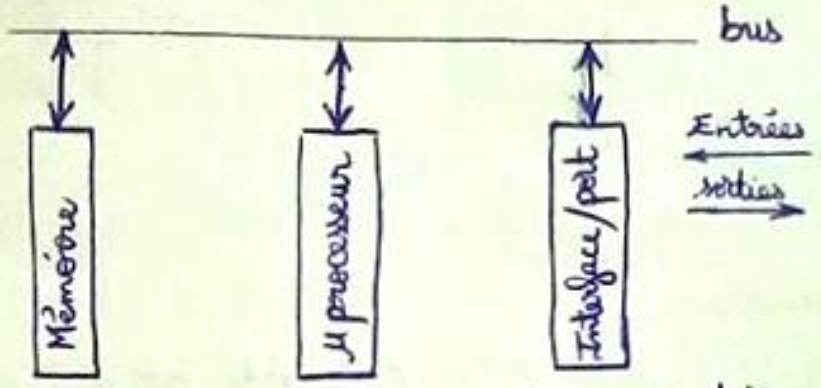
* Notion de base sur les µ processeurs :

S.M.P

* caractéristiques : nombre de transistors, largeur du bus de données et bus d'adresse, vitesse max d'horloge (en Quartz)

. Software : Assembleur (Langage machine)

. organisation :



. Stockage :
- programme
- données

. Gestion
. Exécution

. Communication
. adaptation : µp ↔ périphériques

Suite des instructions
Tâche le suivi

opération : Lecture, écriture / Logique / arithmétique

→ format des données : 8 / 16 / 32 bits

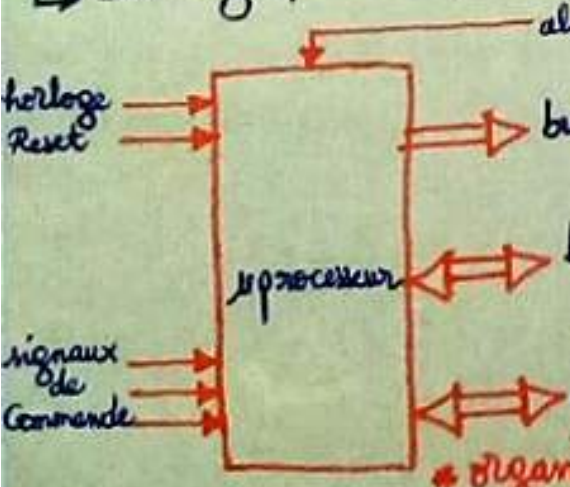
→ puissance de traitement : en MIPS (million instruction / seconde)

→ taille d'espace adressable : nombre des bits du bus d'adresse

Mémoires :

→ RAM mémoire vive volatile → basée sur des bascules
Lecture et écriture, données perdues hors tension
↳ Stockage des données temporaires.

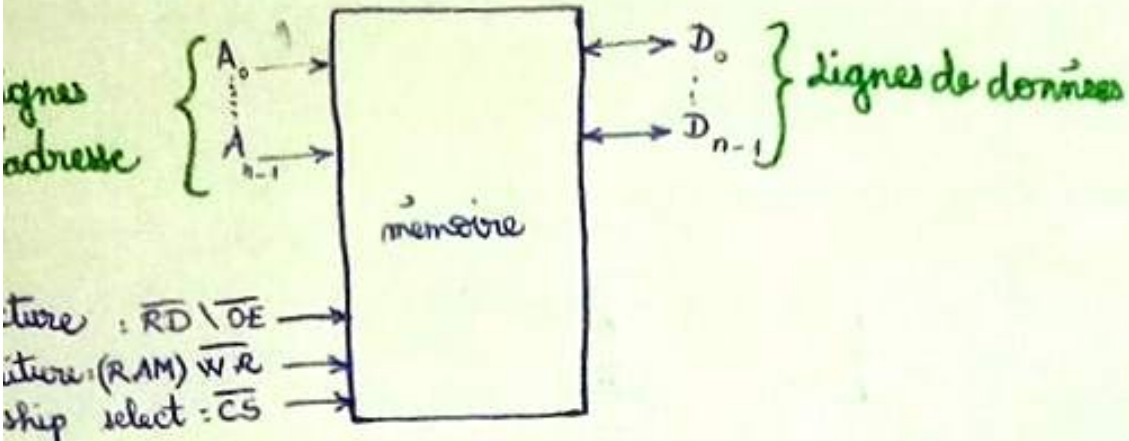
→ ROM mémoire morte non volatile (BIOS)
Lecture seulement, données conservées indéfiniment hors tension
↳ Stockage permanent du programme.



