

**Centre De Formation \*\*HBM Corporation\*\***

## **Programmation Et Interfaçage De Microprocesseur 8086**

### *Support de cours*

Pour les étudiants : 4<sup>ème</sup> Electronique et automatique.  
5<sup>ème</sup> Electronique.  
2 et 3<sup>ème</sup> Informatique.  
Système LMD.

**M<sup>er</sup>. ATOUI Hamza**

*Ingénieur en électronique option contrôle industriel  
Deuxième année magistère option TELECOM et  
Traitement du signal numérique*

Email : [HBMVIPH@hotmail.com](mailto:HBMVIPH@hotmail.com)

2006-2007

## Table Des Matières

I)	Introduction
II)	Macro Architecture
	1- Exemple d'une unité de transfert
	2- Architecture de l'unité de décodage et de contrôle
III)	Conception de $\mu$ p vue de 8086
IV)	Architecture de $\mu$ p 8086
	1- Les bus
	2- Schéma synoptique de $\mu$ p 8086
	3- Registres de travaux
	4- Registres de segments
	5- Le registre FLAG
V)	Les modes d'adressages
	- Introduction
	1) Adressage registre
	2) Adressage immédiat
	3) Adressage direct
	4) Adressage indirect par registre
	5) Adressage basé
	6) Adressage indexé
	7) Adressage basé indexé
	8) Adressage des entrées sorties (I/O)
	9) Adressage relatif
	10) Adressage vectorisé
VI)	Jeu d'instructions
	a) Instructions de Transferts
	b) Instructions Arithmétiques
	c) Instructions Logiques
	d) Instructions de traitement de chaîne
	e) Instructions de saut et d'appelle sous programme
	f) Instructions des interruptions logicielles
	g) Instructions de contrôle du $\mu$ p
VII)	Interfaçage de $\mu$ p 8086
	1) Description physique de $\mu$ p 8086
	2) Schéma fonctionnel de $\mu$ p 8086
	3) Conception de bus système de $\mu$ p 8086
	4) Interfaçage Mémoires
	5) Les Interfaces parallèles
	- Port parallèle
	- Coupleur parallèle 8255
	6) Les Interface séries
	- Coupleur série 8250 (Norme RS232)
	- Le bus SPI (8 bits, 16 bits)
	- Le bus I2C (standard, smart)
	7) Le Timer 8253/8254
	8) Le Coprocesseur mathématique 9511 AMD
VIII)	Le mode interruptible
	Annexe – jeu d'instructions 8086-
	Bibliographie.

***Remerciement***

Remerciements à **M. FRIHI** et **M. BOU-TALBI** et **M. BEN-SAOULA** et **M. LAKEL** pour les efforts fournis aux étudiants de 4<sup>ème</sup> et 5<sup>ème</sup> année électronique pour comprendre le module.

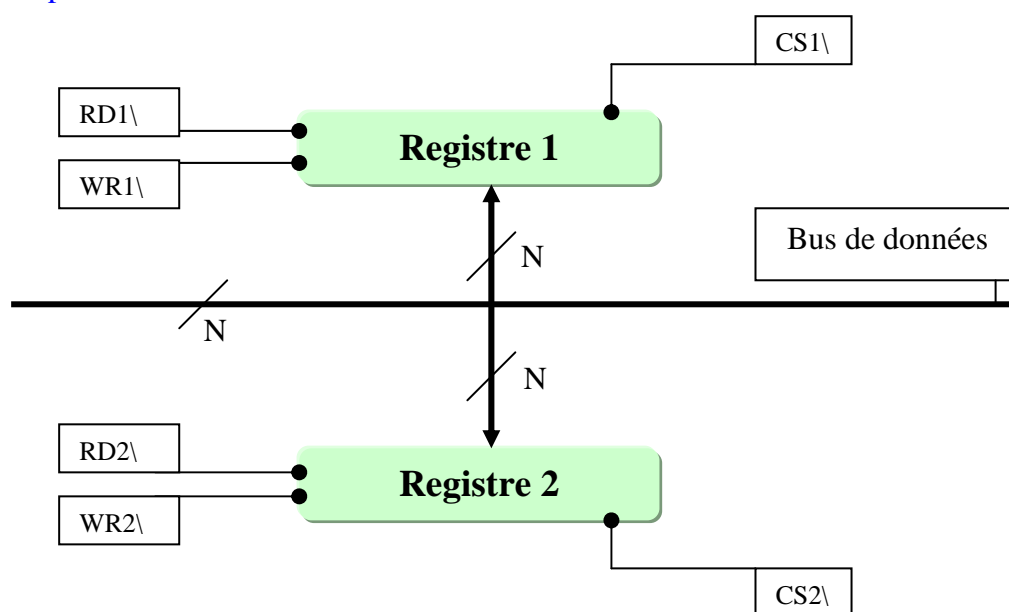
## I) Introduction :

Un microprocesseur est un circuit intégré complexe. Il résulte de l'intégration sur une puce de fonctions logiques combinatoires (logiques et/ou arithmétique) et séquentielles (registres, compteur,...). Il est capable d'interpréter et d'exécuter les instructions d'un programme. Son domaine d'utilisation est donc presque illimité.

Le concept de microprocesseur a été créé par la société INTEL. Cette Société, créée en 1968, était spécialisée dans la conception et la fabrication de puces mémoire. À la demande de deux de ses clients — fabricants de calculatrices et de terminaux — Intel étudia une unité de calcul implémentée sur une seule puce. En 1971, c'est la date de premier microprocesseur, le 4004, qui était une unité de calcul 4 bits fonctionnant à 108 kHz. Il résultait de l'intégration d'environ 2300 transistors.

## II) Macro Architecture :

### 1) Exemple d'une unité de transfert :



Discrétion des signaux :

RD\ : signal de lecture active au niveau bas.

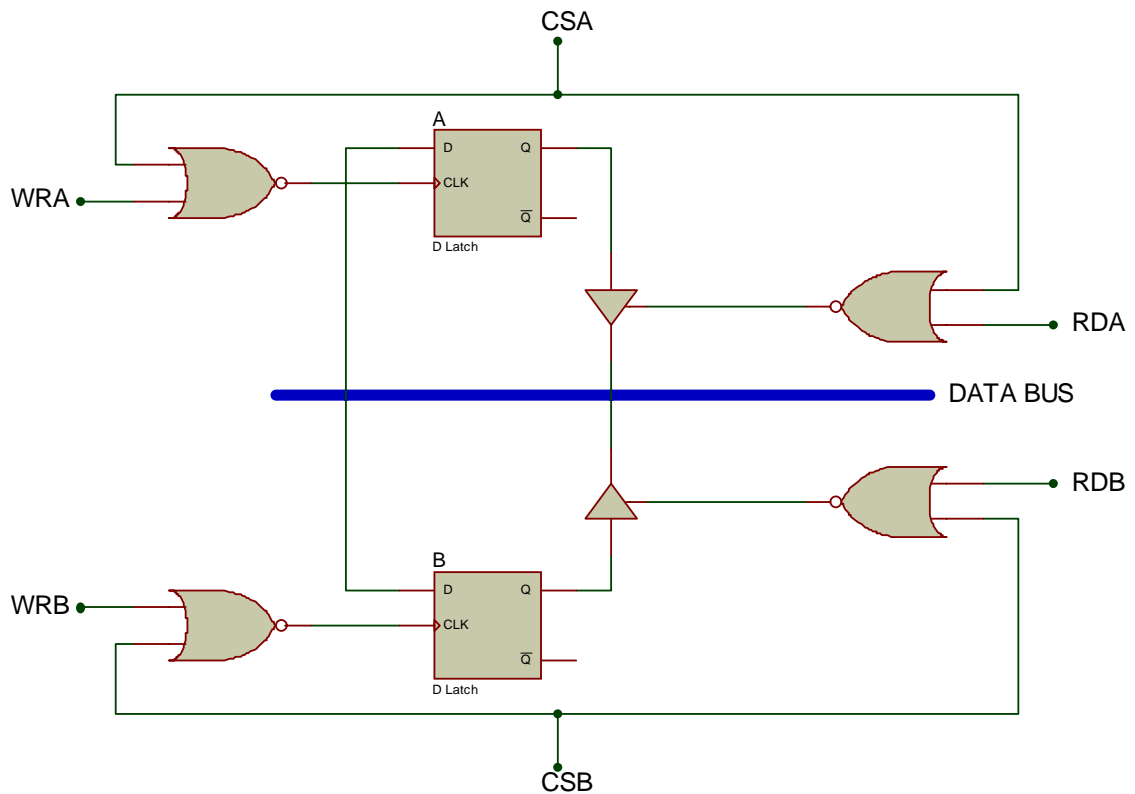
WR\ : signal d'écriture active au niveau bas.

CS\ : activation de registre (validation de BUS)

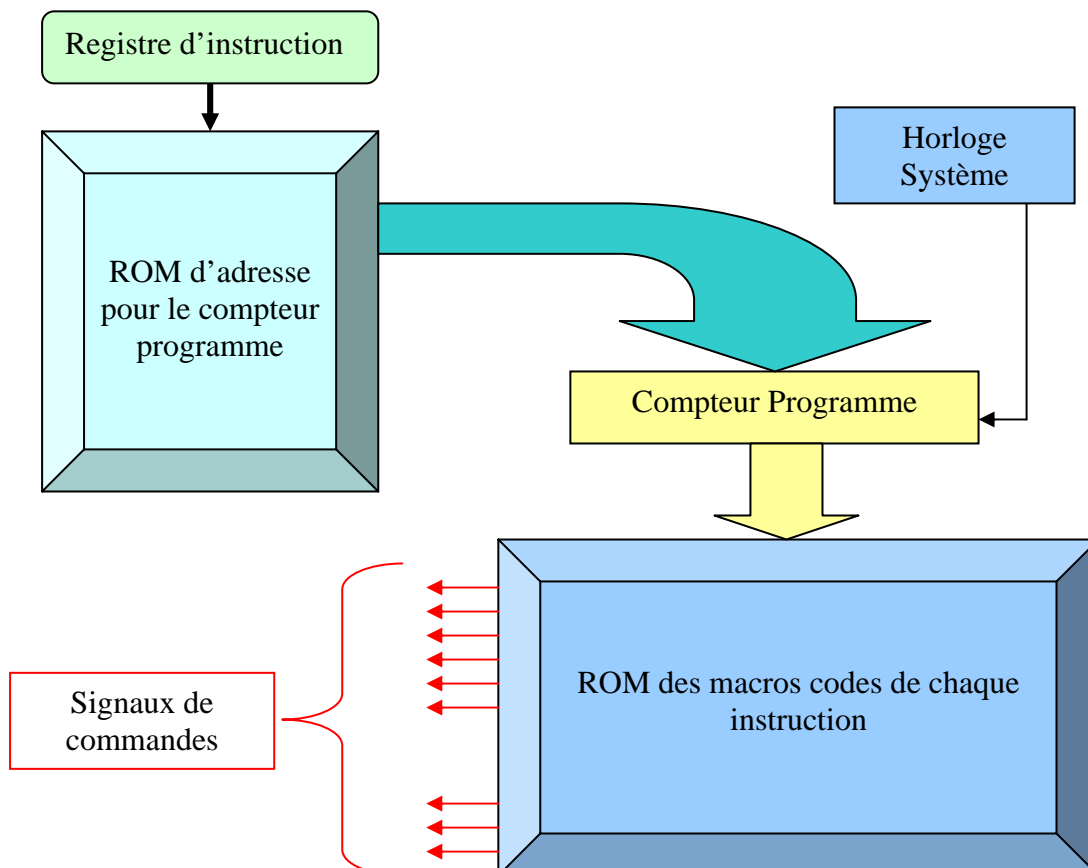
On demande d'activer les signaux de contrôles (RD\, WR\, CS\) de chaque registre pour faire un transfert du registre 1 vers le registre 2.

Temps	Signaux de contrôles de registre 1			Signaux de contrôles de registre 2			Commentaires
	CS\	RD\	WR\	CS\	RD\	WR\	
T	1	1	1	1	1	1	Etat de repos
T+1	0	0	1	1	1	1	Activation + lecture de reg 1
T+2	0	0	1	0	1	1	Activation de reg 2
T+3	0	0	1	0	1	0	Activation+écriture dans reg2
T+4	1	1	1	1	1	1	Revient à l'état initial

Exemple avec les circuits logiques :

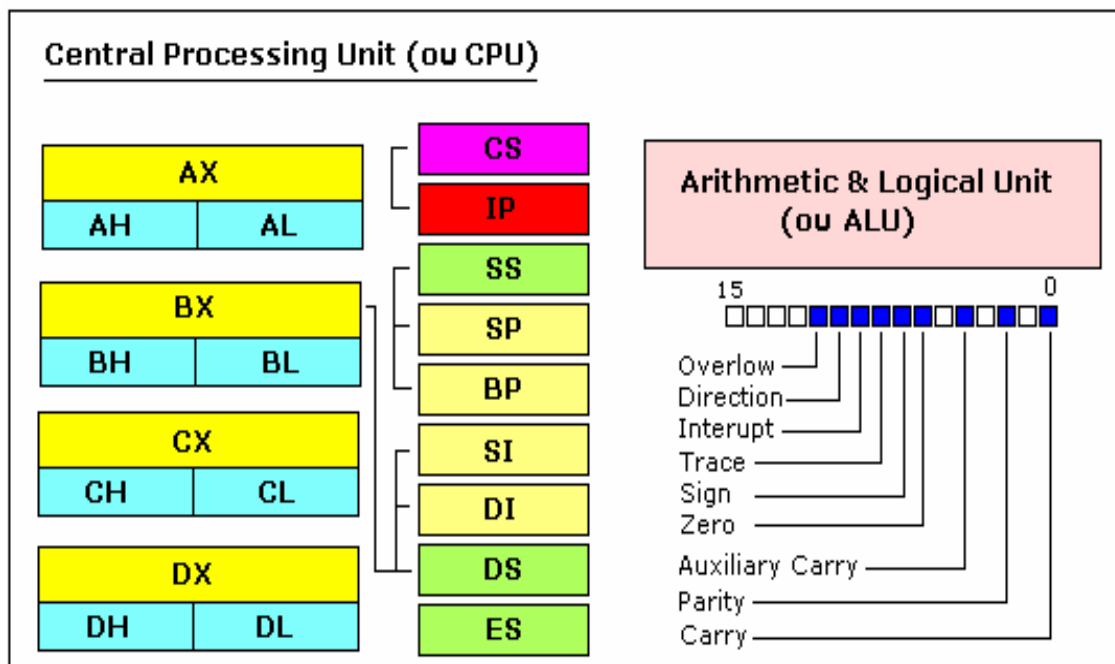
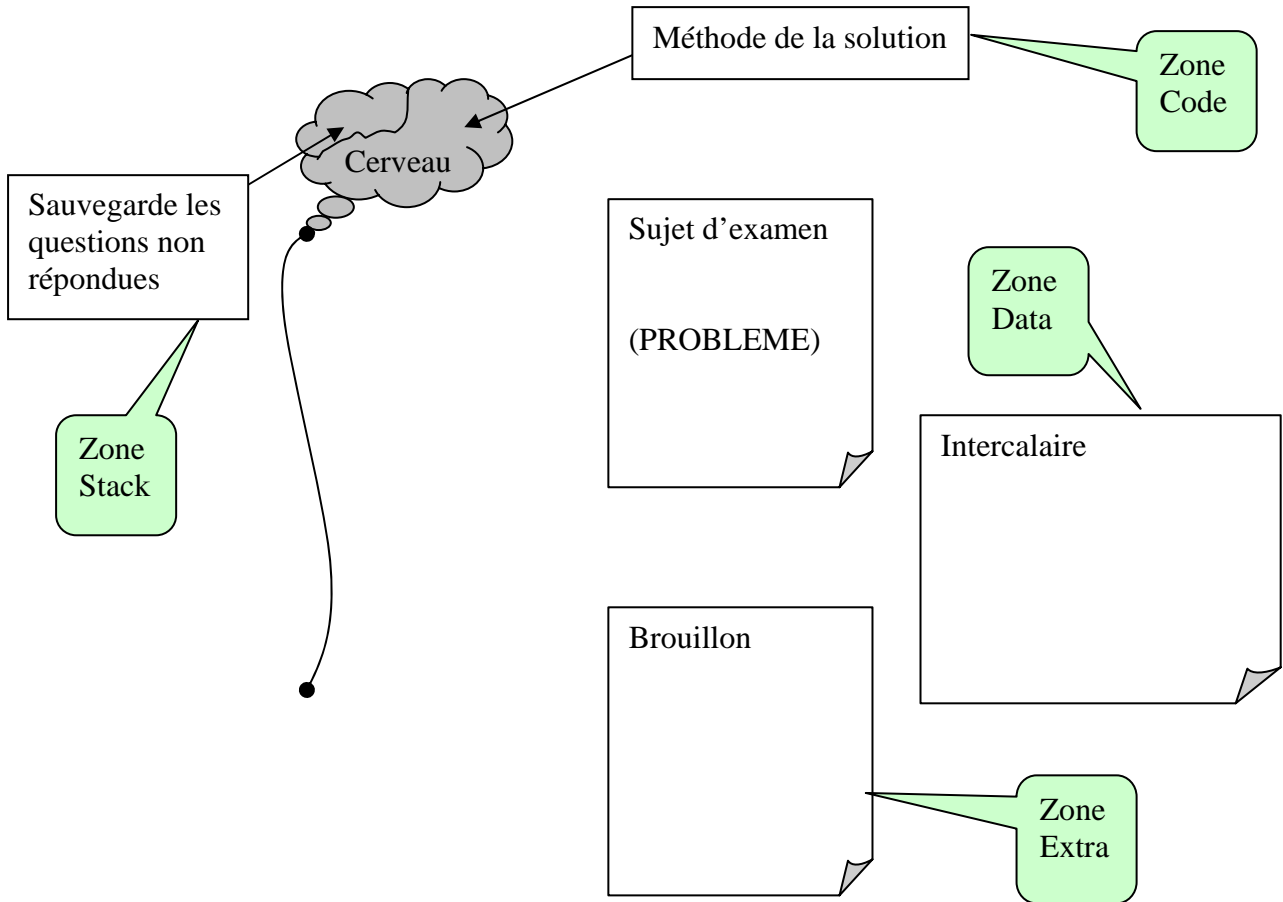


## 2) Architecture de l'unité de décodage et de contrôle :



### III) Conception de $\mu$ p vue de 8086 :

La conception de INTEL est basée sur une modélisation de raisonnement du cerveau humain. Pour simplifier les choses, je donne la scène de résolution de sujet d'examen.



