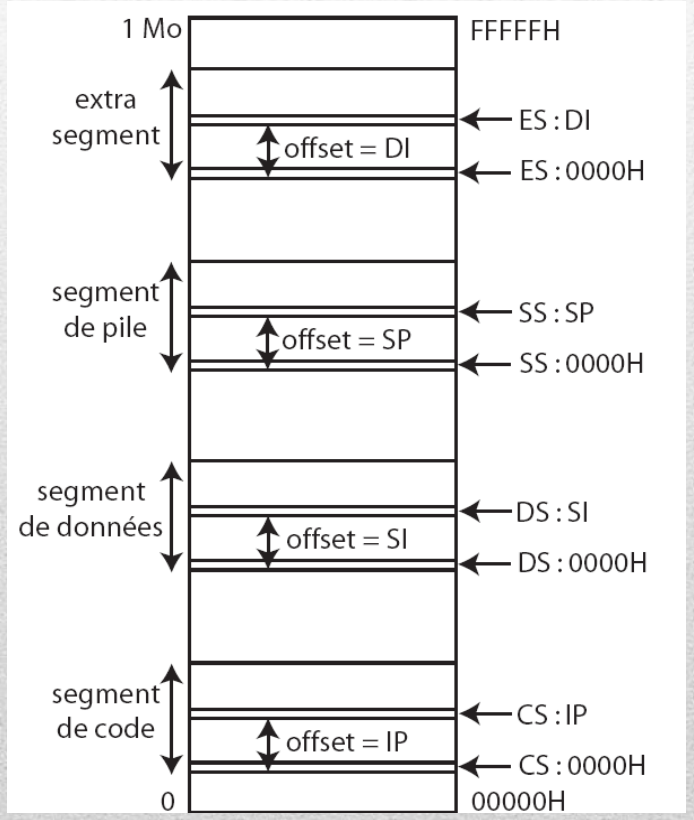
10

- **Registres généraux :** 4 registres sur 16 bits.

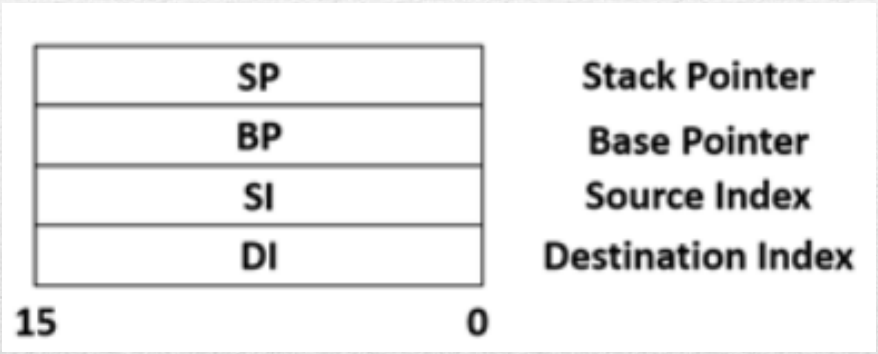


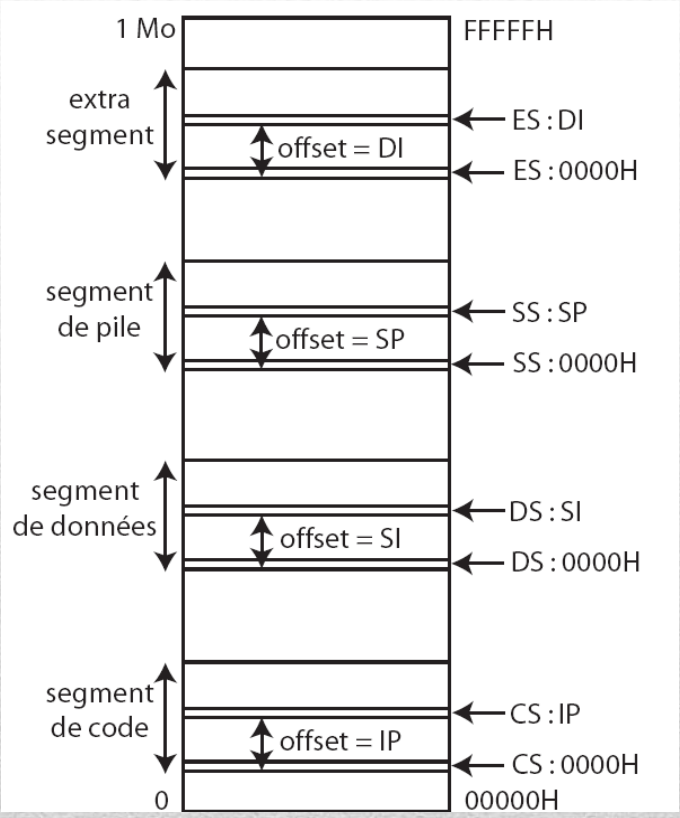
- Le 8086 a 4 registres segments de 16 bits chacun :