* organisation du µ processeur *

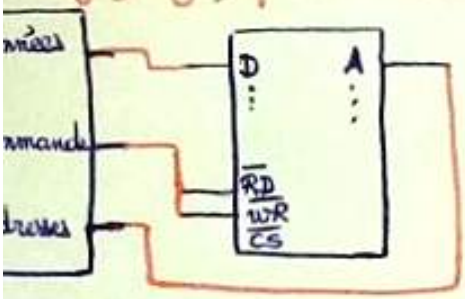
* Les mémoires RAM ou ROM *

caractéristiques : capacité : nbre total des cases mémoire



tifs à l'état bas

Interfaçage µp / mémoire :



2^3 2^2 2^1 2^0

A15 A14 A13 | A12 A11 .... A2 A1 A0

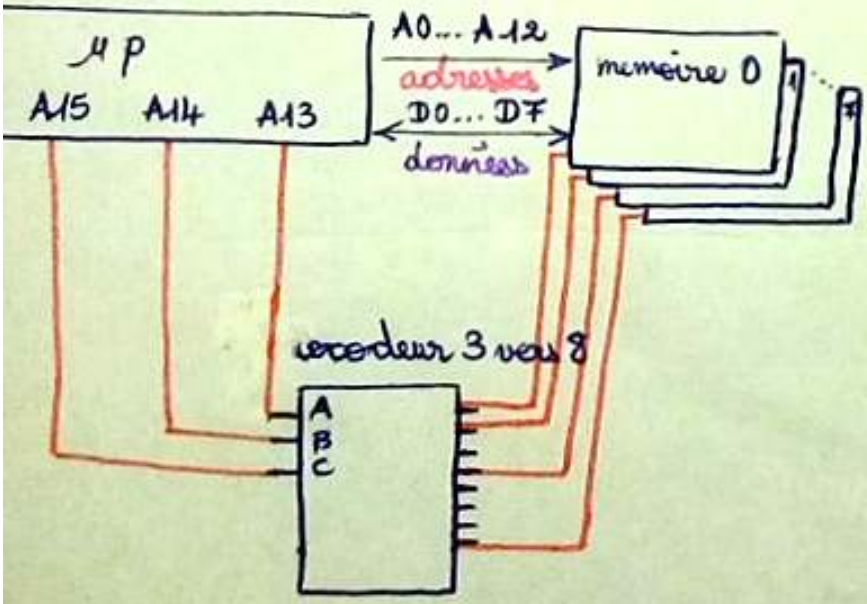
0000 (H) → FFFF (H) (16 bits) : mémoire totale adressable par µp