#### IV) Architecture du µp 8086 :

##### 1) Les bus :

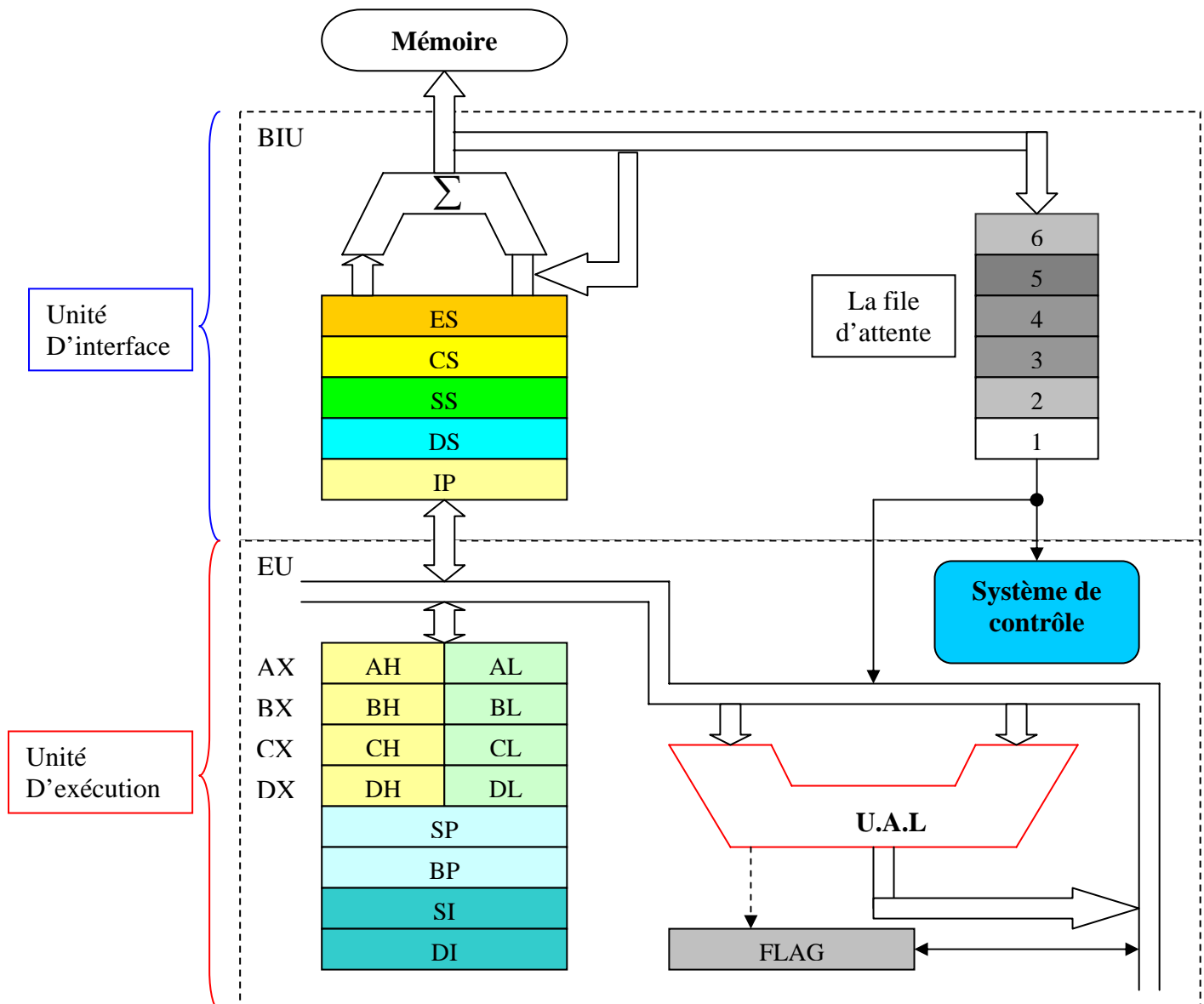
Un bus est un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

**Bus de données** : c'est un bus bidirectionnel qui assure la transmission et la réception des données en parallèle. (Pour le 8086 est de taille 16 bits).

**Bus d'adresse** : c'est un bus unidirectionnel qui permet la sélection des informations à traiter dans un *espace mémoire* (ou *espace adressable*) qui peut avoir  $2^n$  emplacements, avec  $n$  = nombre de conducteurs du bus d'adresses. (Dans le cas du 8086 est de taille 20 bits).

**Bus de contrôle** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

##### 2) Schéma synoptique du µp 8086 :



### 3) Registres de travaux :

Le CPU 8086 comporte 8 registres à usage général, chaque registre est identifié par son nom :

AX	Registre accumulateur 16 bits (composé de deux registres 8 bits, <b>AH</b> et <b>AL</b> ).
BX	Registre d'adresses de base 16 bits (composé de deux registres 8 bits, <b>BH</b> et <b>BL</b> ).
CX	Registre compteur 16 bits (composé de deux registres 8 bits, <b>CH</b> et <b>CL</b> ).
DX	Registre de données 16 bits (composé de deux registres 8 bits, <b>DH</b> et <b>DL</b> ).
SP	Registre pointeur de pile 16 bits.
BP	Registre pointeur de base 16 bits.
SI	Registre d'index source 16 bits.
DI	Registre d'index destination 16 bits.

### 4) Registres de segments :

CS	Segment de code contenant le programme en cours.
DS	Segment de données contenant l'adresse du segment de données dans lequel le programme déposera ses éléments de travail (variables, champs, tableaux de branchement, etc.).
ES	Extra segment appartient au programmeur pour définir son utilisation.
SS	Segment de pile contenant l'adresse de la pile.

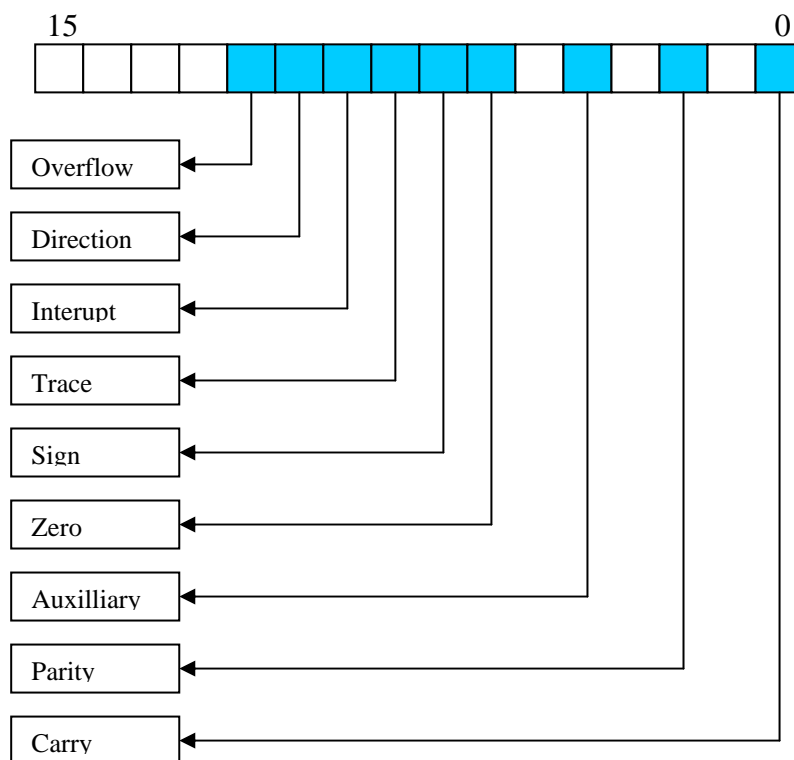
Plus 2 registres spéciaux :

Le registre IP : pointeur d'instruction.

Le registre FLAG : indique l'état en courant de µp.

### 5) le registre FLAG :

La plupart des instructions arithmétiques et logiques affectent le registre d'état du processeur (ou registre de FLAG).

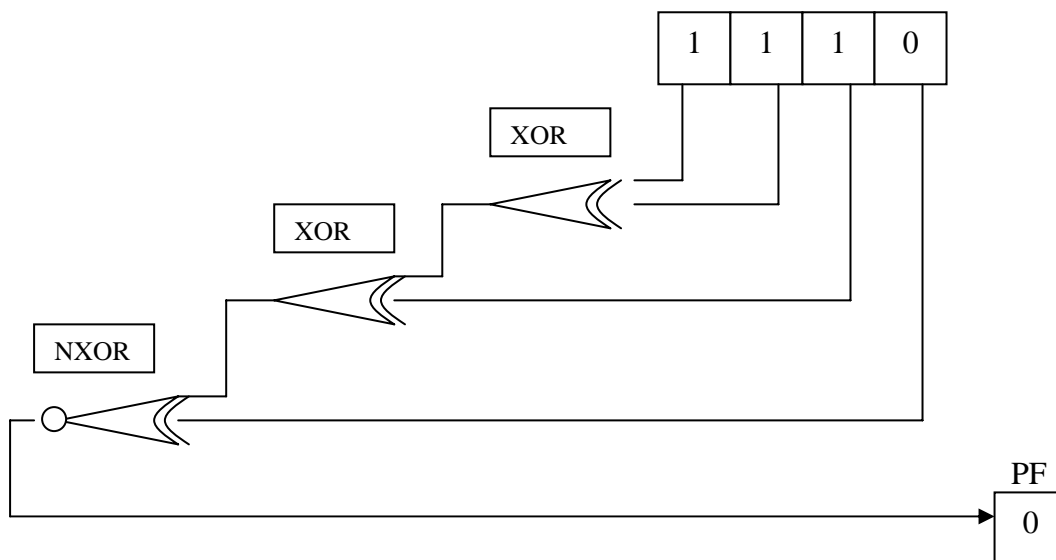


Comme vous pouvez le remarquer, c'est un registre 16 bits, chaque bit s'appelle un flag (ou un indicateur) et peut prendre la valeur **1** ou **0**.

**CARRY flag** : ce flag positionne à « 1 » lorsque il y a un débordement.

$$\begin{array}{cccc}
 & 1 & 0 & 0 & 1 \\
 + & 1 & 0 & 0 & 0 \\
 \hline
 = & 0 & 0 & 0 & 0 \\
 \text{CF} & & & & \\
 \boxed{1} & & & & 
 \end{array}$$

**PARITY flag :** la parité est celle du nombre de bits égaux à « 1 » s'il est pair PF est mis à « 1 ».

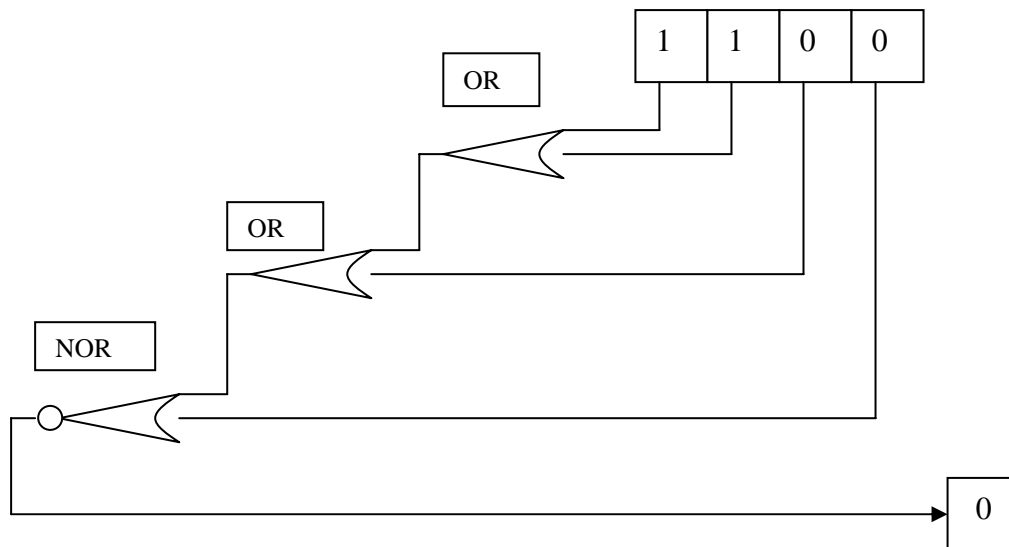


**AUXILIARY flag :** ce flag est à « 1 » lorsqu'un **débordement** a lieu de demi-mot de 8bits ou 16bits.

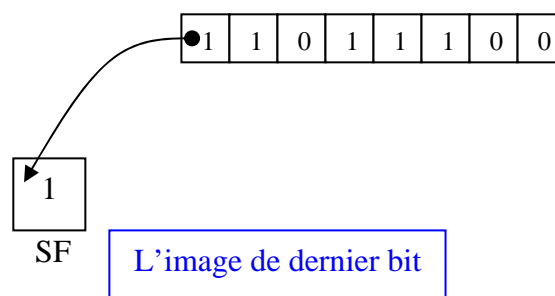
bits.

	AF			
		<div>1</div>		
	1	0	1	0
+	0	0	1	1
<hr/>				
=	1	1	0	1

**ZERO flag** : ce flag est à « 1 » lorsque tous les bits d'un mot sont à « 0 ».



**SIGN flag** : ce flag est à **1** lorsque le résultat est **négatif**. Lorsque le résultat est **positif**, le flag est à **0**. Ce flag prend la valeur du bit le plus fort (MSB).



**OVERFLOW flag** : ce flag est à « 1 » lorsqu'un **débordement signé**.

0	1	1	1	(+7)
+	0	1	0	(+5)
=	1	1	0	0

(Résultat non signé = 12)  
(Résultat signé = - 4).

OF  
1

**TRACE flag :** ce bit est à « 1 » signifie que l'exécution est en mode pas à pas (STEP BY STEP).

**INTERUPT flag :** si ce bits à « 1 » le  $\mu$ p prend en compte les interruptions venant de la pin INTR (INTERRUPT REQUEST) « les interruption hardware ».

**DIRECTION flag :** ce flag est utilisé par quelques instructions pour traiter les chaînes de données, Lorsque ce drapeau est placé à **0**, la chaîne est traitée octet par octet en **incrémentant**, lorsque ce drapeau est placé à **1**, la chaîne est traitée octet par octet en **décrémentant**.

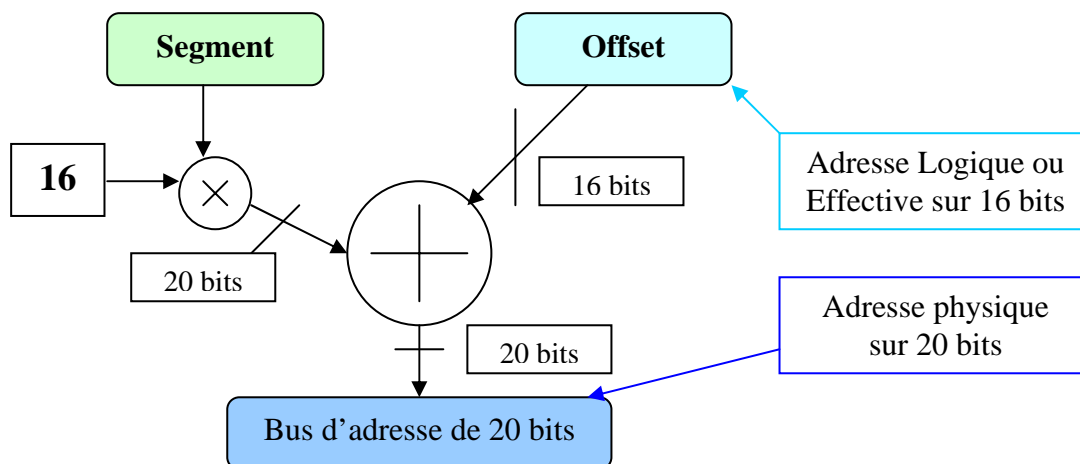
## V) Les modes d'adressages :

### Introduction :

Avant de commencer, il faut résoudre le problème de la taille de bus d'adresse de 20 bits et la taille des registres internes du  $\mu$ p (16 bit), la question qui il faut poser est « Quelle est la solution proposer par INTEL pour fabriquer une adresse de 20 bits à partir des registres de 16 bits ? ».

La solution est la suivante :

Chaque registre de segment sera multiplié par 16 plus un offset sur 16 bits comme l'indique le schéma au-dessous :



1) **Adressage registre** : ce mode utilise les registres internes de  $\mu$ p, dans ce mode il faut respecté la taille de différents registres.

#### Exemple :

Dans le plus part du cours, on va utiliser l'instruction MOV (instruction de transfert) :

#### Syntaxe :

MOV Destination, Source ; Destination  $\leftarrow$  Source.

MOV BX, AX ; Transfert du contenu de AX vers BX.

MOV BX, AL ; Déclaration d'erreur par ce que BX est registre 16 bits et  
; AL est registre 8 bits.

MOV BL, BH ; Transfert du contenu de BH vers BL.

2) **Adressage immédiat** : est un mode de manipulation des valeurs latéraux avec un registre interne ou une case mémoire.

#### Exemple :

MOV Alpha, 40h ; Alpha est une case mémoire de 8 bits.

MOV AL, 102h ; déclaration d'une erreur, AL registre de 8 bits, mais 102h est  
; une valeur immédiate sur 16 bits.

MOV AX, 40h ; cette instruction est juste  
; Par ce que, on peut écrire 40h  $\rightarrow$  0040h valeur sur 16 bits.

3) **Adressage direct** : comme l'indique son nom, c'est la réalisation des opérations entre les registres internes de  $\mu$ p et la mémoire.

#### Exemple :

MOV AL, Bêta ; transférer le contenu de la case mémoire Bêta vers AL.