✓CS (code segment)
✓DS (Data segment)
✓ES (Extra segment)
✓SS (Stack segment)



- Il existe 4 registres d'adressage de taille 16 bits (SI, DI, SP , BP).

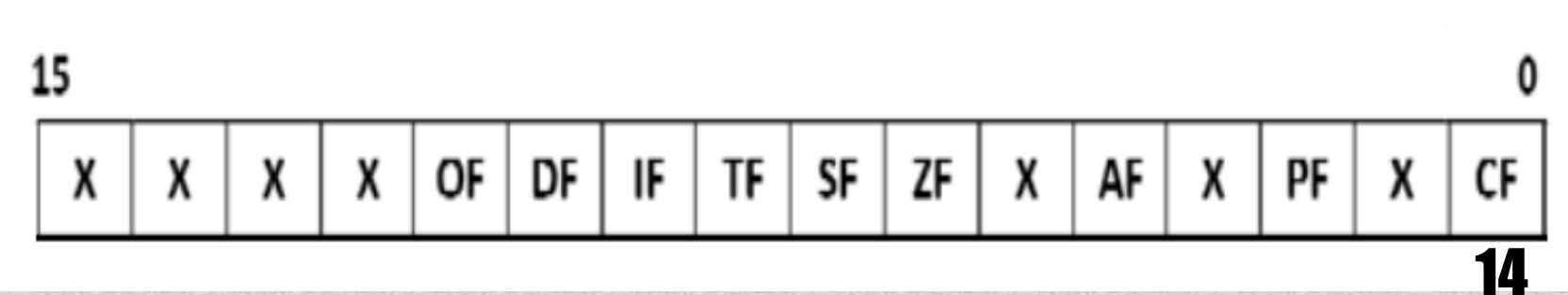




- 2 registres de 16 bits : un pointeur d'instruction et un indicateur.

✓ **Le registre IP** (Pointeur d'instruction / Le compteur de programme): Il contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter. Le registre IP est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

✓ **Le registre d'état (indicateur – Flag):** sert à contenir l'état de certaines opérations effectuées par le processeur. Le registre d'état du 8086 est formé par les bits suivants :