hexadécimal

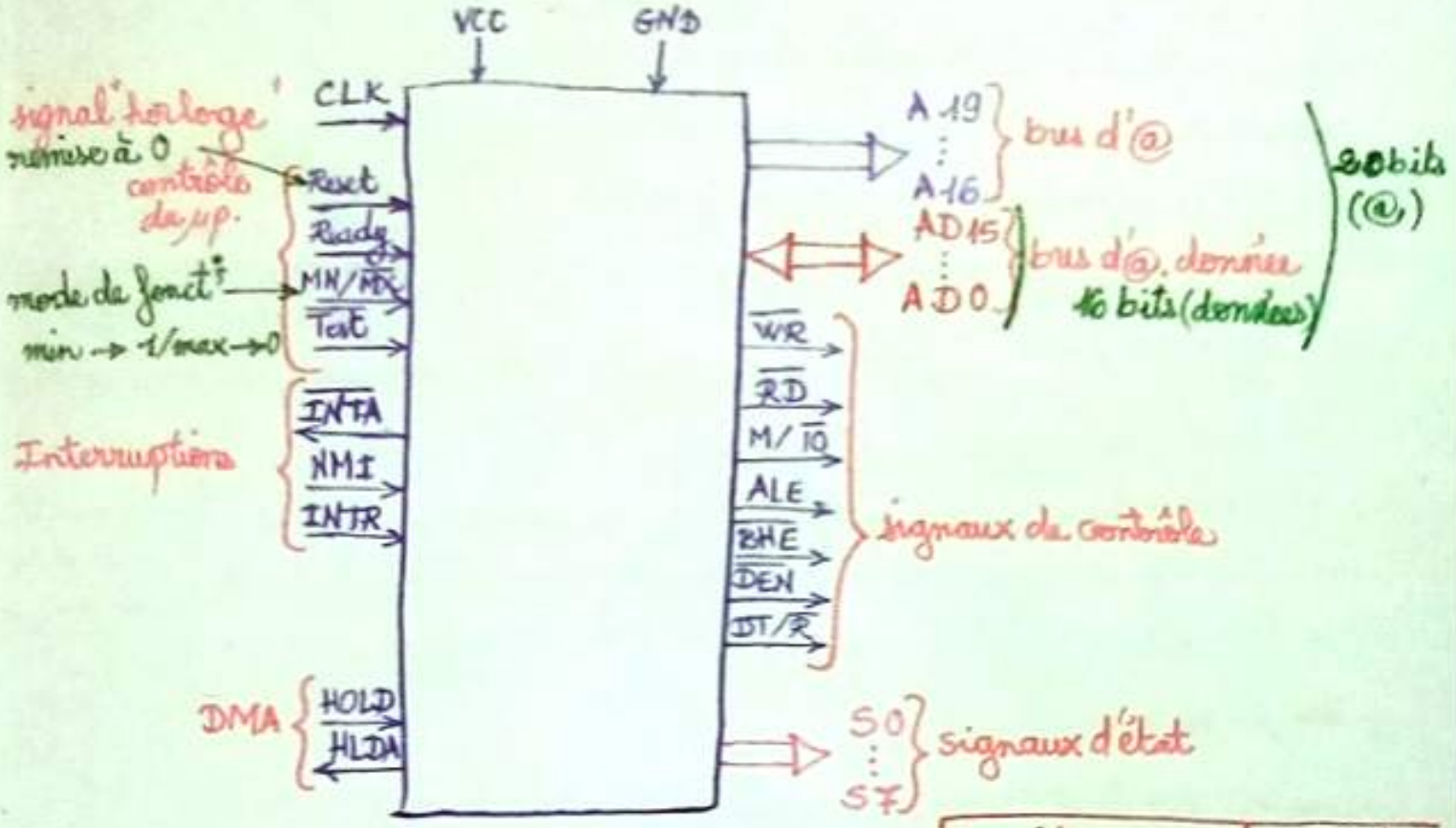
3 bus : bus d'adresse, bus des données, bus de commande.

Read, write, ...