MOV (2000h), BL ; transférer l contenu de BL vers la case mémoire de l'adresse  
; 2000h par rapport au DATA segment.

4) **Adressage indirect par registre** : c'est le mode d'accès entre la mémoire et les registres internes de µp par un intermédiaire, ces sont les registres pointeurs : (SI, DI, BX).

**Exemple :**

MOV AX, [BX] ; transfert de contenu de la case mémoire 16 bits pointer par BX  
; par rapport au DATA segment vers AX.

MOV [SI], DL ; transfert de contenu de DL vers la case mémoire 8 bits pointer  
; par SI par rapport au DATA segment.

5) **Adressage basé** : tous simplement, il utilise la registres de base comme pointeur entre la mémoire et les registres internes de µp ; les registres de base sont (BX, BP).

**Syntaxe :**

MOV [BX/BP + déplacement 8 bits ou 16 bits], source.

MOV destination, [BX/BP + déplacement 8 bits ou 16 bits].

**Exemple :**

MOV [BX+12], AL ; transfert de contenu de AL vers la case mémoire pointer par  
; (BX+12) par rapport au DATA segment.

MOV SI, [BP+400] ; transfert de contenu de la case mémoire 16 bits pointer par  
; (BP+400) par rapport au STACK segment vers SI.

6) **Adressage indexé** : le même mode que le basé, mais il utilise les registres d'indexés comme pointeur ; les registres d'indexés sont (SI, DI).

**Syntaxe :**

MOV [SI/DI + déplacement 8 bits ou 16 bits], source.

MOV destination, [SI/DI + déplacement 8 bits ou 16 bits].

Remarque : les cases mémoires consultées ou modifiées sont par rapport au DATA segment

7) **Adressage basé indexé** : c'est un mode combine entre de modes d'adressage comme l'indique son nom.

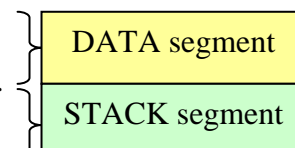
**Syntaxe :**

MOV [BX+ SI/DI + déplacement 8 bits ou 16 bits], source.

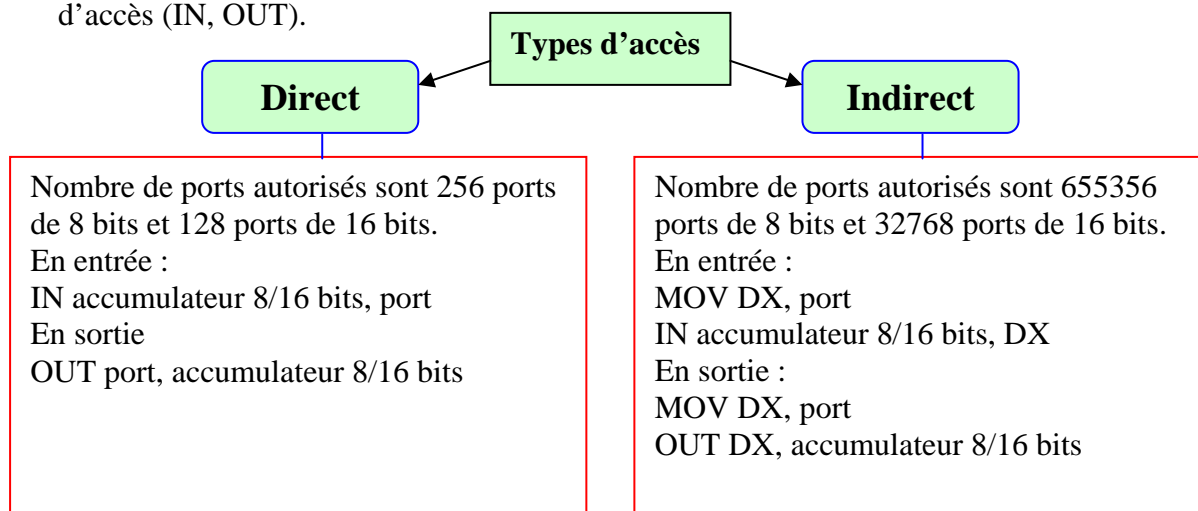
MOV destination, [BX + SI/DI + déplacement 8 bits ou 16 bits].

MOV [BP+ SI/DI + déplacement 8 bits ou 16 bits], source.

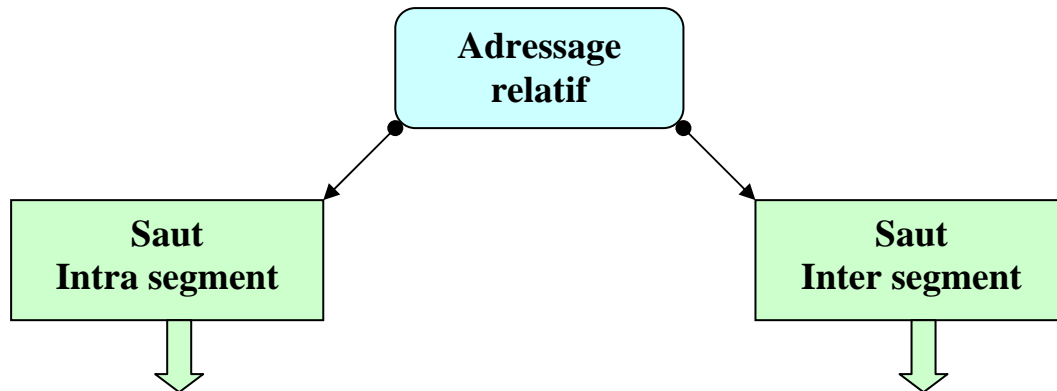
MOV destination, [BP + SI/DI + déplacement 8 bits ou 16 bits].



8) **Adressage des entrées sorties (I/O)** : dans ce mode, le bus d'adresse devient de taille 16 bits → l'espace adressable par I/O est 64 KO. Il utilise comme instructions d'accès (IN, OUT).



9) **adressage relatif** : ce type d'adressage est utilisé par les instructions de saut.



Ce type de saut change seulement le pointeur d'instructions (IP), et le CS ne change pas.

- 1) **saut conditionnel** : le saut n'est réalisé que si une condition est vérifiée. L'adresse est calculée en ajoutant au contenu de (IP), avec cette valeur est un déplacement signé sur 8bits.

Exemple :

DEC CX

JNZ ETIQUETTE

CALL SEND\_BYTE

- 2) **saut inconditionnel** : le saut est réalisé quelque soit l'état de  $\mu$ p, 2 types d'adressage sont utilisés :

- a) **ADR direct relatif** : l'adresse est calculée par l'ajout d'un déplacement signé à (IP).

**Long** : (+32767 à -32768 bytes)

**Exemple** : JMP LONG ETIQ1

**Court** : (+127 à -128)

**Exemple** : JMP SHORT ETIQ2

- b) **ADR indirect** : ici, la nouvelle valeur du contenu de (IP) est le contenu d'une case mémoire ou d'un registre.

**Exemple** :

JMP AX ; IP=AX (par registre)

JMP WORD PTR [BX] ; IP=case de 16 bits pointé par BX par rapport à DS (par mémoire).

Dans ce mode de saut le changement se fait sur 32 bits [CS:IP].

- 1) **ADR direct** : la valeur de saut est précisée le programme après le code de l'instruction de saut

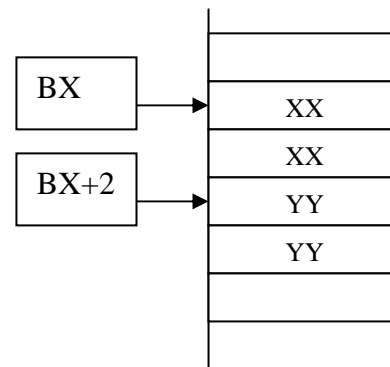
**Exemple** :

JMP 6060h:4000h ; CS  $\leftarrow$  6060h IP  $\leftarrow$  4000h.

- 2) **ADR indirect** : la valeur de saut (32bits) se trouve dans la mémoire, dans ce mode on utilise le mode d'adressage indirect.

**Exemple** :

JMP DWORD PTR [BX]



IP  $\leftarrow$  [BX].

CS  $\leftarrow$  [BX+2].

IP  $\leftarrow$  XXXX.

CS  $\leftarrow$  YYYY.

10) **Adressage vectorisé** : ce mode d'adressage est utilisé seulement pour le traitement des d'interruption.

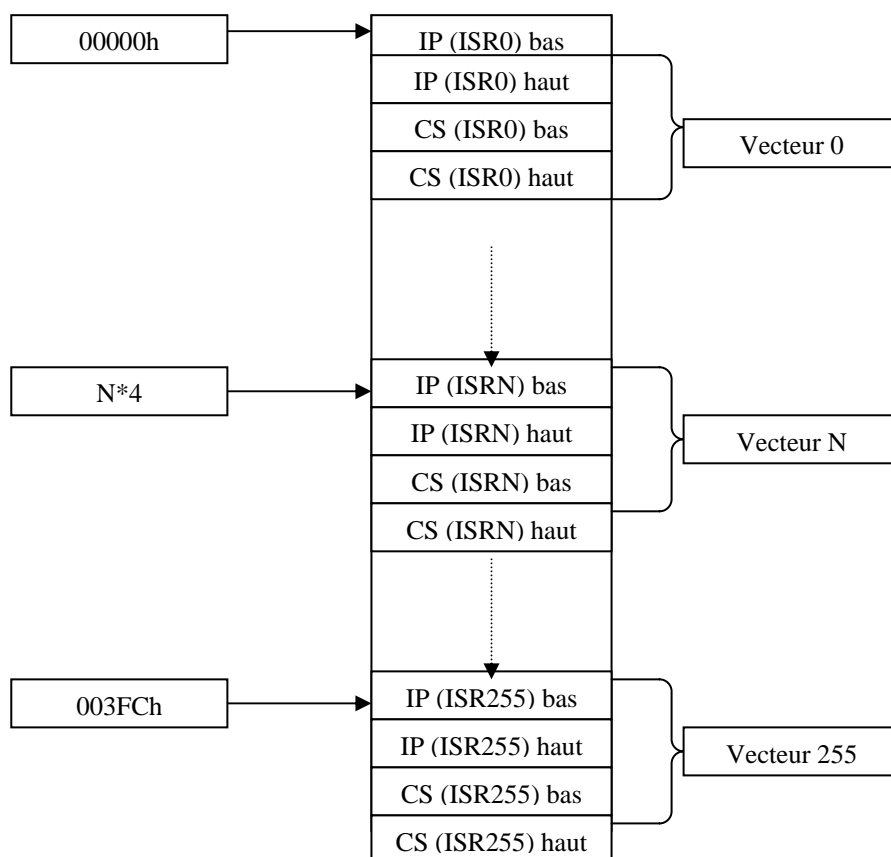
Le mot **VECTEUR** est l'adresse de l'ISR. (INTERUPT SERVICE ROUTINE) dans le premier Kilo bytes du bloc mémoire (00000h : 003FFh).

Cet espace mémoire est segmenté en 256 vecteurs → 4 bytes par vecteur.

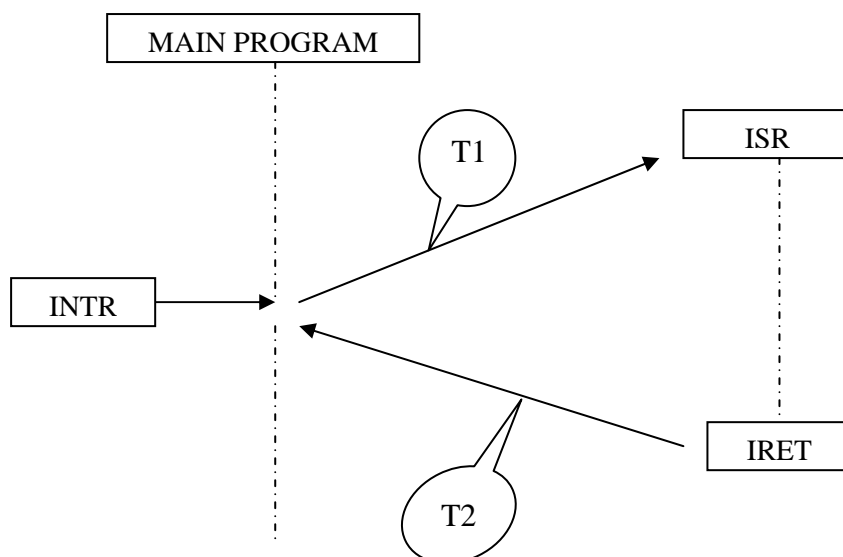
**Pour quoi 4 bytes ?**

Par ce que le vecteur composé par la valeur de (CS) et la valeur de (IP) de l'ISR.

L'organisation de **la table des vecteurs** :



Le mécanisme général de traitement d'une interruption :



Dans (T1) le µp sauvegarde le contenu de l'adresse de retour (CS : IP) et le registre FLAG dans le sommet de la PILE de la façon suivante :

-IP→CS→ le registre FLAG.

-Et mettre le registre FLAG INTERRUPT (IF) à zéro.

Le µp exécute le (ISR) jusqu'à la dernière instruction (IRET), cette dernière dépile le sommet de la (PILE) par l'ordre suivant le registre FLAG→CS→IP.

Et autorise l'interruption externe (INTR).

Les interruptions sont classées en deux :

### Les interruptions Externes :

Le µp 8086 contient 3 sources d'interruptions :

1) **Reset** : réinitialisation du µp :

FLAG : 0000h

CS : FFFFh

IP : 0000h

DS : 0000h

ES : 0000h

SS : 0000h

→ L'adresse de démarrage est :

ADR\_START = CS\*10h+IP=FFFF0h.

2) **NMI** : (NO MASKABLE INTERRUPT)

Cette IT interrompt directement la CPU est réservée le vecteur V2.

3) **INTR** : (INTERUPT REQUASTE)

C'est une IT autorisée si IF=1 sinon elle est masquée.

### Les interruptions internes :

Il existe 2 types de IT :

1) **IT Logicielle** : par l'instruction INT :

**Syntaxe :**

INT Numéro\_de\_Vecteur

**Exemple :**

INT 21h

2) **IT Automatiques** :

**Exemple :**

a) dépassement de capacité (OF=1)

INTO (vecteur 04h)

b) division par zéro (vecteur 00h)

c) exécution pas à pas avec TF=1 (Vecteur 01h)

### La relation entre le numéro de vecteur et l'adresse de la table des vecteurs :

Pour obtenir l'adresse physique de vecteur de IT, il faut multiplier le numéro de vecteur par 4.

Exemple :

Le vecteur 02h (NMI) :

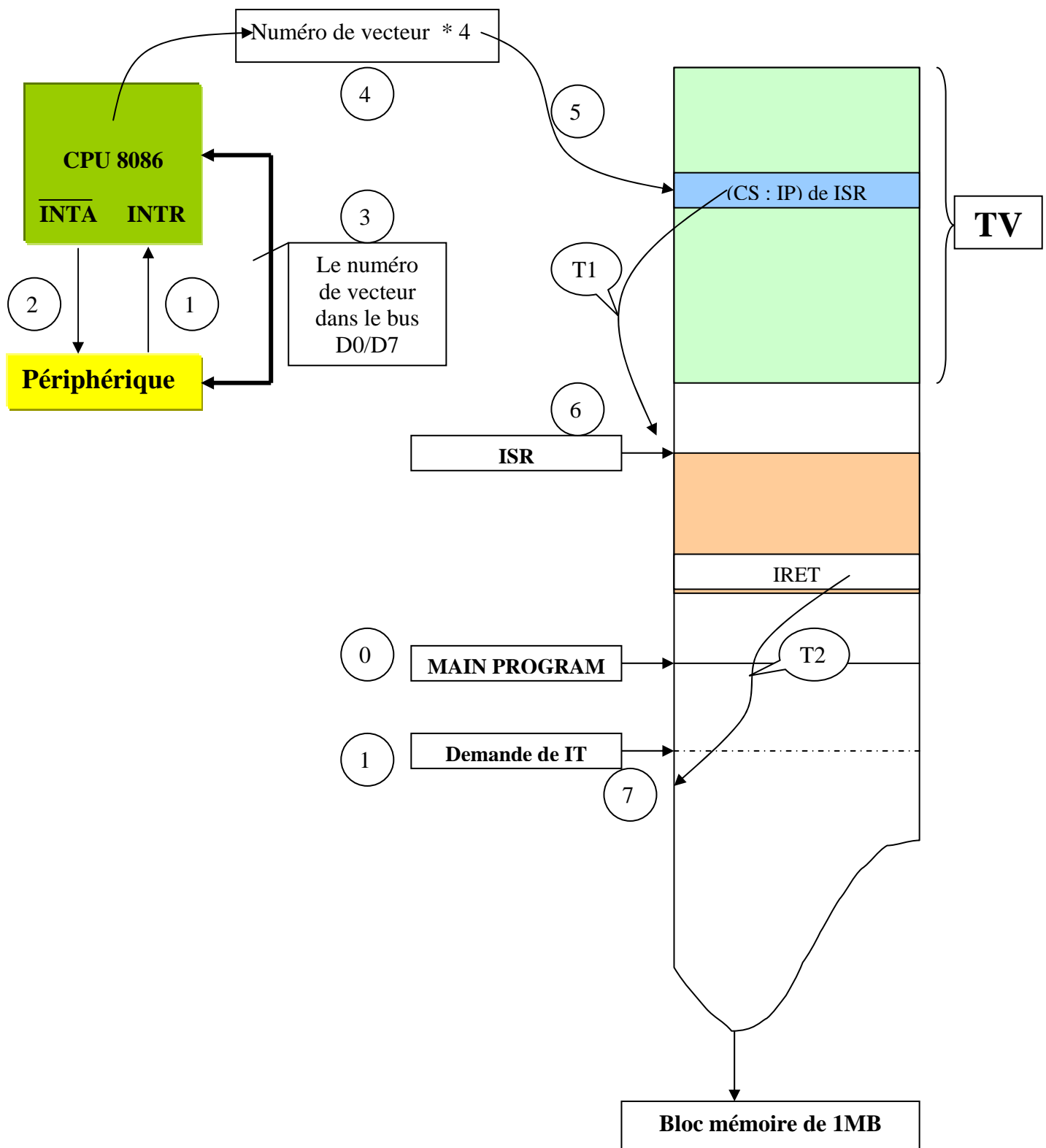
NVECTEUR\*4 = 2\*4=8 → ADR (V2)=00008h dans la table des vecteurs.