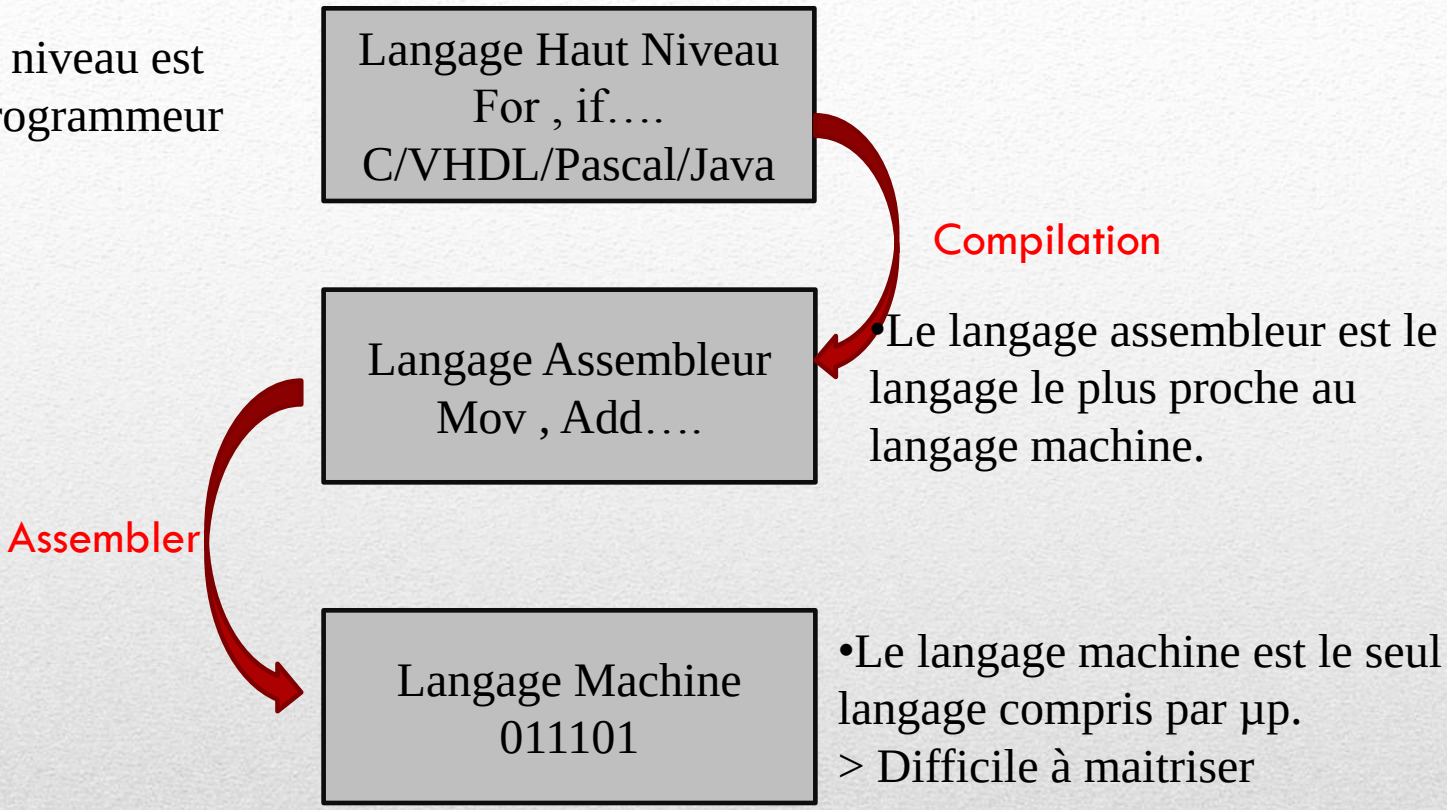


Chapitre 4: le Microprocesseur 80x86

1. Description 80x86
2. Gestion et segmentation de la Mémoire
3. Les Registres du 80x86
4. Jeux d'instructions 80x86
5. Les sous programmes

15

•Le langage haut niveau est plus adapté au programmeur



•Le langage assembleur est le langage le plus proche au langage machine.

•Le langage machine est le seul langage compris par µp.
> Difficile à maitriser

>chaque processeur posséde un assembleur différent. (opérations assez simple)

```
X<=X-1;
```

compilateur ARM

```
LDR R4,@X
SUB R4,R4,#1
STR @X, R4
```

Structure d'un programme en assembleur :

```
TITLE nomprogramme ; cette directive permet de nommer votre programme

Pile SEGMENT STACK; segment de pile dont le nom est Pile
; déclarer la pile et sa taille
Pile ENDS

Donnees SEGMENT; voici le segment de données
; Placez ici les déclarations de données
Donnees ENDS

code SEGMENT ; voici le segment d'instructions
ASSUME DS:donnee, CS: code
debut:
mov ax, donnee
 mov ds, ax
; placez vos instructions…
code ENDS
END debut; fin du programme suivie de l'étiquette de la première instruction
```