Décodage d'adresse :



Chapitre : µp Intel 8086



registres segments : registres d'index peuvent être associés

Organisation internes du 8086 :

* unité d'exécution : EU

8 bits	8 bits	
AH	AL	= AX
BH	BL	= BX
CH	CL	= CX
DH	DL	= DX
SI		indexes
DI		indexes
SP		pointeurs
BP		pointeurs

registres généraux : 16 bits

Pointeurs indexe

* Unité d'Interface du bus : BIU

	16 bits
Data Segment	DS
Stack Segment	SS
Code Segment	CS
Extra Segment supplémentaire pour données	ES
	IP

registre de segments

pointeur d'instruction (flag)

* unité de commande *

→ 14 registres de 2 unités en 4 groupes : Tout registre : 16 bits

* Registres généraux *

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

* Registres d'adressage *

SI
BP
SI
DI

entre 2 zones mémoires

* Registres de segments *

CS
SS
CS
ES

* Registres de commande *

IP
FLAGS

• Registres généraux :

- AX : accumulateur : calcul arith., division, multiplication.
- BX : base : adressage, calcul sur les @ ⇒ Segment : offset ou [offset] — DS (par défaut).
- CX : comptage / calcul : compteur dans les boucles
- DX : data : stockage des données, division, multiplication : extension au registre AX pour < 16 bits

• Registre d'adressage :

zone de sauvegarde de données en cours d'exécution

- SP : pointeur de pile (offset → SS) : adressage
- BP : pointeur de base (offset → SS) : adressage des données sur pile.
- SI : registre d'index (source) (offset → DS) adressage
- DI : registre d'index (destination) (offset → DS) adressage

} transfert entre 2 zones mémoires

} contiennent adresses de cases mémoire

• Registres de segments :

- CS : code segment : début de mémoire programme : @ d'instructions du programme.
- DS : data segment : début de mémoire donnée : stockage des données traitées.
- SS : stack segment : début de pile LIFO :
- ES : extra segment : début de segment auxiliaire des données.

• Registres d'états :

• Flags :

16 bits :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

- 11 : O ⇒ OF : dépassement (overflow)
- 10 : D ⇒ DF : décrémentation
- 9 : I ⇒ IF : autorisation d'interruption
- 8 : T ⇒ TF : exécution pas à pas (trap)
- 7 : S ⇒ SF : indicateur signe
- 6 : Z ⇒ ZF : indicateur zéro
- 4 : A ⇒ AF : retenue auxiliaire
- 2 : P ⇒ PF : indicateur parité
- 0 : C ⇒ CF : indicateur retenue

• IP : pointeur d'instruction

* Format d'adresse : adresse ⇒ segment : offset adresse logique

2 registres pour @ d'une case mémoire donnée :

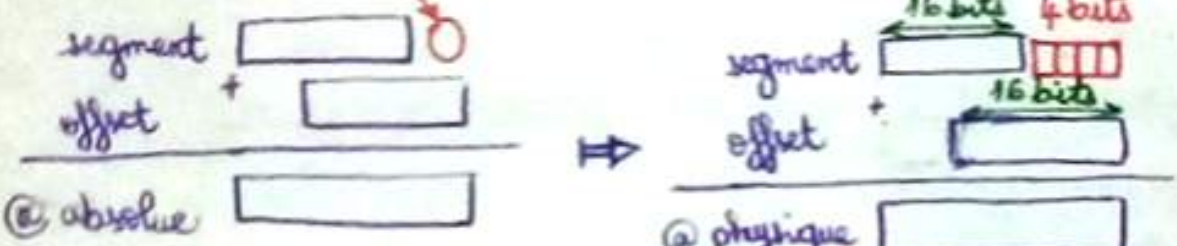
→ adressage segment : registre segment.

→ adressage à l'intérieur du segment : registre d'adressage / d'offset.