### La procédure détaillée de l'exécution d'une IT de type INTR sur µp 8086 :

On remarque que toutes les IT leur numéros de vecteur est visibles sauf la source (INTR). INTEL propose la gestion de une IT Hard par la façon suivante :

- la prise en compte d'une IT de type (INTR) par la CPU est autorisée par (IF=1).
- Complété l'exécution de l'instruction en cours.
- Envoyée 2 impulsions négatives sur la pin (INTA\ ) (INTERUPT ACK) vers le périphérique qui génère le IT.
- Le périphérique correspond reprendre par l'émission de vecteur de IT sur le BUS de données D0/D7.
- Le µp lire le BUS et multiplier par 4 pour générer l'adresse physique de vecteur dans la table de IT.
- Mettre le flag IF=0.
- Sauvegarde les adresses de retours et le FLAG dans la PILE.
- Chargement de CS:IP par l'adresse de ISR.
- Exécution de la routine d'interruption (ISR).
- Après l'exécution la CPU depile les adresses de retour vers le MAIN PROGRAM et active les interruptions par (IF=1).

Schéma synoptique de la procédure de gestion de IT par (INTR) :



T1 : IF = 0, sauvegarde l'adresse de retour et le registre FLAG.

T2 : Dépile le registre FLAG, l'adresse de retour et IF = 1.

**Jeu d'instructions :**

Chaque microprocesseur reconnaît un ensemble d'instructions appelé **jeu d'instructions** (Instruction Set) fixé par le constructeur. Pour les microprocesseurs classiques, le nombre d'instructions reconnues varie entre 75 et 150 (microprocesseurs **CISC** : COMPLEX INSTRUCTION SET COMPUTER). Il existe aussi des microprocesseurs dont le nombre d'instructions est très réduit (microprocesseurs **RISC** : REDUCED INSTRUCTION SET COMPUTER) : entre 10 et 30 instructions, permettant d'améliorer le temps d'exécution des programmes.

Une instruction est définie par son code opératoire, valeur numérique binaire difficile à manipuler par l'être humain. On utilise donc une **notation symbolique** pour représenter les instructions : les **mnémoniques**.

Un programme constitué de mnémoniques est appelé **programme en assembleur**.

Les instructions peuvent être classées en 7 groupes :

- ✚ Instructions de transferts
- ✚ Instructions arithmétiques
- ✚ Instructions logiques
- ✚ Instructions de traitement de chaîne
- ✚ Instructions de saut et d'appel de sous-programmes
- ✚ Instructions des interruptions logicielles
- ✚ Instructions de contrôle du µp

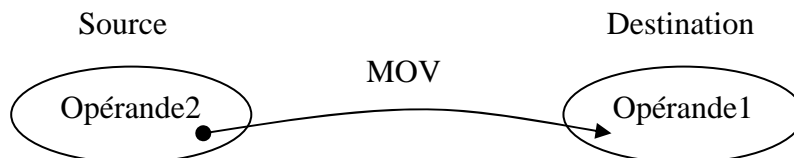
#### A) Instructions de Transferts :

Ces sont les instructions suivantes :

**MOV PUSH PUSHF POP POPF XCHG XLAT LEA LDS LES LAHF SAHF IN OUT**

##### 1) Instruction MOV:

Copie l'opérande2 vers l'opérande1.



##### Syntaxe :

**MOV Destination, Source** ou **MOV Op1, Op2**

La source peut être : Registre, mémoire, valeur immédiate (latérale)

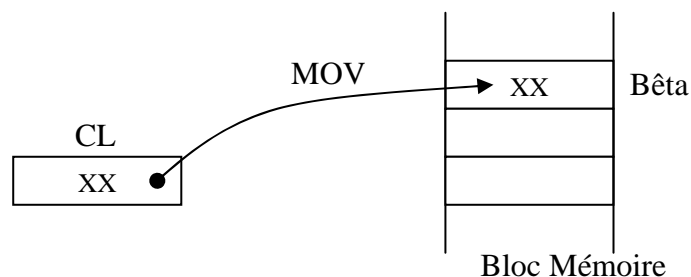
La destination peut être : Registre, mémoire.

##### Exemple :

MOV AL, BL ; Transférez le contenu de BL vers AL

MOV AX, SI ; Transférez le contenu de SI vers AX

MOV Bêta, CL ; Transférez le contenu de CL vers la case mémoire Bêta



##### Des conseils :

- 1- Interdit d'utiliser le registre (CS) comme destination.
- 2- Interdit de lire ou écrire dans le registre (IP).
- 3- Interdit de copier la valeur d'un registre de segment vers un autre registre de segment, il faut passer par une case mémoire ou par un registre interne de 16 bits.
- 4- Interdit de copier une valeur immédiate vers un registre de segment, il faut passer par un registre interne ou une case mémoire de 16 bits.
- 5- Interdit de transférer le contenu d'une case mémoire vers une autre, il faut passer par un registre interne de même taille.

## 2) Instruction PUSH :

Empile une valeur de 16 bits dans le sommet de Pile (STACK SEGMENT).

**Syntaxe :**

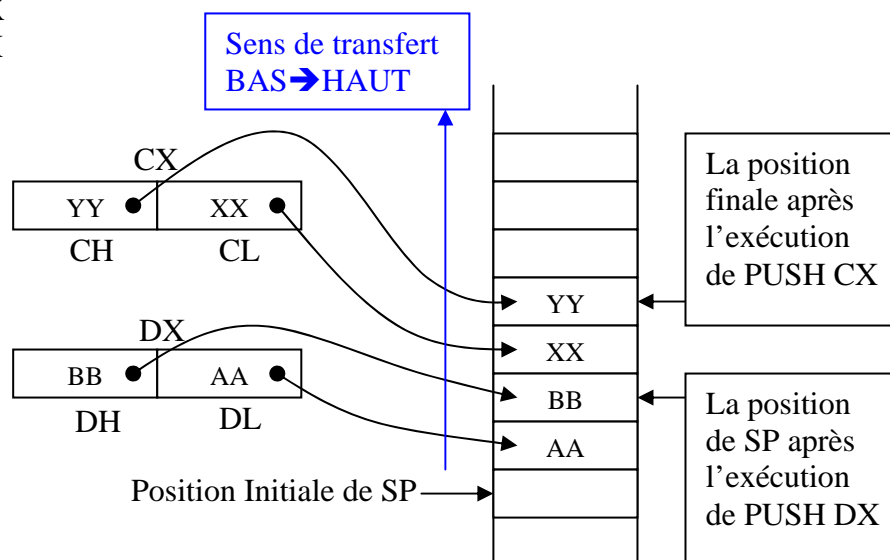
**PUSH Source** ; la source peut être un registre ou une case mémoire de 16 bits

La destination est une case mémoire de 16 bits pointer par SS : [SP].

**Exemple :**

PUSH DX

PUSH CX



Après chaque exécution de l'instruction PUSH la valeur de SP décrémente automatiquement par deux (2)  **$SP \leftarrow SP - 2$** .

## 3) Instruction PUSHF :

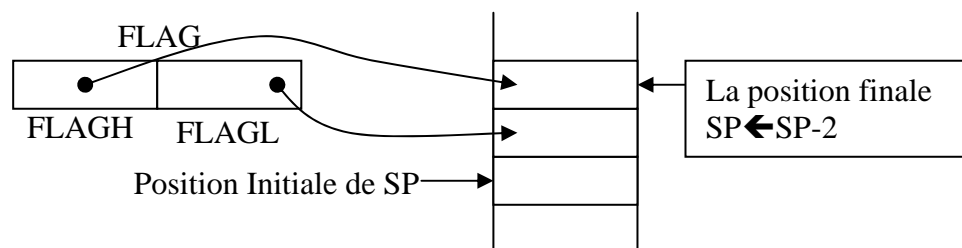
Empilement de registre FLAG vers le sommet de la PILE.

**Syntaxe :**

**PUSHF** ; sans opérande.

La source est le registre FLAG

La destination est une case mémoire de 16 bits pointer par SS : [SP].



## 4) Instruction POP :

Dépile une valeur de 16 bits depuis le sommet de la Pile (STACK SEGMENT)

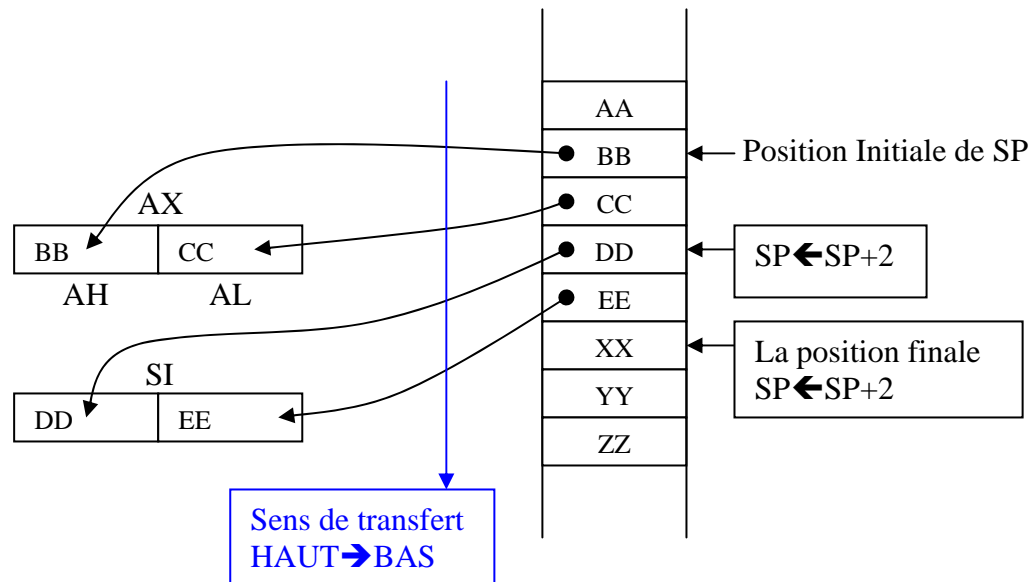
**Syntaxe :**

**POP Destination ;** la destination peut être un registre ou une case mémoire de 16 bits  
La source est une case mémoire de 16 bits pointer par SS : [SP].

**Exemple :**

POP AX

POP SI



Après chaque exécution de l'instruction POP la valeur de SP incrémente automatiquement par deux (2)  **$SP \leftarrow SP+2$** .

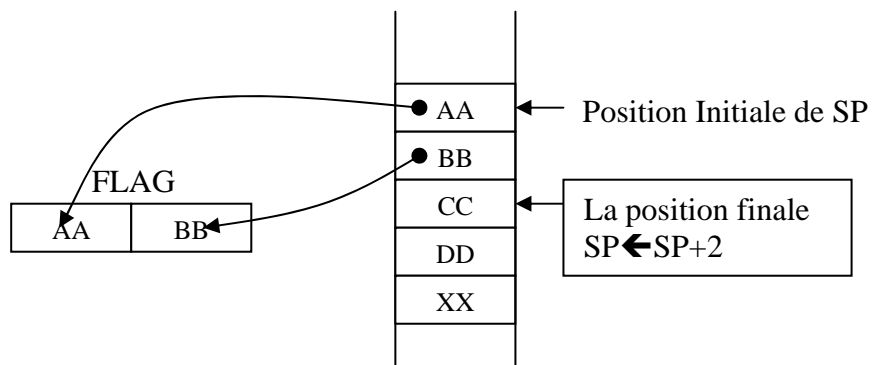
#### 5) Instruction POPF :

Dépilement de registre FLAG depuis le sommet de la Pile (STACK SEGMENT).

**Syntaxe :**

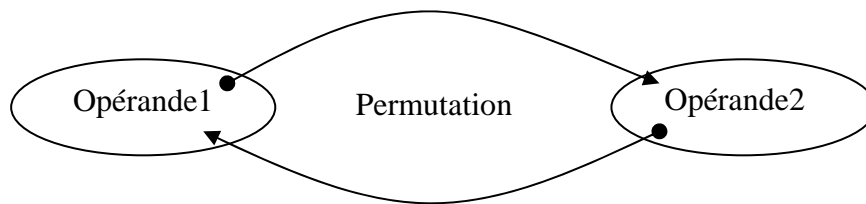
**POPF ;** sans opérande.

La source est une case mémoire de 16 bits pointer par SS : [SP].



#### 6) Instruction XCHG :

Echange les valeurs de deux opérandes.



**Syntaxe :**

**XCHG Op1, Op2**

**Exemple :**

XCHG AL, AH

XCHG Alpha, BX

XCHG DX, Bêta

### 7) Instruction XLAT :

Consultation d'un octet (Byte) depuis une table pointer par DS : [BX+AL].

**Syntaxe d'utilisation:**

On place l'adresse de la table dans le registre BX

MOV BX, OFFSET TABLE

On place l'indice de l'octet à consulter dans AL

MOV AL, indice ;  $0 \leq \text{indice} \leq 255$

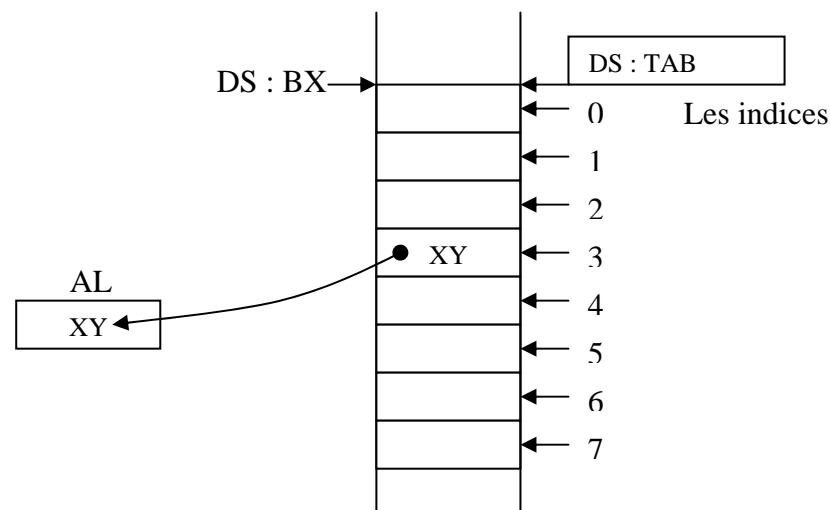
**XLAT ; AL  $\leftarrow$  DS: [BX+AL].**

**Exemple :**

MOV BX, OFFSET TAB

MOV AL, 03h

XLAT



Après l'exécution de l'instruction XLAT la valeur dans AL est écrasé par le contenu de la case mémoire pointer par DS : [BX+AL].

### 8) Instruction LEA :

Chargement de l'adresse effective d'une case mémoire dans un registre 16 bits.

**Syntaxe :**

**LEA registre16, mémoire.**

**Exemple :**

Alpha DB 40h ; déclaration d'une case mémoire dans le DATA SEGMENT  
; Initialisé par 40h

LEA SI, Alpha ... (1)

MOV AL, Alpha ... (2)

Dans (1) : SI contient l'adresse effective de la case Alpha.

Dans (2) : AL contient le contenu de la case Alpha (AL ← 40h).

### 9) Instruction LDS :

Chargement d'un pointeur long (32 bits) à partir d'une case mémoire de 16 bits dans le registre destination de 16 bits et le registre de segment DS.

**Syntaxe :**

**LDS destination, mémoire16.**

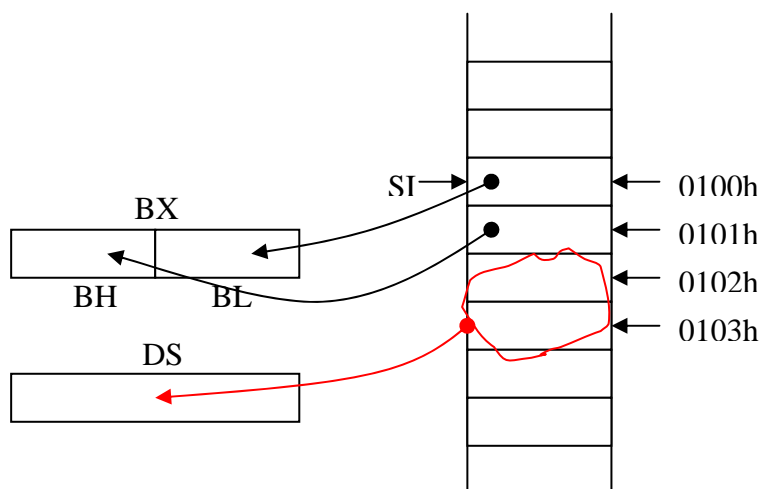
**Exemple :**

MOV SI, 0100h

LDS BX, WORD PTR [SI]

BX ← le contenu de la case mémoire 16 bits DS : [SI].

DS ← le contenu de la case mémoire 16 bits DS : [SI+2].



### 10) Instruction LES :

Chargement d'un pointeur long (32 bits) à partir d'une case mémoire de 16 bits dans le registre destination de 16 bits et le registre de segment ES.

**Syntaxe :**

**LES destination, mémoire16.**

Cette instruction est semblable à l'instruction LDS.

**Exemple :**

LES DI, WORD PTR [BX].

DI ← le contenu de la case mémoire DS : [BX].

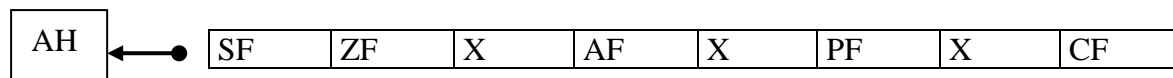
ES ← le contenu de la case mémoire DS : [BX+2].

### 11) Instruction LAHF :

Charge AH par les 8 bits du poids faible de registre FLAG.