Variables

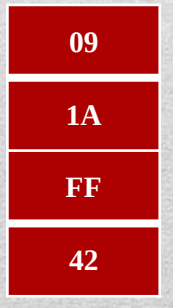
En 8086 la base de représentation :

- ✓ Hexa : commence par un chiffre et se termine par un H
- ✓ Binaire : commence par un chiffre et se termine par un b.
- ✓ Décimal : que des chiffres
- Directive pour 8086:
- ✓ DB (Define Byte): variable 8 bits
- ✓ DW(Define Word): variable 16 bits
- ✓ EQU: constante

```
A DW 1234H
B DB 126
M DB "salut $"

TAB DB 20 dup (?)  ; Réservation de 20 éléments (octets) en mémoire non initialisés

DELTA DB 9,1AH,-1,'B' ; suite d'éléments dans un  tableau
```



1. Les instructions de transfert

Elles permettent de déplacer des données d'une **source** vers une **destination** :

- Registre vers mémoire ;
- Registre vers registre ;
- Mémoire vers registre.

Syntaxe : MOV destination, source

2. Les instructions arithmétiques

- **Syntaxe: ADD Opérande 1 , Opérande 2**

Opérande 1 ← Opérande 1 + Opérande 2

Exemples :

```
ADD AX,123 ;

ADD AX,BX ;

ADD [123],AX ;
```

- **Syntaxe: SUB Opérande 1 , Opérande 2**

Opérande 1 ← Opérande 1 – Opérande 2

- **Syntaxe: MUL Opérande**

Instruction à un seul opérande. Elle effectue une multiplication non signée entre l'accumulateur (AL ou AX) et l'opérande Op. Le résultat de taille double est stocké dans l'accumulateur et son extension (AH:AL ou DX:AX)

MUL Op_8	AX ← AL x Op_8
MUL Op_{16}	DX:AX ← AX x Op_{16}

- L'opérande Op ne peut pas être une donnée, c'est soit un registre soit une position mémoire, dans ce dernier cas, il faut préciser la taille (byte ou word)

Multiplication	Opérande 1	Opérande 2	Résultat
Octet x Octet	AL	Registre ou memoire	AX
Mots x Mots	AX	Registre ou memoire	DX AX
Mots x Octet	AL= Octet, AH=0	Registre ou memoire	DX AX

Exemples :

```
mov al,51
mov bl,8
mul bl
```

<=> AX ← AL x BL

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

<=> DX:AX ← AX x CX

- **Syntaxe: DIV Opérande**

Effectue la division AX/Op_8 ou $(DX|AX)/Op_{16}$ selon la taille de Op qui doit être soit un registre soit une mémoire. Dans le dernier cas il faut préciser la taille de l'opérande, exemple : DIV byte [adresse] ou DIV word [adresse].

Opération sur 8 bits	Opération sur 16 bits
DIV Op_8 ; AX /Op_8 AL ← Quotient AH ← Reste AX | Op_8 AH | AL	DIV Op_{16} ; DX:AX /Op_{16} AX ← Quotient DX ← Reste DX:AX | Op_{16} DX | AX

3. Les instructions logiques

- **Syntaxe: AND Od , Os**

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

Exemple :

```
MOV AX , 503H ; AX = 0000 0101 0000 0011
AND AX , 0201H;
```

```
      0000 0101 0000 0011
AND   0000 0010 0000 0001
----------------------------
=     0000 0000 0000 0001
```

- Il y a deux types de décalage :
  - Les décalages logiques (opérations non signées)
  - Les décalages arithmétiques (opérations signées).