@ adresse physique = 16 × segment + offset : @ : déplacement nécessaire en mémoire / à @ de réf. pour atteindre @ de réf. c.à.d entre 2 emplacements mémoire

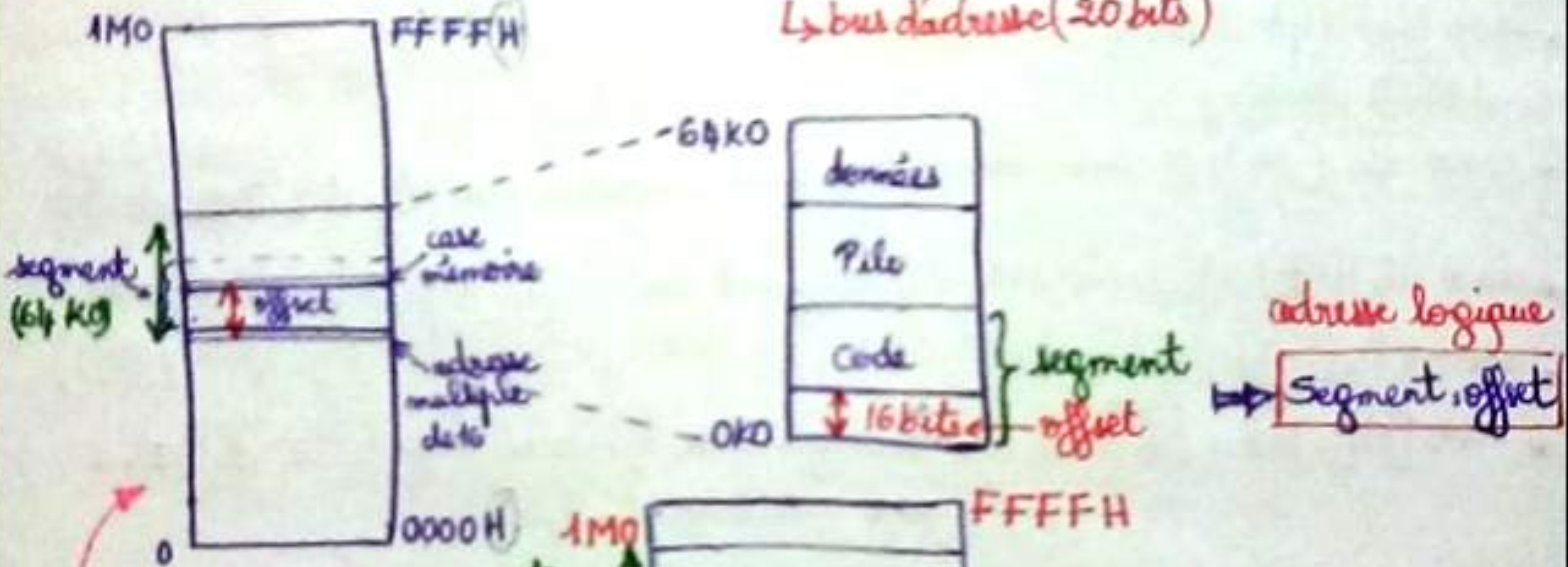
16 bits

décalage par le bit du poids faible

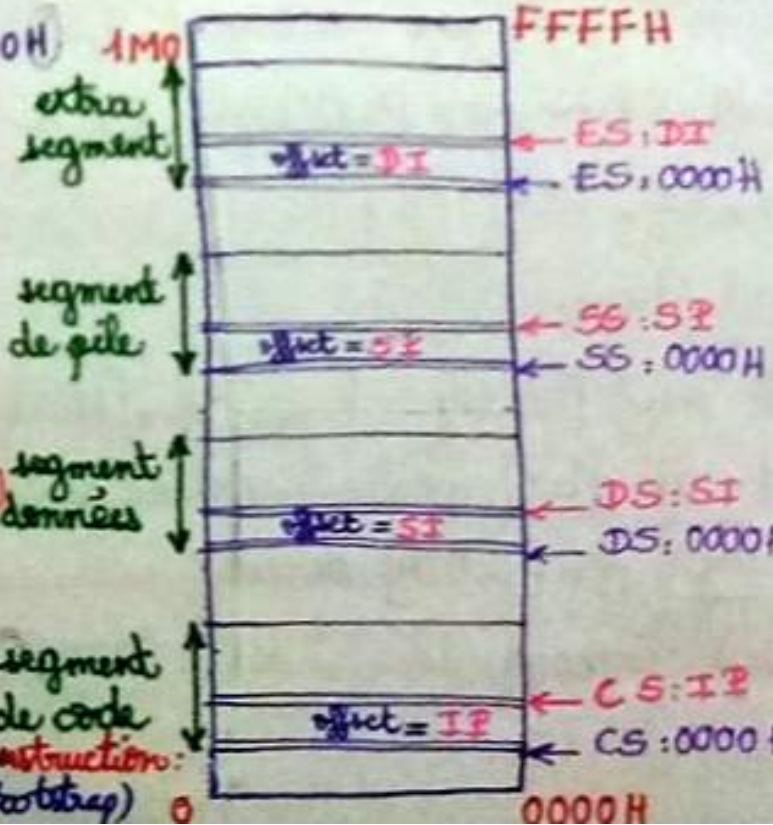


L→ bus d'adresse (20 bits)

* Gestion de mémoire par 8086 :



* segmentation de mémoire *



Contenu registres après RESET :

IP = 0000H (offset)

CS = FFFFH (segment)

DS = 0000H

SS = 0000H

ES = 0000H

L→ adresse logique de 1ère instruction :

⇒ FFFFH : 0000H (Bootstrap)

Programmation assembleur du 8086 :

* Instructions de transfert : deplacement des données : source (SI) → destination (DI)

3types : registre → mémoire, mémoire → registre et registre → registre.

mov destination , source

* modes d'adressage :

mov ax,bx = @ par registre.

mov al , 12h : @ immediat.

mov bl , [1200h] : @ direct.

* mov bl, ES: [1200h] : @ direct avec forçage de segment.

ajout du préfixe du segment

* mov byte ptr [1100H] : @ immediat avec spécificateur de format. (initialis

1octet :8 bits

* mov word ptr [1100H] : @ immediat avec spécificateur de format.

2 octets :16 bits

* mov al , [bx] : @ base sur DS segment associé par défaut à bx est DS (DS:[BX])