**Syntaxe :**

**LAHF** ; sans opérande.



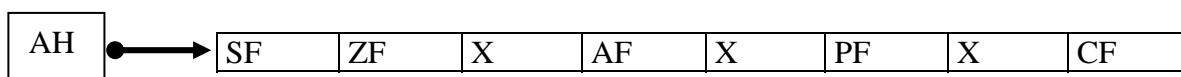
X : non utilisé.

## 12) Instruction SAHF :

Place dans les 8 bits du poids faible de registre FLAG le contenu de AH.

**Syntaxe :**

**SAHF** ; sans opérande



X : non utilisé.

## 13) Instruction IN :

Transfert de la donnée lue dans un port I/O vers l'accumulateur 8 bits (AL) ou 16 bits (AX). Il existe deux modes d'accès, l'accès direct et l'accès indirect.

**Accès direct :**

**Syntaxe :**

**IN AL, PORT** ; lecture d'un port de 8 bits avec  $0 \leq \text{PORT} \leq 255$ .

**IN AX, PORT** ; lecture d'un port de 16 bits avec  $0 \leq \text{PORT} \leq 255$ .

PORT est l'adresse du port sur 8 bits.

**Accès indirect :**

On va mettre l'adresse du port dans le registre DX.

**MOV DX, PORT**

**IN AL, DX** ; lecture d'un port de 8 bits avec  $0000h \leq \text{PORT} \leq FFFFh$ .

**IN AX, DX** ; lecture d'un port de 16 bits avec  $0000h \leq \text{PORT} \leq FFFFh$ .

## 14) Instruction OUT :

Transfert de la valeur dans accumulateur (AL ou AX) vers un PORT I/O.

Il existe deux modes d'accès comme l'instruction IN.

**Accès direct :**

**Syntaxe :**

**OUT PORT, AL** ; écriture dans un port une valeur de 8 bits.

**OUT PORT, AX** ; écriture dans un port une valeur de 16 bits.

PORT est l'adresse du port sur 8 bits  $0 \leq \text{PORT} \leq 255$ .

**Accès indirect :**

On va mettre l'adresse du port dans le registre DX.

**MOV DX, PORT**

**MOV AL, data8**

**OUT DX, AL** ; écriture dans un port une valeur de 8 bits.

**MOV AX, data16**

**OUT DX, AX** ; écriture dans un port une valeur de 16 bits.

PORT est l'adresse du port sur 16 bits  $0000h \leq \text{PORT} \leq FFFFh$ .

## B) Instructions Arithmétiques :

ADD ADC INC SUB SBB DEC CMP NEG MUL DIV IMUL IDIV CBW CWD  
 Instructions d'ajustements ASCII : AAA AAS AAM AAD  
 Instructions d'ajustements Décimales : DAA DAS.

### 1) Instruction ADD :

Addition entre deux opérandes.

#### Syntaxe :

**ADD Destination, Source** ; Destination  $\leftarrow$  destination+source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

#### Exemple :

MOV AL, 03h

MOV BL, 02h

ADD AL, BL ; AL  $\leftarrow$  AL+BL  $\leftrightarrow$  AL=3+2=5

Plus la mise à jour des FLAG : CF $\leftarrow$ 0 ; OF $\leftarrow$ 0 ; AF $\leftarrow$ 0 ; SF $\leftarrow$ 0 ; ZF $\leftarrow$ 0 ; PF $\leftarrow$ 1.

### 2) Instruction ADC :

Addition entre deux opérandes avec carry.

#### Syntaxe :

**ADC Destination, Source** ; Destination  $\leftarrow$  destination+source+CF.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

#### Exemple :

STC ; forcé le CF à « 1 ».

MOV DL, 08h

MOV DH, 00h

ADC DL, DH ; DL  $\leftarrow$  DL+DH+CF  $\leftrightarrow$  DL=8+0+1=9.

Plus la mise à jour des FLAG : CF $\leftarrow$ 0 ; OF $\leftarrow$ 0 ; AF $\leftarrow$ 0 ; SF $\leftarrow$ 0 ; ZF $\leftarrow$ 0 ; PF $\leftarrow$ 1.

### 3) Instruction INC :

Incrémentation d'un opérande.

#### Syntaxe :

**INC Destination** ; Destination  $\leftarrow$  destination+1.

La destination peut être : registre, mémoire.

Les FLAG affectés :

OF	AF	SF	ZF	PF
----	----	----	----	----

CF reste inchangé.

#### Exemple :

MOV Alpha, 0Fh

INC Alpha ; Alpha  $\leftarrow$  Alpha+1  $\leftrightarrow$  Alpha=10h.

Plus la mise à jour des FLAG : CF $\leftarrow$ U ; OF $\leftarrow$ 0 ; AF $\leftarrow$ 1 ; SF $\leftarrow$ 0 ; ZF $\leftarrow$ 0 ; PF $\leftarrow$ 0.

### 4) Instruction SUB :

Soustraction entre deux opérandes.

**Syntaxe :**

**SUB Destination, Source ;** Destination  $\leftarrow$  destination – source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

#### 5) Instruction SBB :

Soustraction entre deux opérandes avec carry.

**Syntaxe :**

**SBB Destination, Source ;** Destination  $\leftarrow$  destination – source – CF.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, valeur immédiate.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

#### 6) Instruction DEC :

Décrémentation d'un opérande.

**Syntaxe :**

**DEC Destination ;** Destination  $\leftarrow$  destination – 1.

La destination peut être : registre, mémoire.

Les FLAG affectés :

OF	AF	SF	ZF	PF
----	----	----	----	----

CF reste inchangé.

#### 7) Instruction CMP :

La comparaison entre deux opérandes.

**Syntaxe :**

**CMP Op1, Op2 ;** Résultat  $\leftarrow$  Op1 – Op2.

Résultat est un registre virtuel, mais le plus important, c'est la mise à jour des FLAG.

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

Op1 peut être : registre, mémoire.

Op2 peut être : registre, mémoire, valeur immédiate.

**Exemple:**

MOV AL, 40h

MOV AH, 30h

CMP AL, AH

Plus la mise à jour des FLAG : CF $\leftarrow$ 0 ; OF $\leftarrow$ 0 ; AF $\leftarrow$ 0 ; SF $\leftarrow$ 0 ; ZF $\leftarrow$ 0 ; PF $\leftarrow$ 0.

On peut dire que AL  $\neq$  AH.

On peut dire que AL plus grand que AH.

#### 8) Instruction NEG :

La négation d'une valeur (changement de signe en complément à deux).

**Syntaxe :**

**NEG Destination** ; Destination  $\leftarrow$  destination \* (-1).

La destination peut être : registre, mémoire.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

Algorithme de la négation en complément à deux :

- 1- inversé tous les bits de l'opérande.
- 2- Additionné « 1 » au résultat précédent.

**Exemple :**

MOV AL, 05h

NEG AL ; AL  $\leftarrow$  AL\*(-1)

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

**Inversion bit à bit**

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

**Addition d'une unité**

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

AL = FBh la représentation de (-5) en complément à 2.

D'une façon générale le calcul de complément à (n) d'une valeur est calculé par l'algorithme suivant :

Le complément à (n)  $\leftarrow$  inversion bit à bit (valeur) + n/2. (Avec (n) est un entier pair).

Exemple de complément à 10 : COM10  $\leftarrow$  NOT (Value) + 05h.

#### 9) Instruction MUL :

Multiplication entre l'accumulateur et un opérande (non signé).

**Syntaxe :**

**MUL Opérande.**

Si l'opérande est une valeur sur 8 bits : AX  $\leftarrow$  AL\*Opérande.

Si l'opérande est une valeur sur 16 bits : DX : AX  $\leftarrow$  AX\*Opérande.

L'opérande peut être : registre, mémoire.

Les FLAG affectés :

CF	OF
----	----

Le ZF, SF, PF, et AF sont inconnus (X).

**Exemple :**

MOV AL, 20h

MOV CL, 02h

MUL CL ; AX  $\leftarrow$  AL\*CL.

AX=20h\*02h=40h.

#### 10) Instruction DIV:

Division entre l'accumulateur et un opérande (non signé).

**Syntaxe :**

**DIV Opérande.**

Si l'opérande sur 8 bits :  $AL \leftarrow AX / \text{Opérande}$  ;  $AH \leftarrow \text{Reste}$ .

Si l'opérande sur 16 bits :  $AX \leftarrow (DX : AX) / \text{Opérande}$  ;  $DX \leftarrow \text{Reste}$ .

L'opérande peut être : registre, mémoire.

Le CF, OF, ZF, SF, PF, et AF sont inconnus (X).

**Exemple :**

MOV DX, 0000h

MOV AX, 2000h

MOV CX, 0002h

DIV CX ;  $AX \leftarrow (DX: AX) / CX$   
 ;  $AX \leftarrow (00002000h) / 2 = 1000h$ .  
 ;  $DX \leftarrow 0000h$ .

### 11) Instructions IMUL et IDIV :

La multiplication et la division de l'accumulateur avec un opérande (signé).

Les mêmes syntaxes que MUL et DIV.

### 12) Instruction CBW :

Conversion de l'accumulateur de 8 bits à un mot de 16 bits signé.

**Syntaxe :**

**CBW** ; sans opérande.

La source est l'accumulateur de 8 bits (AL).

La destination est l'accumulateur de 16 bits (AX).

Les FLAG reste inchangé.

**Exemple :**

MOV AL, 50h ; (SF=0).

CBW ;  $AX \leftarrow 0050h$

MOV AL, 80h ; (SF=1).

CBW ;  $AX \leftarrow FF80h$

### 13) Instruction CWD :

Conversion de l'accumulateur de 16 bits à un mot de 32 bits signé.

**Syntaxe :**

**CWD** ; sans opérande.

La source est accumulateur de 16 bits.

La destination est formée par la concaténation entre les registres DX et AX.

Les FLAG reste inchangé.

**Exemple :**

MOV AX, 4002h ; (SF=0)

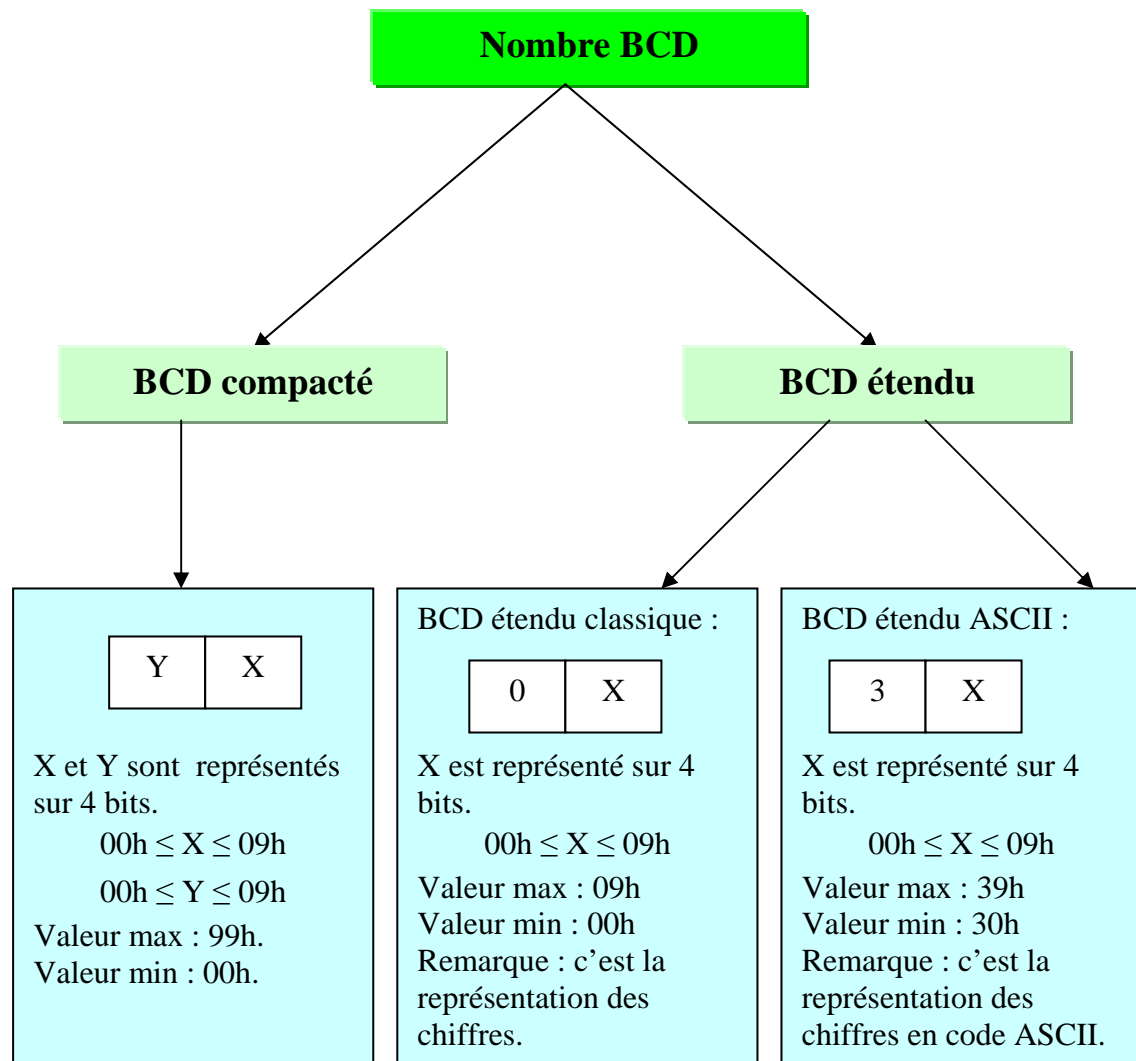
CWD ;  $DX: AX \leftarrow 00004002h$ .

MOV AX, 80FFh ; (SF=1)

CWD ;  $DX: AX \leftarrow FFFF80FFh$ .

## Instructions d'ajustements ASCII

Avant d'étudier ce type d'instructions, il faut passer par la définition des différentes représentations d'un nombre BCD.



#### 14) Instruction AAA :

Ajustement ASCII après addition.

**Syntaxe :**

**AAA** ; sans opérande.

**Algorithme :**

Corrige le résultat dans AL et AH après addition lors d'une opération en valeurs BCD codé ASCII.

**Si** NIBBLE (quartet ou demi octet) de poids faible dans AL > 9 ou AF = 1 alors :

AL = AL + 6;

AH = AH + 1;

AF = 1; CF = 1;

**Sinon**

AF=0; CF=0;

**Fin si.**

Dans les deux cas le NIBBLE de poids fort de (AL) sera effacer.

**Exemple :**

MOV AH, 00h ; effacé AH.  
 MOV AL, '4' ; AL  $\leftarrow$  34h  
 MOV BL, '7' ; BL  $\leftarrow$  37h  
 ADD AL, BL ; AL  $\leftarrow$  34h+37h=6Bh.  
 AAA ; AX  $\leftarrow$  0101h la représentation de la valeur 11 BCD  
 ; étendu classique.

	34h	0	0	1	1	0	1	0	0
+	37h	0	0	1	1	0	1	1	1

= 6Bh 0 1 1 0 1 0 1 1  
 AF=0, mais le NIBBLE de poids faible de AL > 9  $\rightarrow$  on ajoute à AL la valeur 06h.

	6Bh	0	1	1	0	1	0	1	1
+	06h	0	0	0	0	0	1	1	0

= 71h 0 1 1 1 0 0 0 1  
 On va écraser le NIBBLE de poids fort de AL.

	71h	0	1	1	1	0	0	0	1
AND	0Fh	0	0	0	0	1	1	1	1

= 01h 0 0 0 0 0 0 0 1

Et AH  $\leftarrow$  AH+1.

A la fin : AL  $\leftarrow$  01 et AH  $\leftarrow$  01  $\rightarrow$  AX  $\leftarrow$  0101h.

CF=1 et AF=1.

### Remarque :

Les FLAG affectés sont CF, AF ; et ZF, OF, SF, PF restes indéfinis (X).

### 15) Instruction AAS :

Ajustement ASCII après soustraction.

#### Syntaxe :

**AAS** ; sans opérande.

#### Algorithme :

Corrige le résultat dans AL et AH après soustraction lors d'une opération en valeurs BCD codé ASCII.

**Si** NIBBLE (quartet ou demi octet) de poids faible dans AL > 9 ou AF = 1 alors :

AL = AL - 6;

AH = AH - 1;

AF = 1; CF = 1;

#### **Sinon**

AF=0 ; CF=0 ;

#### **Fin si**

Dans les deux cas le NIBBLE de poids fort de (AL) sera effacer.

#### Exemple :

```
MOV AH, 00h
MOV AL, '4'
MOV BL, '7'
SUB AL, BL
AAS          ; AX ← FF07h
              ; C'est le complément à 10 de la valeur 3.
```

#### 16) Instruction AAM :

Ajustement ASCII après multiplication.

**Syntaxe :**

**AAM** ; sans opérande.

**Algorithme :**

Corrige le résultat dans AX en valeur BCD étendu classique après multiplication.

AH ← AL / 10 et AL ← le reste de la division.

**Exemple :**

```
MOV AH, 00h
MOV AL, 04h
MOV BL, 04h
MUL BL      ; AX ← 0010h
AAM         ; AX ← 0106h
              ; C'est la représentation de la valeur BCD étendu classique de 16.
```

Les FLAG affectés sont :

ZF	SF	PF
----	----	----

Le CF, OF, AF reste indéfinis (X).

#### 17) Instruction AAD :

Ajustement ASCII avant division.

**Syntaxe :**

**AAD** ; sans opérande.

**Algorithme :**

Prépare le contenu de (AX) à une division décimale. En effet, elle multiplie par 10 le contenu de AH et l'ajoute au contenu de AL puis met le contenu de AH à zéro → On passe de BCD étendu vers de code binaire (HEXA).

**Exemple :**

```
MOV AX, 0106h ; la valeur 16 en BCD étendu.
AAD           ; AX ← 0010h.
```

Les FLAG affectés :

ZF	SF	PF
----	----	----

Le CF, OF, AF reste indéfinis (X).

### Instructions d'ajustement décimales

### 18) Instruction DAA :

Ajustement décimal après addition.