Les décalages logiques (opérations non signées): SHR et SHL

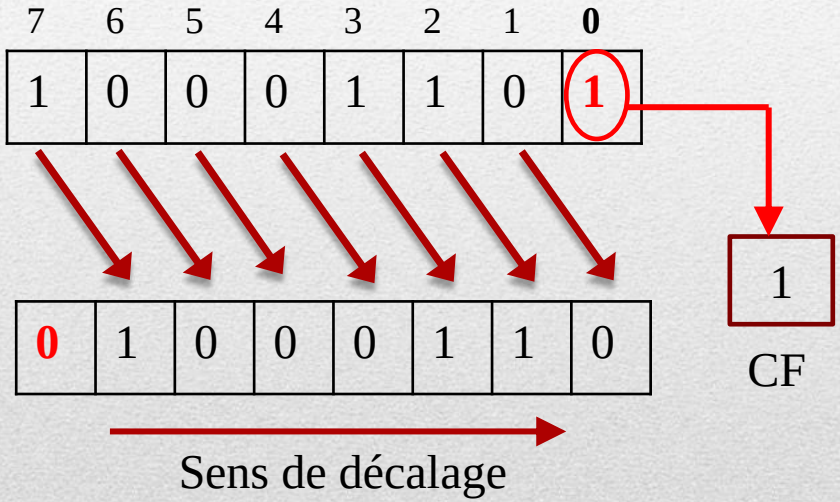
- **Syntaxe: SHR Opérande , n**

  (SHift logical Right) : Cette instruction décale l'opérande de n positions vers la droite.

Exemple:

```
MOV AL,10001101B
SHR AL,1
```

- Entrée d'un ***0 à la place du bit de poids fort*** ; le bit sortant passe à travers l'indicateur de retenue **CF.**



Exemple : Décalage de AL de quatre positions vers la droite :

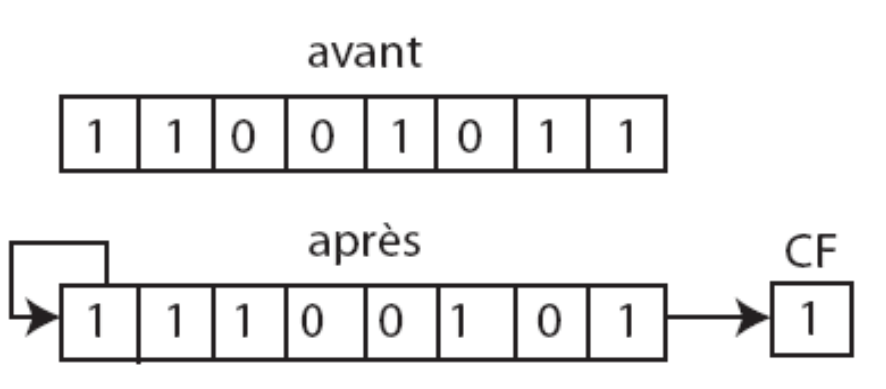
```
MOV CL,4
SHR AL, CL
```

Les décalages arithmétiques (opérations signées): SAR et SAL

- **SAR op, n**

  Ce décalage conserve le bit de signe bien que celui-ci soit décalé.
  le bit de signe est réinjecté.

Exemple :
mov al,11001011B
sar al,1



Les décalages arithmétiques permettent de conserver le signe. Ils sont utilisés pour effectuer des opérations arithmétiques comme des multiplications et des divisions par 2.

Les instructions de Rotations: ROR et ROL

- **Syntaxe: ROR Opérande , n**

  Rotation à droite (Rotate Right) : Cette instruction décale l'opérande de n positions vers la droite et réinjecte par la gauche les bits sortant.

Exemple :

```
mov al,11001011B
ror al,1
```

→Réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

4. Les instructions de branchement

Instruction de saut inconditionnel :

- **Syntaxe: JMP etiquette**

↳ se brancher" inconditionnellement à l'instruction marquée par label (etiquette)