* mov al , [BP] : @ base sur SS segment associé par défaut à BP est SS

* mov al, [SI] : @ indexé (SS:[SI]) (SS:[BP])

* mov [DI], bx : @ indexé charger case mémoire d'offset DI et DI+1 par contenu du bx

* mov [SI+100H], ax ~ mov [SI][100H], ax ~ mov 100H[SI], ax

* mov ah, [bx+SI+100H] : @ base indexé

bx : base ; SI : index source

* Opérations Arithmétiques :

- ADD :
  - add ah, [1100H] : ah ← ah+[1100H] @ direct
  - add ah , [bx] : ajouter le contenu de bx au registre ax @ base
  - add byte ptr [1200H], 05H : @ immédiat avec spécificateur de format (1o = 8bits)
- SUB : SUB ah, [1100H] : ah ← ah − [1100H]

.MUL : * mov al, 51
mov bl, 32
mul bl
} ax = 51 × 32 ; résultat toujours dans ax

* si octet × octet : le résultat sur 16 bits et stocké dans ax

* mov ax, 4253
mov bx, 1689
mul bx
} (ax, dx) = 4253 × 1689

* si mot × mot (2 octets) : le résultat sur 32 bits et stocké dans ax et dx

.DIV :

* mov ax, 35
mov bl, 10
div bl
} . sur ax : al = quotient, ah = reste

* si division sur 1 octet : résultat 16 bits stocké dans ax où : { al = quotient ; ah = reste }

(attention les signes)

* mov dx, 0
mov ax, 1234
mov bx, 10
div bx
} . sur (ax, dx) : ax = quotient, dx = reste