**Syntaxe :**

**DAA** ; sans opérande.

**Algorithme :**

Corrige le résultat de l'addition de deux valeurs BCD compactées.

**Si** le demi octet de poids faible dans AL > 9 ou AF = 1 alors :

AL = AL + 06h ;

AF = 1 ;

**Fin si.**

**Si** AL > 9Fh ou CF = 1 alors :

AL = AL + 60h;

CF = 1;

**Fin si.**

**Exemple :**

MOV AL, 29h

MOV BL, 35h

ADD AL, BL ; AL ← 5Eh

DAA ; AL ← 64h

29h	0	0	1	0	1	0	0	1
+								
35h	0	0	1	1	0	1	0	1

= 5Eh 0 1 0 1 1 1 1 0  
AF=0; mais le NIBBLE de poids faible de AL > 9 → on ajoute à AL la valeur 06h.

5Eh	0	1	0	1	1	1	1	0
+								
06h	0	0	0	0	0	1	1	0

= 64h 0 1 1 0 0 1 0 0  
AF=1.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

### 19) Instruction DAS :

Ajustement décimal après soustraction.

**Syntaxe :**

**DAS** ; sans opérande.

**Algorithme :**

Corrige le résultat de la soustraction des deux valeurs BCD compactées.

**Si** le demi octet de poids faible dans AL > 9 ou AF = 1 alors :

AL = AL - 6 ; AF = 1 ;

**Si** AL > 9Fh ou CF = 1 alors :

AL = AL - 60h; CF = 1;

**Exemple:**

MOV AL, 65h

MOV BL, 27h

SUB AL, BL ; AL  $\leftarrow$  3Eh

DAS ; AL  $\leftarrow$  38h

65h	0	1	1	0	0	1	0	1
-								
27h	0	0	1	1	0	1	1	1
=	3E	0	0	1	1	1	1	0
AF=1 et le NIBBLE de poids de AL > 9 $\rightarrow$ on soustraire 06h.								
3Eh	0	0	1	1	1	1	1	0
-								
06h	0	0	0	0	0	1	1	0
=	38h	0	0	1	1	1	0	0

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

### C) Instructions Logiques :

## OR XOR AND TSET NOT

Instructions de décalages : SHL SAL SHR SAR

Instructions de rotations : ROL ROR RCL RCR

### 1) Instruction OR :

OU logique entre deux opérandes bit à bit.

**Syntaxe :**

**OR Destination, Source ;** Destination  $\leftarrow$  destination (ou) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAG affectés :

ZF	SF	PF
----	----	----

CF=0 ; OF=0 ; AF=X (X : état inconnu).

**Exemple :**

MOV AL, 01h

MOV gamma, 02h

OR AL, gamma ; AL  $\leftarrow$  AL (ou) gamma = 03h.

### 2) Instruction XOR:

OU Exclusif entre deux opérandes bit à bit.

**Syntaxe :**

**XOR Destination, Source ;** Destination  $\leftarrow$  destination (ou exclusif) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAG affectés :

ZF	SF	PF
----	----	----

CF=0 ; OF=0 ; AF=X (X : état inconnu).

**Exemple :**

MOV DL, 55h

XOR DL, 0FFh ; DL  $\leftarrow$  DL (XOR) 0FFh = 0AAh.

### 3) Instruction AND :

ET logique entre deux opérandes bit à bit.

**Syntaxe :**

**AND Destination, Source ;** Destination  $\leftarrow$  destination (et) source.

La destination peut être : registre, mémoire.

La source peut être : registre, mémoire, immédiate.

Les FLAG affectés :

ZF	SF	PF
----	----	----

CF=0 ; OF=0 ; AF=X (X : état inconnu).

**Exemple :**

MOV CL, 32h

AND CL, 0Fh ; CL  $\leftarrow$  CL (ET) 0Fh = 02h.

### Astuces sur les instructions logiques (OR, XOR, AND) :

1. Forçage à 1 : c'est le OR avec un « 1 ».
2. Masquage à 0 : c'est le AND avec un « 0 ».
3. Basculement : c'est le XOR avec un « 1 ».

**Exercice :**

Donnez à chaque fois la valeur immédiate qui réalise les opérations suivantes :

1. forcez le bit (1) de registre AL.
2. masquez les bits (3→5) de registre AL.
3. basculez les bits (2, 7) de registre AL.

#### 4) Instruction TEST :

ET logique entre deux opérandes positionne uniquement les FLAG.

##### Syntaxe :

**TEST Op1, Op2** ; Résultat ← Op1 (ET) Op2.  
; Plus la mise à jour des FLAG.

Op1 peut être : registre, mémoire.

Op2 peut être : registre, mémoire, immédiate.

Les FLAG affectés :

ZF	SF	PF
----	----	----

CF=0 ; OF=0 ; AF=X (X : état inconnu).

##### Exemple :

MOV AL, 0F2h

MOV BL, 3Dh

TEST AL, BL

F2h	1	1	1	1	0	0	1	0
(AND)								
3Dh	0	0	1	1	1	1	0	1
=								
30h	0	0	1	1	0	0	0	0

ZF=0, SF=0, PF=1, CF=0, OF=0, AF=X.

#### 5) Instruction NOT:

Inversion bit à bit d'un opérande.

##### Syntaxe :

**NOT Destination** ; Destination ← destination.

La destination peut être : registre, mémoire.

##### Exemple :

MOV AL, 45h

NOT AL ; AL ← 0BAh.

### Instructions de Décalages

#### 6) Instructions SHL et SAL :

Décalage arithmétique et logique à gauche.

##### Syntaxe :

**SHL Destination, N**

Si N=1 alors la syntaxe devienne **SHL Destination, 1**.

Sinon le nombre de décalage a effectué sur la destination mettre dans CL :

**MOV CL, N**

**SHL Destination, CL**

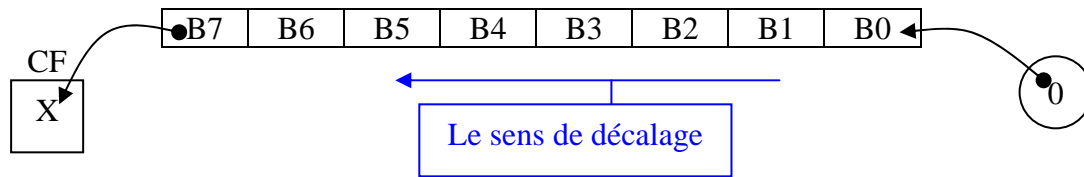
La destination peut être : registre, mémoire.

Les FLAG affectés :

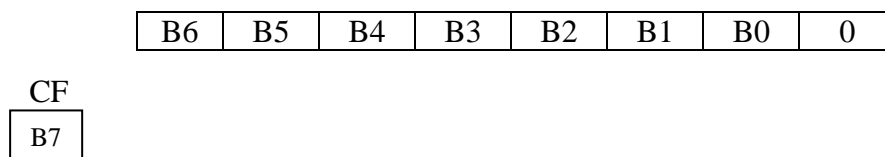
CF	OF
----	----

La synoptique de décalage :

### La destination avant le décalage



### La destination après le décalage



**Astuce :**

Cette instruction réalise l'opération suivante :

**Destination  $\leftarrow$  destination  $\times 2^N$**

### 7) Instruction SAR :

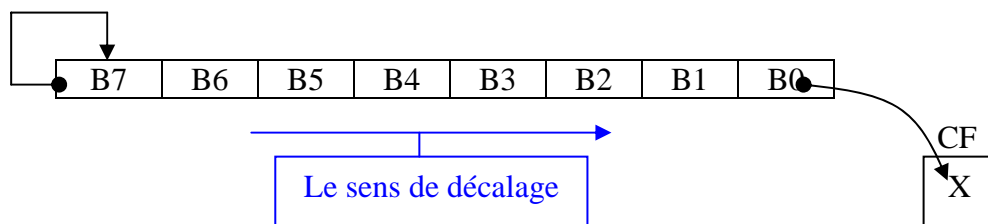
Décalage arithmétique à droite.

**Syntaxe :**

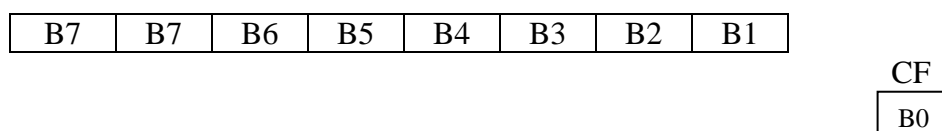
La même syntaxe que SHL. Et les mêmes FLAG affectés.

**La synoptique de décalage :**

### La destination avant le décalage



### La destination après le décalage



Cette opération réalise aussi la fonction suivante :

**Destination  $\leftarrow$  destination  $/ 2^N$  (signé).**

### 8) Instruction SHR :

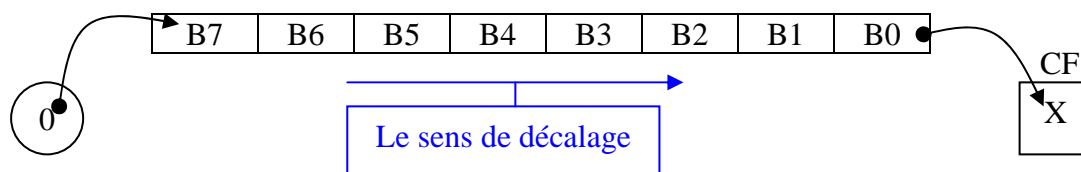
Décalage logique à droite.

**Syntaxe :**

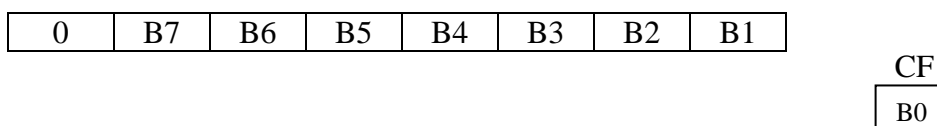
La même syntaxe que SHL. Et les mêmes FLAG affectés.

**Synoptique de décalage :**

La destination avant le décalage



La destination après le décalage



Elle réalise la même fonction que SAR mais pour les nombres non signés.

### Instructions de Rotations

9) Instructions ROL et ROR :

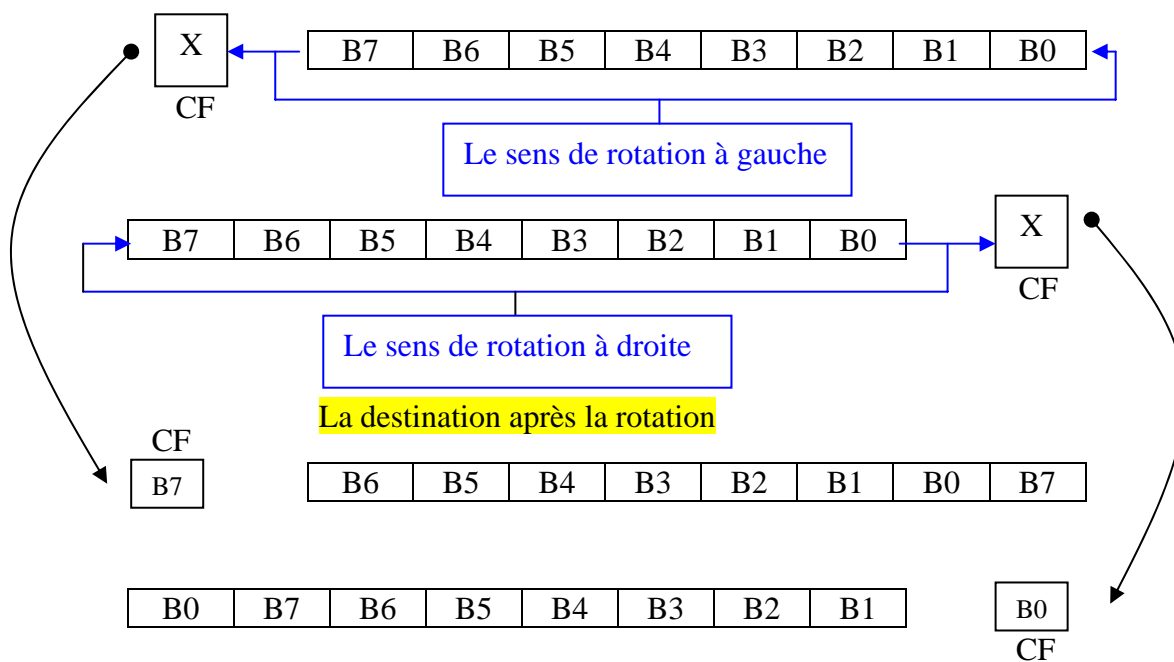
Rotation à gauche et à droite.

**Syntaxe :**

La même syntaxe que SHL. Et les mêmes FLAG affectés.

**Synoptique de rotation :**

La destination avant la rotation



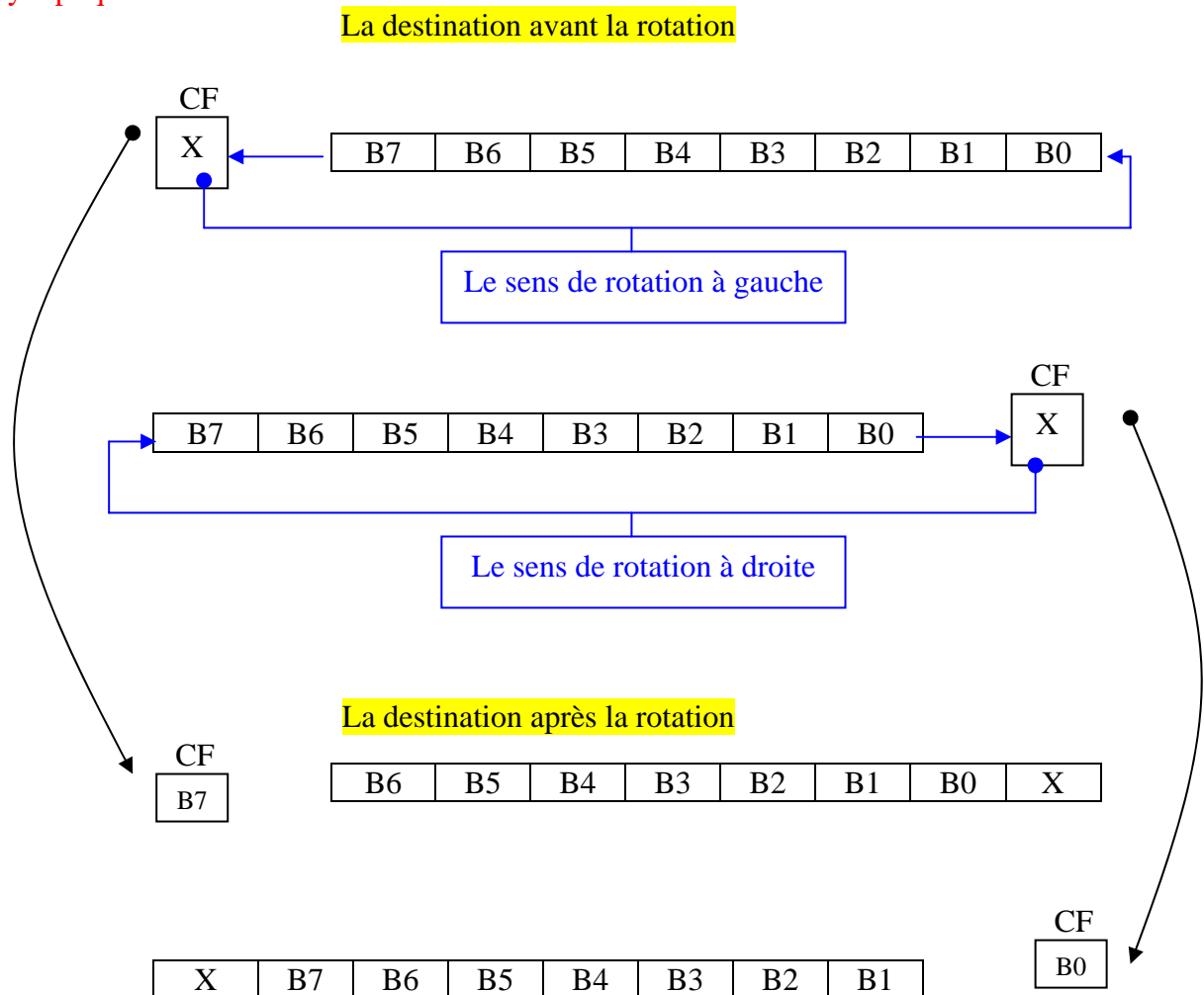
10) Instructions RCL et RCR :

Rotation à gauche et à droite avec carry.

**Syntaxe :**

La même syntaxe que SHL. Et les mêmes FLAG affectés.

**Synoptique de rotation :**



**D) Instructions de traitement de chaîne :**

Une chaîne de caractère est un tableau de type octets ou mots. Les instructions de traitement de chaînes utilisent uniquement l'adressage indexé → les registres utilisés sont : pour la source DS:SI, et pour la destination ES:DI.

A chaque exécution d'une instruction de traitement de chaîne, les registres d'index sont incrémentés ou décrémentés selon le flag DF de registre d'état.

L'ensemble des instructions de traitement de chaîne :

**LODS STOS SCAS MOVS CMPS**

### 1) Instruction LODS (LODSB, LODSW) :

Chargement d'un octet ou un mot depuis DS : [SI] dans l'accumulateur 8 ou 16 bits plus la mise à jour de l'index SI.

**Syntaxe :**

**LODS** ; sans opérande.

**Le cas de AL :**

**LODSB** ; AL ← DS : [SI].  
; SI ← SI±1 selon DF.

**Le cas de AX :**

**LODSW** ; AX ← DS : [SI].  
; SI ← SI±2 selon DF.

Si DF=0 alors, on va incrémenter l'index SI

Sinon on décrémente.

**Exemple :**

**CLD** ; effacé DF

**LEA SI, TAB** ; chargement de l'adresse du TAB dans SI

**LODSB** ; AL ← DS : [SI].  
; SI ← SI+1.

Le registre FLAG reste inchangé.