- **Syntaxe: INT n**

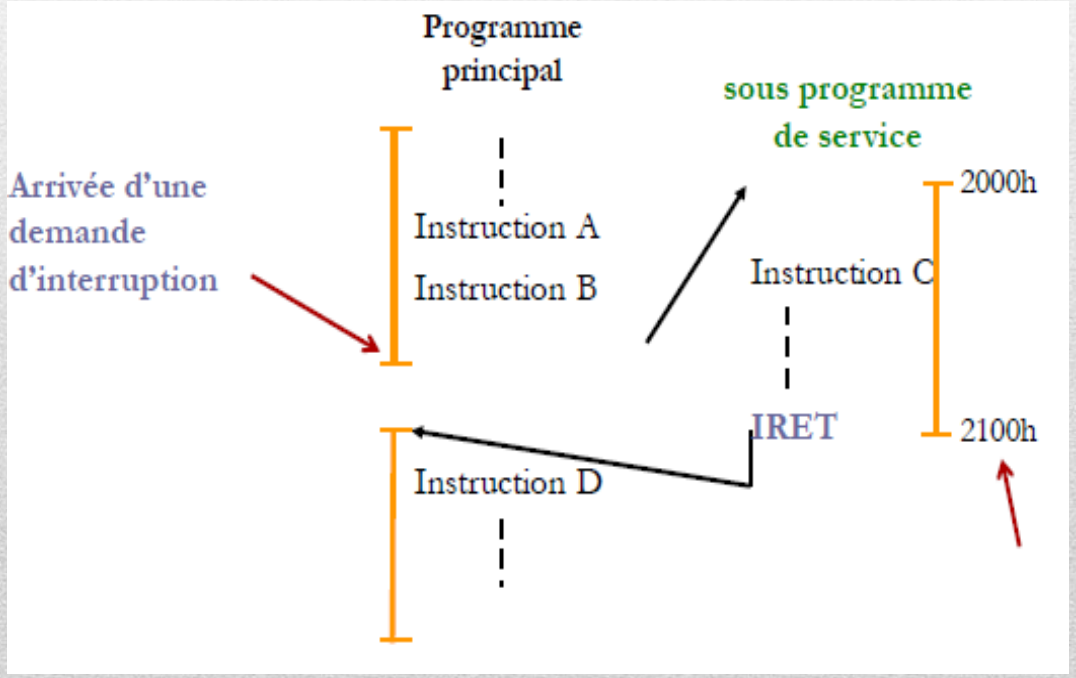
↳ appel à l'interruption logicielle n° n

L'interruption 21h du DOS

L'interruption 21h peut réaliser plusieurs fonctions DOS différentes. Nous ne citerons ici quelques exemples :

⇨ La valeur du registre AH permet d'indiquer quelle est la fonction que l'on appelle : `MOV AH, numero_fonction INT 21H`

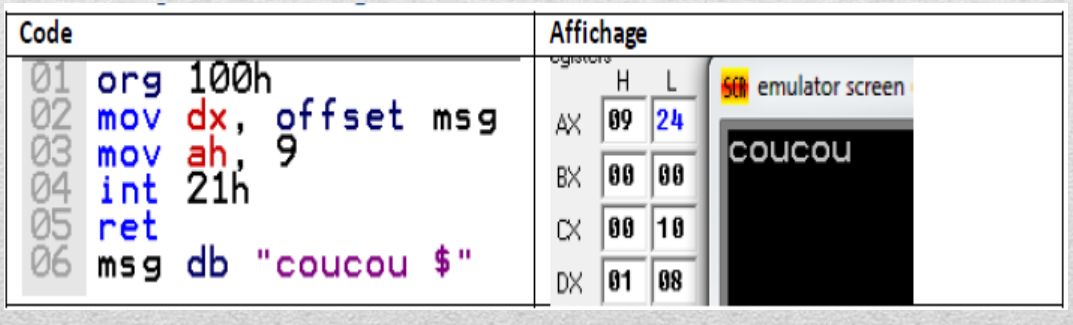
Une interruption est un évènement qui provoque l'arrêt du programme en cours et provoque le branchement du microprocesseur à un sous- programme particulier dit de "traitement de l'interruption".



Numéro Fonction

.

01H	Lecture caractère met le code ascii lu dans AL
02H	Affiche caractère code ascii dans registre DL
09H	Affiche chaîne de car DX=adresse début chaîne, terminée par '$‘
0BH	Lit état clavier met AL=1 si caractère, 0 sinon



Offset indique l’adresse mémoire du premier élément de **Dx** et le caractère **‘$’** indique la fin de la chaine.