* si division sur 2 octets : résultat 32 bits stocké dans (ax, dx) où : { ax = quotient ; dx = reste }

* Instructions logiques :

.AND : mov al, 10010110B
mov bl, 11001101B
and al, bl

```
1 0 0 1 0 1 1 0   al
1 1 0 0 1 1 0 1   bl
-----------------
1 0 0 0 0 1 0 0   al
```

→ application masquage :
masquer les bits 0, 1, 6 et 7 (mettre à 0) :

```
7 6 5 4 3 2 1 0
0 1 0 1 0 1 1 1
0 0 1 1 1 1 0 0   ⟸ masque
---------------
0 0 1 0 1 0 0
```

.OR : mov al, 10010110B
mov bl, 11001101B
or al, bl

ou bien :
mov al, 10010110B
or al, 11001101B

→ application masquage :
mettre à 1 les bits 1 et 3 :

```
7 6 5 4 3 2 1 0
1 0 1 1 0 0 0 1
0 0 0 0 1 0 1 0   ⟸ masque
---------------
1 0 1 1 1 0 1 1
```

. complément à 1 :

```
mov al, 10010001B
not al
```
} $al = \overline{10010001B} = 01101110B$

. complément à 2 :

```
mov al,25
mov bl, 12
neg bl
add al,bl
```
$al = 25 + (-12) = 13$

. XOR (ou exclusif) : **mise à 0 d'un registre**

```
mov al, 25
xor al,al
```
$al = 0$

. Instructions décalage et rotations :

* shr : décalage à droite ; * shl : décalage à gauche
* sar : décalage arithmétique à droite ou à gauche
* ror : rotation à droite ; * rol : rotation à gauche

. Instructions de branchement :

. jmp : saut

. jnz : jump if not zero

. cmp : comparaison
  jg : jump if greater
  jl : jump if less

. jz : jump if zero
. je : jump if equal
. jne : jump if not equal
. loop : repeter jusqu'à CX vérifié

. PROC :
  :
  ret
  ENDP

. call :

. Push :
. POP :
} entrer une donnée d'un registre et la supprimer

* Modes des Adressage :

```
mov ax, bx      @ par registre
mov ax, 12h     @ immédiat
mov ax, [1200h] @ direct
mov al, [bx]    } @ basé  sur DS
mov al, [bp]    }         sur SS
```

```
mov [SI+100h], ax
ou
mov [SI][100h], ax   } @ indexé avec déplacement
ou
mov 100h[SI], ax
mov ah, [bx+SI+100h]  @ indexé basé avec déplacement
```

* **Exercice** : Parmi les instructions suivantes, indiquer les instructions correctes et les modes d'adressage :

```
mov ax, [193] : @ direct
mov 11, ax : faux mov registre dans valeur
mov al, cx : faux mov registre 16bits dans registre à 8 bits
mov ax, 206 : @ immédiat
mov ax, [bx+DI] : @ indexé basé
```

* **Rappel** :

. affichage d'un caractère :

```
mov DL, 'A' : caractère
mov AH, 02h (ou bien mov AH, 2) : fonction n°2
INT 21H : appel système
```

(affichage)

. Saisir un caractère :

. avec écho : `mov ah, 01h` / `INT 21h` — Résultat dans AL (en hexadécimal)

. sans écho : `mov ah, 08h` / `INT 21h`

. arrêt du programme : `mov ah, 4Ch` / `INT 21h`

* **Exercice :**

* Affichage alphabet en majuscule :

```
CODE Segment
ASSUME CS: code
Main  mov DL, 'A'
      mov Cx, 26
ENC : mov ah, 2
      INT 21h
      INC Dl
      Loop ENC
      Mov ax, 4C00h
      INT 21h
code ENDS
END Main
```

* saisir un caractère du clavier et afficher son code

```
CODE Segment
ASSUME CS:CODE
MAIN: mov AH, 1
INT 21h
```