### 2) Instruction STOS (STOSB, STOSW) :

Copie le contenu de registre accumulateur 8 ou 16 bits dans la case mémoire pointer par ES : [DI] plus la mise à jour de l'index DI selon la valeur de flag DF.

**Syntaxe :**

**STOS** ; sans opérande.

**Le cas de AL :**

**STOSB** ; ES : [DI] ← AL.  
; DI ← DI±1 selon DF.

**Le cas de AX :**

**STOSW** ; ES : [DI] ← AX.  
; DI ← DI±2 selon DF.

Si DF=0 alors, on va incrémenter l'index DI

Sinon on décrémente.

**Exemple :**

**STD** ; la mise à 1 du DF.

**LEA DI, TAB** ; chargement de l'adresse du TAB dans DI.

**MOV AX, 55AAh**

**STOSW** ; ES: [DI] ← AX=55AAh.  
; DI ← DI - 2.

Le registre FLAG reste inchangé.

### 3) Instruction SCAS (SCASB, SCASW) :

Comparaison entre la case mémoire pointer par ES : [DI] avec l'accumulateur, plus la mise à jour de l'indexe DI et le registre d'état.

**Syntaxe :**

**SCAS** ; sans opérande.

**Le cas de AL :**

**SCASB** ; réalisation de l'instruction arithmétique CMP ES : [DI], AL  
 ; DI ← DI±1 selon DF.  
 ; mise à jour de registre d'état.

**Le cas de AX :**

**SCASW** ; réalisation de l'instruction arithmétique CMP ES : [DI], AX  
 ; DI ← DI±2 selon DF.  
 ; mise à jour de registre d'état.

Si DF=0 alors, on va incrémenter l'indexe DI

Sinon on décrémente.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

**Exemple :**

**CLD** ; effacé le DF.  
**MOV DI, offset TAB** ; chargement de l'adresse du TAB dans DI.  
**MOV AL, 'A'** ; chargement de AL par le caractère « A ».  
**SCASB** ; CMP ES : [DI], AX.  
 ; DI ← DI+1.  
 ; mise à jour de registre d'état.

#### 4) Instruction MOVS (MOVSB, MOVSW) :

Copie le contenu de la case mémoire pointer par DS : [SI] dans la case mémoire pointer par ES : [DI], plus la mise à jour les indexes SI et DI.

**Syntaxe :**

**MOVS** ; sans opérande.

**Le cas d'une case mémoire 8 bits :**

**MOVSB** ; ES : [DI] ← DS : [SI].  
 ; SI ← SI±1 selon DF.  
 ; DI ← DI±1 selon DF.

**Le cas d'une case mémoire 16 bits :**

**MOVSW** ; ES : [DI] ← DS : [SI].  
 ; SI ← SI±2 selon DF.  
 ; DI ← DI±2 selon DF.

Si DF=0 alors, on va incrémenter les indexe SI et DI.

Sinon on décrémente.

**Exemple :**

**CLD** ; effacé le DF  
**LEA SI, TABSRC** ; chargement de l'adresse de la table source dans SI.  
**LEA DI, TABDEST** ; chargement de l'adresse de la table destination dans DI.  
**MOVSW** ; ES : [DI] ← DS : [SI].  
 ; SI ← SI+2.  
 ; DI ← DI+2.

Le registre FLAG reste inchangé.

#### 5) Instruction CMPS (CMPSB, CMPSW) :

Comparaison entre la case mémoire pointer par ES : [DI] avec la case mémoire pointer par DS : [SI], plus la mise à jour des indexes SI, DI et le registre d'état.

**Syntaxe :**

**CMPS** ; sans opérande.

**Le cas d'une case mémoire 8 bits :**

**CMPSB** ; réalisation de l'instruction CMP ES : [DI], DS : [SI].  
; mise à jour de registre FLAG.  
; SI ← SI±1 selon DF.  
; DI ← DI±1 selon DF.

**Le cas d'une case mémoire 16 bits :**

**CMPSB** ; réalisation de l'instruction CMP ES : [DI], DS : [SI].  
; mise à jour de registre FLAG.  
; SI ← SI±2 selon DF.  
; DI ← DI±2 selon DF.

Si DF=0 alors, on va incrémenter les indexe SI et DI.

Sinon on décrémente.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF
----	----	----	----	----	----

**Exemple :**

**STD** ; mise à 1 de DF  
**MOV SI, offset TABSRC**  
**MOV DI, offset TABDEST**  
**CMPSB** ; CMP ES : [DI], DS : [SI].  
; mise à jour de registre d'état.  
; SI ← SI-1.  
; DI ← DI-1.

### Les Préfixes de Répétition des Instructions de Traitement de Chaîne

**a) le préfixe REP :**

Ce préfixe utilise le registre CX comme un compteur de répétition.

Les instructions qui utilisent le préfixe REP sont :

**MOVS LODS STOS.**

**Algorithme de fonctionnement :**

Si CX≠0 alors

- 1- on exécute l'instruction de traitement de chaîne.
- 2- CX ← CX - 1.
- 3- Répété.

Sinon

Sort de la boucle de répétition.

Fin si.

**Exemple :**

Transfert de 10 éléments d'un tableau source dans DS vers un tableau destination dans ES.

**CLD**

**MOV CX, 10**

**MOV SI, offset TABSRC**

**MOV DI, offset TABDEST**

**REP MOVSB**

**b) les préfixes REPE et REPZ :**

Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=1 et CX≠0.

**Algorithme de fonctionnement :**

Si CX≠0 alors

- 1- On exécute l'instruction de traitement de chaîne.
- 2- CX ← CX-1.
- 3- Si ZF=1 alors
  - Répété.
  - Sinon
    - Sort de la boucle de répétition.
- Fin si

Sinon

Sort de la boucle de répétition.

Fin si.

**Exemple :**

La recherche d'un élément différent de caractère « A » dans un tableau de caractères.

CLD

MOV CX, TAILLETAB

LEA DI, TAB

MOV AL, 'A'

REPE SCASB

CMP CX, 0 ; test de condition d'arrêt, est ce que CX=0 ou ZF=0.

JE NOT\_FOUND ; saut si égal vers l'étiquette n'existe pas.

**c) les préfixes REPNE et REPNZ :**

Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=0 et CX≠0.

**Algorithme de fonctionnement :**

Si CX≠0 alors

- 1- On exécute l'instruction de traitement de chaîne.
- 2- CX ← CX-1.
- 3- Si ZF=0 alors
  - Répété.
  - Sinon
    - Sort de la boucle de répétition.
- Fin si

Sinon

Sort de la boucle de répétition.

Fin si.

**Exemple :**

La recherche l'un élément dans un tableau.

CLD

MOV DI, offset TAB

MOV AL, 'B'

MOV CX, N

REPNE SCASB

CMP CX, 0 ; test de condition d'arrêt, est ce que CX=0 ou ZF=1.

JE NOT\_FOUND ; saut si égal vers l'étiquette n'existe pas.

**E) Instructions de saut et d'appelle sous programme :**

### E.1) Instruction de saut :

JA ou JNBE	(1)	Saut si « supérieur »	(si $CF + ZF = 0$ ).
JAE ou JNB	(1)	Saut si « supérieur ou égal »	(si $CF = 0$ ).
JB ou JNAE	(1)	Saut si « inférieur »	(si $CF = 1$ ).
JBE ou JNA	(1)	Saut si « inférieur ou égal »	(si $CF + ZF = 1$ ).
JC		Saut en cas de retenue	(si $CF = 1$ ).
JE ou JZ		Saut si « égal » ou « nul »	(si $ZF = 1$ ).
JG ou JNLE	(2)	Saut si « plus grand »	(si $(SF \oplus OF) + ZF = 0$ ).
JGE ou JNL	(2)	Saut si « plus grand ou égal »	(si $SF \oplus OF = 0$ ).
JL ou JNGE	(2)	Saut si « plus petit »	(si $SF \oplus OF = 1$ ).
JLE ou JNG	(2)	Saut si « plus petit ou égal »	(si $(SF \oplus OF) + ZF = 1$ ).
JNC		Saut si « pas de retenue »	(si $CF = 0$ ).
JNE ou JNZ		Saut si « non égal » ou « non nul »	(si $ZF = 0$ ).
JNO		Saut si « pas de dépassement »	(si $OF = 0$ ).
JNP ou JPO		Saut si « parité impaire »	(si $PF = 0$ ).
JNS		Saut si « signe positif »	(si $SF = 0$ ).
JO		Saut si « dépassement »	(si $OF = 1$ ).
JP ou JPE		Saut si « parité paire »	(si $PF = 1$ ).
JS		Saut si « signe négatif »	(si $SF = 1$ ).
JMP		Saut inconditionnel.	

(1) concerne des nombres non signés.

(2) concerne des nombres signés.

### E.2) Instructions d'appelle sous programme :

#### CALL RET

##### 1) Instruction CALL :

Cette instruction permet l'appel d'un sous programme que celui-ci dépende du même segment ou non. Lors d'un appel, le (IP) est stocké en pile après stockage éventuel du contenu de (CS). L'adressage peut être direct, relatif ou indirect.

##### Appel intra segment :

(IP : changé, CS : inchangé).

##### Exemple :

CALL NEAR SUBROUTINE

##### Appel inter segment :

(IP : changé, CS : changé).

##### Exemple :

CALL FAR SUBROUTINE

##### Appel intra segment indirect :

(IP : changé, CS : inchangé).

##### Exemple :

CALL WORD PTR [BX] ; par une case mémoire pointer par BX par rapport au DS.

CALL BX ; par le contenu de registre BX.

##### Appel inter segment indirect:

(IP : changé, CS : changé).

##### Exemple :

CALL DWORD PTR [BX]

IP ← case mémoire pointer par DS : [BX].

CS ← case mémoire pointer par DS : [BX+2].

## 2) Instruction RET :

Cette instruction termine un sous-programme, mais il faut qu'elle corresponde à l'appel, dans ce sens qu'un appel long (inter segment) exige un retour long, et qu'un appel court (intra segment) exige un retour court.

En effet, on peut, à l'issue d'un retour, depuis la pile, soit un retour court. Il est possible de modifier le contenu de SP d'une quantité donnée. Ceci permet le passage de paramètres, à l'aide de la pile, avant l'appel d'un sous-programme.

### Syntaxe :

**RET** ; sans opérande

**RET valeur** ;  $SP \leftarrow SP + \text{valeur}$ .

### Exemple :

Ecrire une procédure qui fait l'émission d'une table de taille N vers un port :

Mettez la taille de la table dans CX

Mettez l'adresse de la table dans SI

Mettez l'adresse du port dans DX

### La solution :

```
Main :      MOV CX, N
              MOV SI, offset TAB
              MOV DX, 0378h
              CALL NEAR SENDBUF
End          Main
```

```
SENDBUF     PROC NEAR
              PUSH CX
              PUSH SI
              PUSH DX
              PUSH AX
              CLD
Encore:      LODSB
              OUT DX, AL
              INC SI
              DEC CX
              CMP CX, 0
              JNZ Encore
              POP AX
              POP DX
              POP SI
              POP CX
              RET
SENDBUF     ENDP
```

## Les boucles :

### 1) Instruction LOOP :

Cette instruction décrémente le contenu de CX et provoque un saut relatif court si ce dernier n'est pas nul.

#### Syntaxe :

**LOOP Etiquette** ; Etiquette est une valeur sur 8 bits signée.

#### Exemple :

L'ajout d'une constante à un tableau.

```

                MOV AL, 20h
                LEA SI, TAB
                MOV CX, N
Encore :       ADD BYTE PTR [SI], AL
                INC SI
                LOOP Encore
    
```

### 2) Instruction LOOPE ou LOOPZ :

Cette instruction décrémente le contenu de CX et provoque un saut relatif court si le flag ZF=1 et si CX ≠ 0 (ou sort de la boucle si ZF=0 ou CX=0).

#### Syntaxe :

**LOOPE Etiquette** ; Etiquette est une valeur sur 8 bits signée.

#### Exemple :

Programme de recherche de premier élément non nul dans un tableau.

```

                MOV CX, N
                MOV SI, offset TAB
Encore :       MOV AL, BYTE PTR [SI]
                INC SI
                CMP AL, 00h
                LOOPE Encore
    
```

### 3) Instruction LOOPNE ou LOOPNZ :

Cette instruction décrémente le contenu de CX et provoque un saut relatif court si le flag ZF=0 et si CX ≠ 0 (ou sort de la boucle si ZF=1 ou CX=0).

#### Syntaxe :

**LOOPNE Etiquette** ; Etiquette est une valeur sur 8 bits signée.

#### Exemple :

Programme de recherche de premier élément nul dans un tableau.

```

                MOV CX, N
                MOV SI, offset TAB
Encore :       MOV AL, BYTE PTR [SI]
                INC SI
                CMP AL, 00h
                LOOPNZ Encore
    
```

## F) Instructions des interruptions logicielles : INT INTO IRET

### 1) Instruction INT:

Appel sous programme qui s'appelle ISR (INTERRUPT SERVICE ROUTINE).

#### Syntaxe :

**INT VEC** ; VEC est un numéro entre 0 et 255.

#### Algorithme d'exécution de INT :

- 1- Sauvegarde de registre FLAG dans la PILE.
- 2- Sauvegarde de registre CS dans la PILE.
- 3- Sauvegarde de registre IP dans la PILE.
- 4- IF = 0
- 5- Appel ISR

#### Exemple :

Affichage d'un caractère dans l'écran de DOS.

MOV AH, 02h

MOV DL, 'A'

INT 21h

### 2) Instruction INTO:

Déclanche l'interruption de vecteur numéro « 4 » si OF=1.

#### Syntaxe :

**INTO** ; sans opérande.

#### Algorithme :

Si OF=1 alors INT 4.

#### Exemple :

MOV AL, -5

SUB AL, 127

INTO

### 3) Instruction IRET:

Cette instruction termine un sous programme de traitement d'une interruption. Les contenus de IP, CS et le registre FLAG sont rechargés depuis la pile. Le pointeur de pile est incrémenté de 6.

#### Syntaxe :

**IRET** ; sans opérande.

#### Algorithme :

- 1- chargement de registre IP depuis la PILE.
- 2- chargement de registre CS depuis la PILE.
- 3- chargement de registre FLAG depuis la PILE.
- 4- IF = 1.
- 5- Retour programme principal.

#### Exemple :

Programmez un ISR qui calcul la somme d'un tableau des octets.

- 1- Mettez la taille du tableau dans CX.
- 2- Mettez l'adresse du tableau dans SI.
- 3- Le résultat sera dans l'accumulateur AX.

Vecteur d'interruption est 40h.

**La solution :**

; Vectorisation

PUSH DS

MOV AX, CS

MOV BX, ISR

MOV DX, 0

MOV DS, DX

MOV SI, 4\*40h

MOV **WORD PTR** [SI], BX

MOV **WORD PTR** [SI+2], AX

POP DS

; Programme principal

MOV SI, **offset** TAB

MOV CX, N

INT 40h ; AX  $\leftarrow$  la somme du TAB.

; Code source de l'ISR.

ISR :        PUSH CX  
               PUSH SI  
               PUSH BX  
               MOV AX, 0  
               MOV BX, 0

Encore:

MOV BL, BYTE PTR [SI]  
 ADD AX, BX  
 INC SI  
 LOOP Encore  
 POP BX  
 POP SI  
 POP CX  
 IRET

### G) Instructions de contrôle du $\mu$ p :

CLC STC CMC CLD STD CLI STI HLT WAIT LOCK NOP ESC

#### 1) Instruction CLC :

Effacer le carry FLAG.

##### Syntaxe :

CLC ; sans opérande  $CF \leftarrow 0$ .

#### 2) Instruction STC :

Mettre le carry FLAG à « 1 ».

##### Syntaxe :

STC ; sans opérande  $CF \leftarrow 1$ .

#### 3) Instruction CMC :

Complémenter le carry FLAG.

##### Syntaxe :

CMC ; sans opérande  $CF \leftarrow \overline{CF}$ .

#### 4) Instruction CLD :

Effacer le FLAG de direction, dans ce cas les indexes (SI, DI) seront incrémenter après chaque instruction de traitement de chaîne.

##### Syntaxe :

CLD ; sans opérande  $DF \leftarrow 0$ .

#### 5) Instruction STD :

Mettre le FLAG de direction à « 1 », dans ce cas les indexes (SI, DI) seront décrémenter après chaque instruction de traitement de chaîne.

##### Syntaxe :

STD ; sans opérande  $DF \leftarrow 1$ .

#### 6) Instruction CLI :

Effacer le FLAG d'interruption matérielle.

##### Syntaxe :

CLI ; sans opérande  $IF \leftarrow 0$ . « Désactivé ».

#### 7) Instruction STI :

Mettre le FLAG d'interruption matérielle à « 1 ».

##### Syntaxe :

STI ; sans opérande  $IF \leftarrow 1$ . « Activé ».

#### 8) Instruction HLT :

Suspendre l'exécution et placer le 8086 dans le mode Sommeille jusqu' une interruption matérielle arrive.

##### Syntaxe :

HLT ; sans opérande.

#### 9) Instruction WAIT :

Cette instruction met le µp en attente jusqu'à la présence d'un signal sur la ligne TEST\ (passant de 5V à 0V). Cet état peut être interrompu par une interruption externe, le retour a lieu dans ce cas, à l'instruction WAIT elle-même. Elle sert à synchroniser le µp sur des circuits externes.

##### Syntaxe :

WAIT ; sans opérande.

#### 10) Instruction LOCK :

Cette instruction est utilisée en **MULTIPROCESSING** pour éviter les conflits, en particulier lors d'accès mémoire. Il s'agit d'un préfixe. La sortie LOCK est mise à zéro pendant la durée de l'instruction qui suit.

#### 11) Instruction NOP :

Pas d'opération, perdre un cycle horloge.