Exemple :

Ce programme lit un caractère au clavier et l'affiche en majuscule :

```
MOV AH, 01H          ; code fonction DOS
INT 21H              ; attente et lecture d'un caractère

SUB AL, 20H    ; passe en majuscule
MOV DL, AL ;

MOV AH, 02H          ; code fonction affichage
INT 21H              ; affiche le caractère
```

Instruction de saut conditionnel :

Un saut conditionnel n’est exécuté que si une certaine condition est satisfaite, sinon l’exécution se poursuit séquentiellement à l’instruction suivante.
Ils sont généralement employés pour prendre une décision suivant les drapeaux, qui sont mis à jour par les instructions arithmétiques, logiques et de comparaison.
Il existe deux catégories :

-sauts sur les drapeaux
-sauts arithmétiques

- **Sauts sur les drapeaux**

➢ La condition du saut porte sur l'état de l'un (ou plusieurs) des indicateurs d'état flags du microprocesseur :

instruction	nom	condition
`JZ label`	Jump if Zero	saut si ZF = 1
`JNZ label`	Jump if Not Zero	saut si ZF = 0
`JE label`	Jump if Equal	saut si ZF = 1
`JNE label`	Jump if Not Equal	saut si ZF = 0
`JC label`	Jump if Carry	saut si CF = 1
`JNC label`	Jump if Not Carry	saut si CF = 0
`JS label`	Jump if Sign	saut si SF = 1
`JNS label`	Jump if Not Sign	saut si SF = 0
`JO label`	Jump if Overflow	saut si OF = 1
`JNO label`	Jump if Not Overflow	saut si OF = 0
`JP label`	Jump if Parity	saut si PF = 1
`JNP label`	Jump if Not Parity	saut si PF = 0

- **Sauts arithmétiques**

- Ils suivent en général l'instruction de comparaison :

CMP opérande1, opérande2

condition	nombres signés	nombres non signés
=	JEQ label	JEQ label
>	JG label	JA label
<	JL label	JB label
≠	JNE label	JNE label

Les instructions de contrôle de Boucle

- **Instruction LOOP**

"se brancher" à l'instruction marquée par label: tant que CX#0

Syntaxe: **MOV** CX,178

label:**...**

...

LOOP label

Exemple:
on veut additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul.

```
            mov     al,[1100H]
            add     al,[1101H]
            js      negatif
            jz      nul
            mov     [1102H],al
            jmp     fin
negatif :   mov     [1103H],al
            jmp     fin
nul :       mov     [1104H],al
fin :       hlt
```

Chapitre 4: le Microprocesseur 80x86

1. Description 80x86
2. Gestion et segmentation de la Mémoire
3. Les Registres du 80x86
4. Jeux d'instructions 80x86
5. Les sous programmes

37

- Pour éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme, on rédige la séquence une seule fois en lui attribuant un nom (au choix) et on l'appelle lorsqu'on en a besoin.
- Le programme appelant est le programme principal. La séquence appelée est un sous-programme ou procédure.

Remarque : une procédure peut être de type NEAR si elle se trouve dans le même segment ou de type FAR si elle se trouve dans un autre segment.

- **Ecriture de la procédure (ou sous-programme) :** Les instructions composant la procédure sont regroupées entre 2 mots clés :

```
Nom-sp  PROC near
        instruction1
        instruction2
        ...
        RET
Nom-sp  ENDP
```

- Nom-sp représente le nom de la fonction
- Le 1er mot clef PROC marque le début de la procédure
- Le 2nd mot clef RET désigne la dernière instruction
- ENDP annonce la fin de la procédure.

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

  - Lors de l'exécution de l'instruction CALL, le pointeur d'instruction IP est 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-`a-dire que IP doit être rechargé avec l'adresse de cette instruction.

  - Avant de charger IP avec l'adresse du sous-programme, l'adresse de retour au programme principal, c. à d. 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.

- **Syntaxe: PUSH Opérande**

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

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

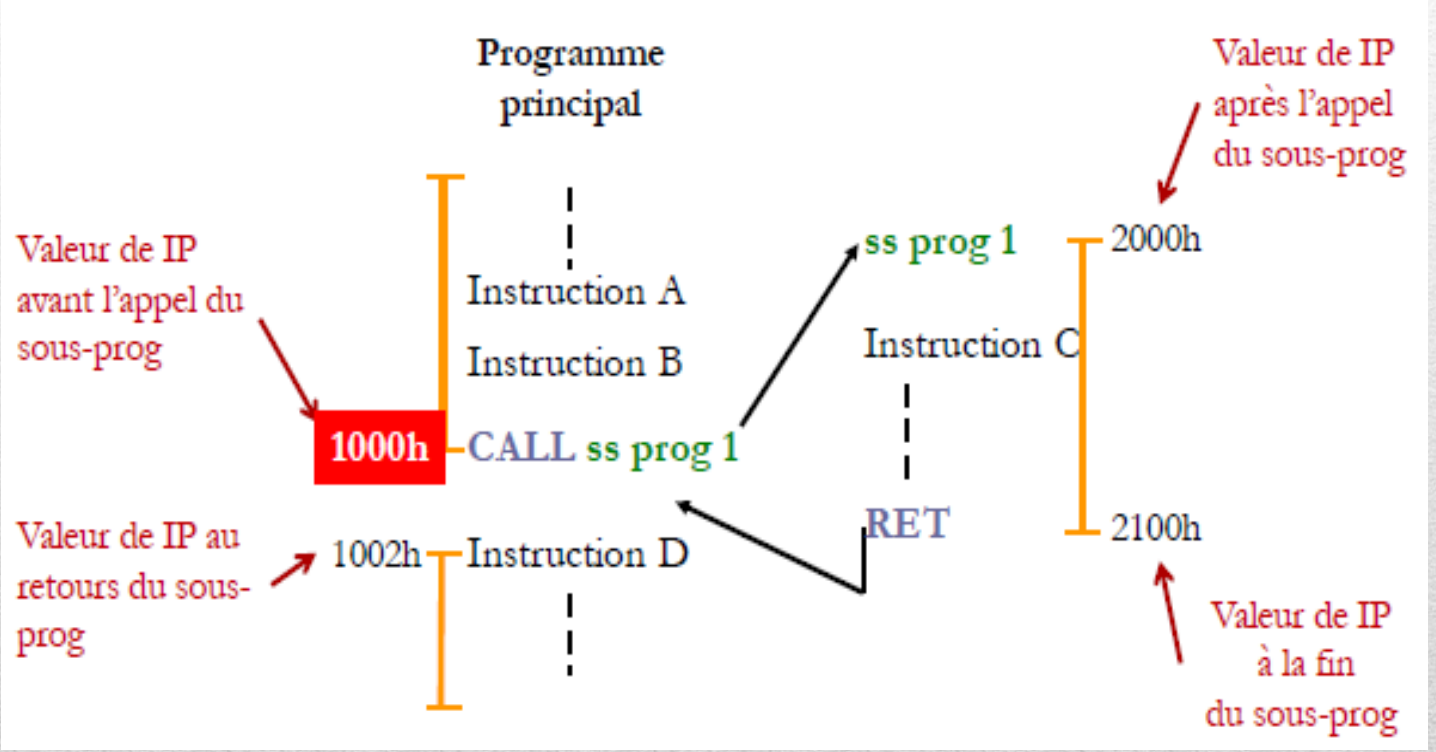
```
PUSH R16
PUSH word [adr]
```

- **Syntaxe: POP Opérande**

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

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

```
POP R16
POP word M
```


Programme principal
Valeur de IP après l'appel du sous-prog
Valeur de IP avant l'appel du sous-prog
Instruction A
Instruction B
1000h
CALL ss prog 1
ss prog 1
2000h
Instruction C
RET
2100h
Valeur de IP au retours du sous-prog
1002h
Instruction D
Valeur de IP à la fin du sous-prog