##### Syntaxe :

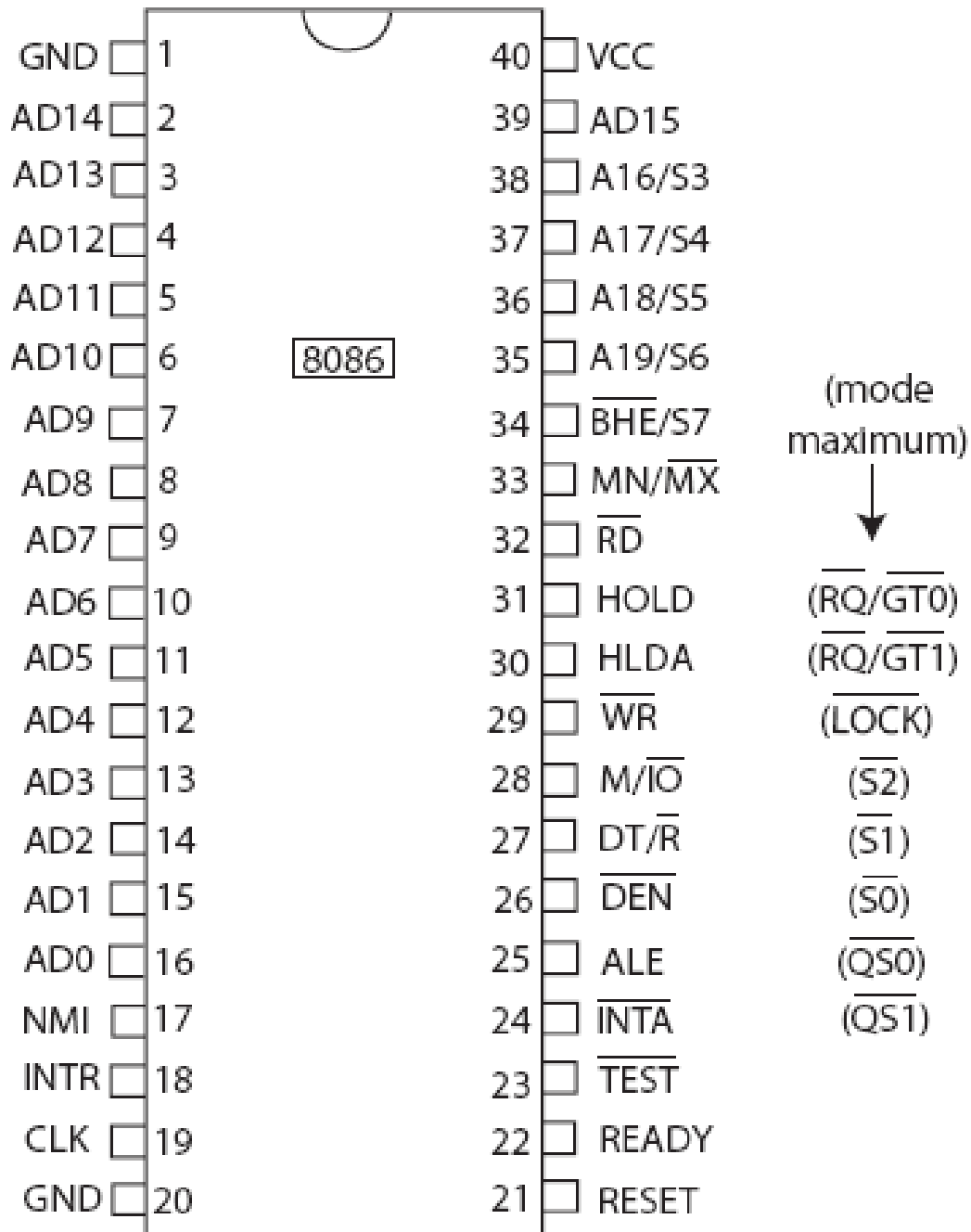
NOP ; sans opérande.

#### 12) Instruction ESC :

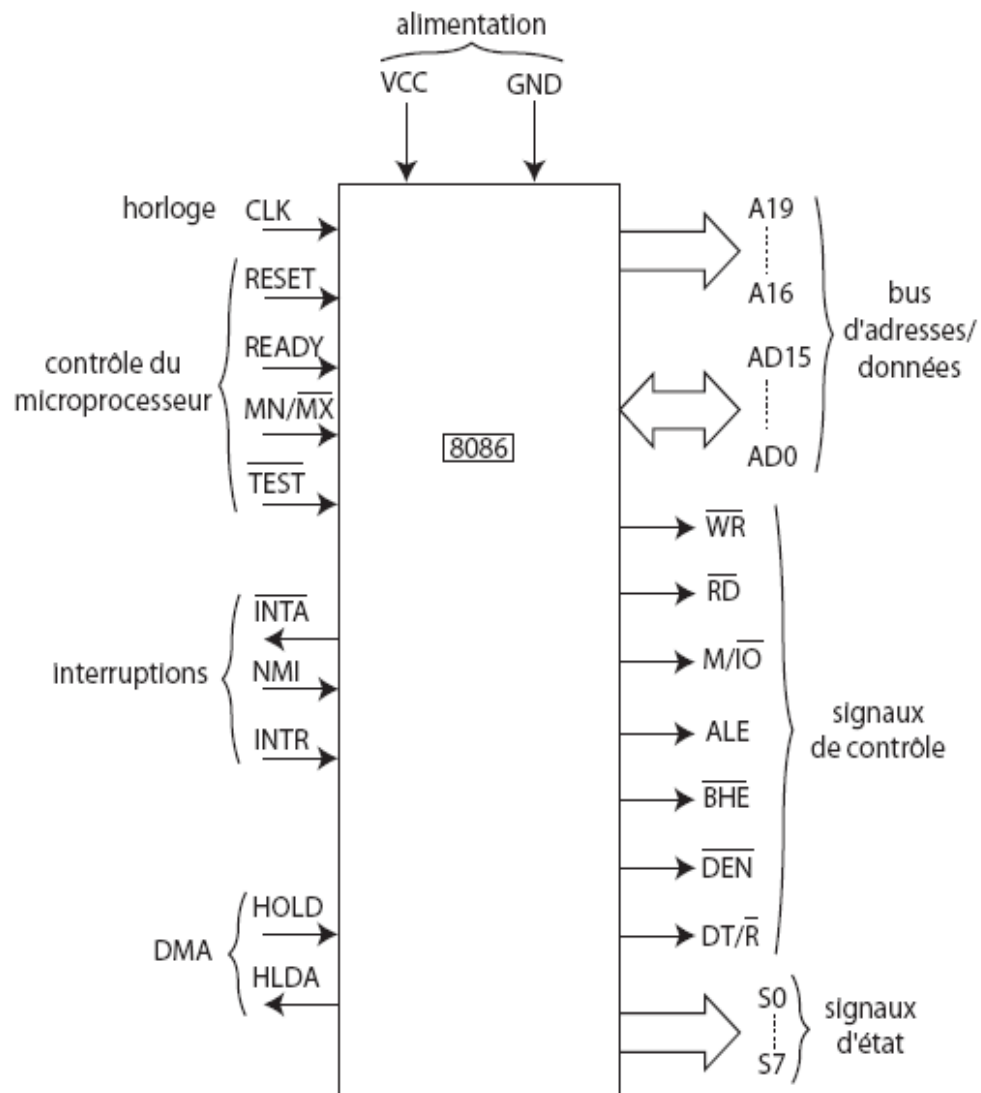
Cette instruction permet d'inclure dans un programme des ordres ne concernant pas le 8086 mais destiné à un autre processeur comme le 8087 ou NPX(**NUMERIC PROCESSOR EXTENSION**).

### 1) Description physique de $\mu$ p 8086 :

Le microprocesseur Intel 8086 est un microprocesseur 16 bits, apparu en 1978. C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...). Il se présente sous la forme d'un boîtier DIP (Dual In Line Package) à 40 broches :



2) Schéma fonctionnel de 8086 :

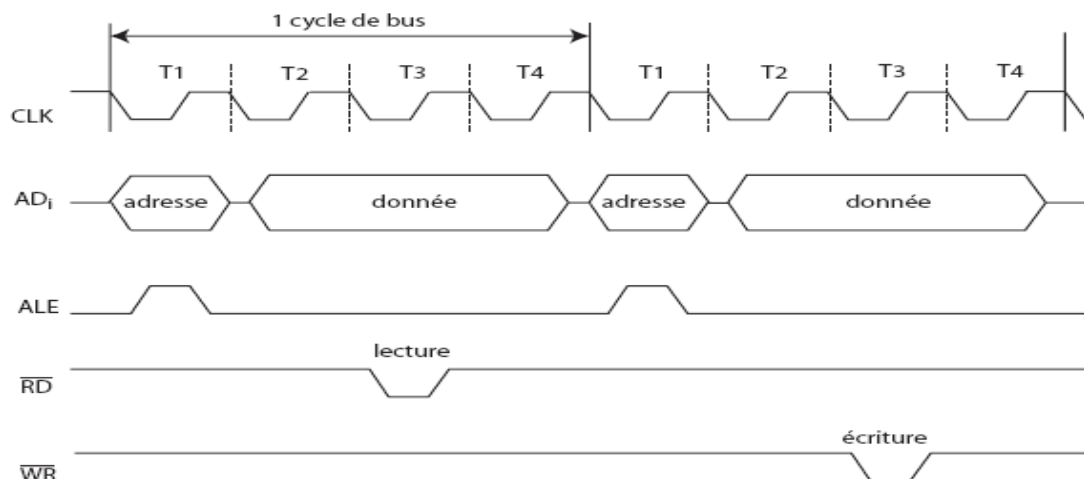


**Description des lignes :**

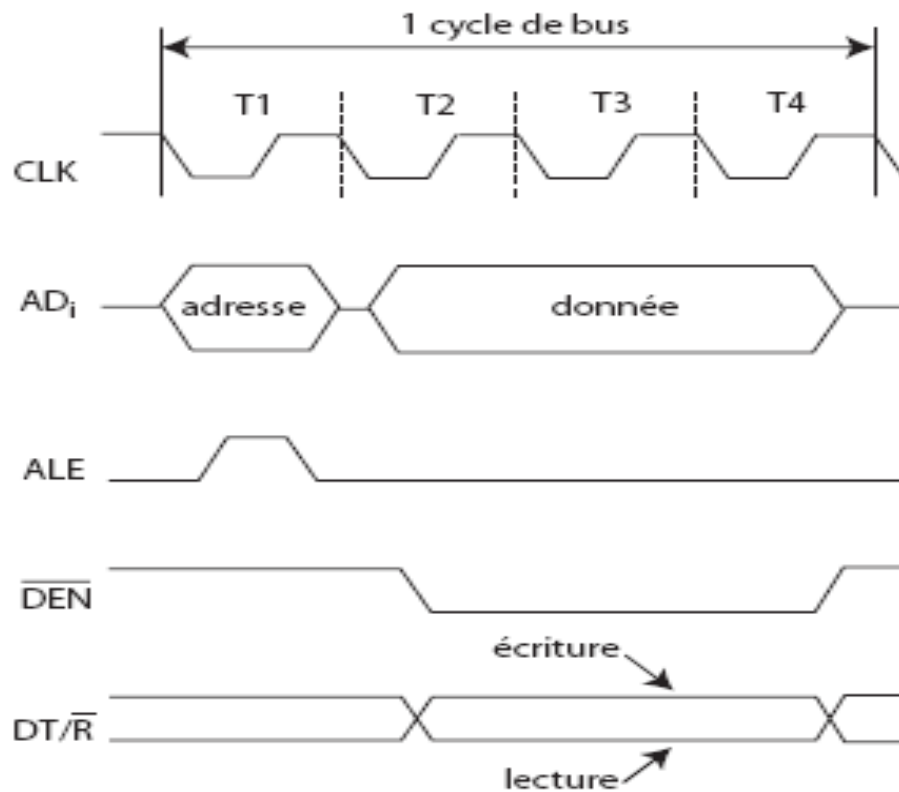
<b>CLK</b>	Entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur.
<b>RESET</b>	Entrée de remise à zéro du microprocesseur.
<b>READY</b>	Entrée de synchronisation avec la mémoire.
<b>TEST\</b>	Entrée de synchronisation du microprocesseur d'un événement extérieur.
<b>MN/MX\</b>	Entrée de choix du mode de fonctionnement du microprocesseur : MN : Mode minimum. MX : Mode maximum.
<b>NMI</b>	Entrée de demande d'interruption non masquée.
<b>INTR</b>	Entrée de demande d'interruption masquée.
<b>INTA\</b>	Sortie indique la réponse à une demande d'interruption.
<b>HOLD et HLDA</b>	Signaux de l'accès direct mémoire par le circuit DMA.
<b>S0 ... S7</b>	Signaux d'état du $\mu$ p en mode STEP BY STEP (pas à pas).
<b>A0 ... A19</b>	Signaux de bus d'adresse de 20 bits (1Mo espace adressable).
<b>D0 ... D15</b>	Signaux de bus de données de 16 bits.
<b>RD\</b>	Signal de demande de lecture.
<b>WR\</b>	Signal de demande d'écriture.
<b>M/IO\</b>	Signal de séparation d'accès mémoire ou port : M/IO\ = 1 $\rightarrow$ accès mémoire. M/IO\ = 0 $\rightarrow$ accès port.
<b>DEN</b>	Sortie indique que l'information qui circule dans bus AD est une donnée.
<b>DT/R\</b>	Sortie indique le sens de transfert des données sur la bus de données : DT/R\ = 1 $\rightarrow$ le bus de donnée en sortie. DT/R\ = 0 $\rightarrow$ le bus de donnée en entrée.
<b>BHE\</b>	Signal d'accès de l'octet du poids fort sur la bus (D8 / D15).
<b>ALE</b>	Sortie indique que l'information qui circule dans bus AD est une adresse.

**Les Chronogrammes d'accès :**

- Chronogramme de séparation de bus AD :

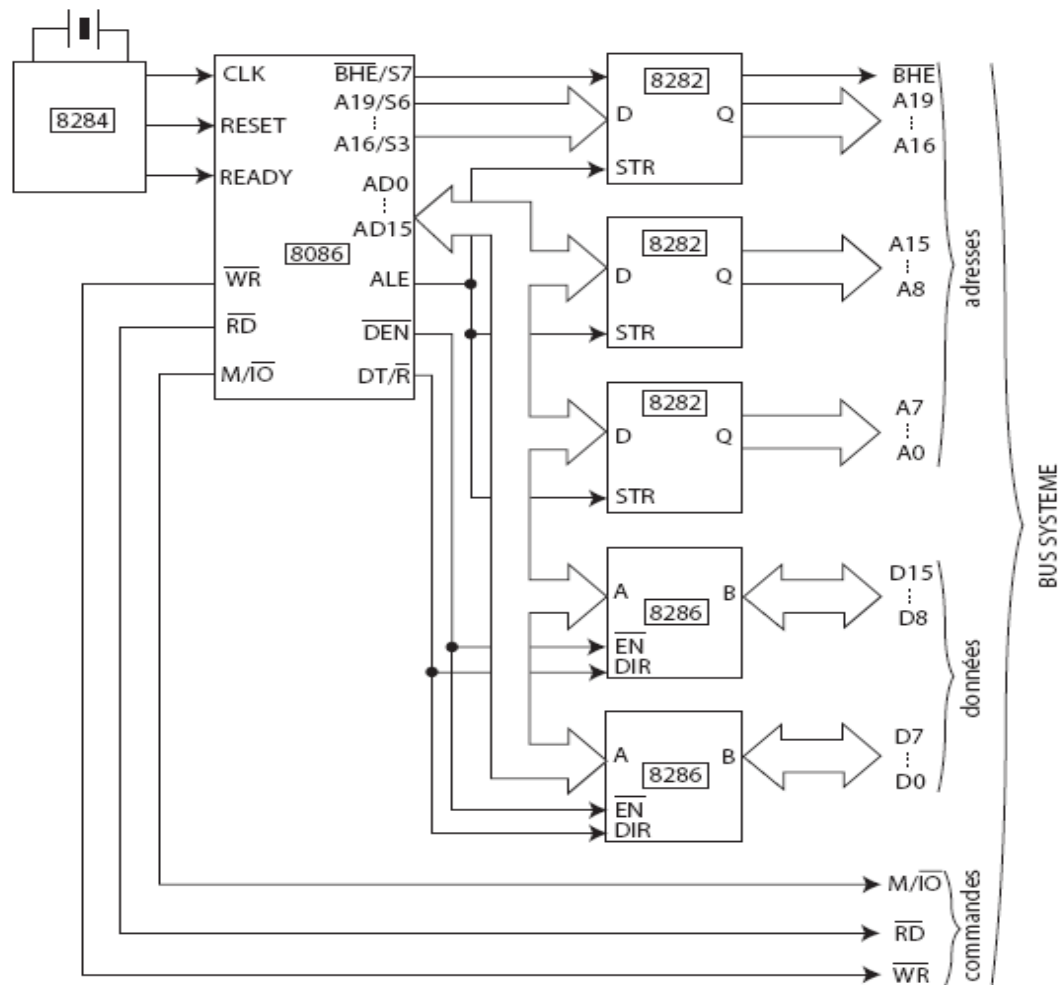


- Chronogramme de sens de transfert de données sur la bus de données.

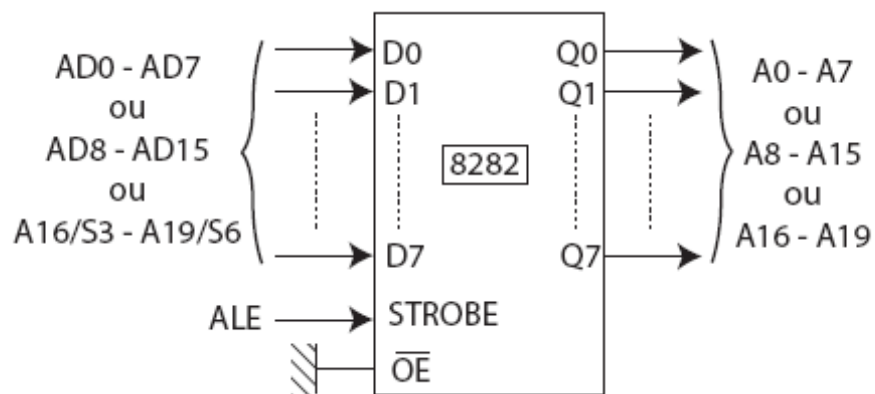


**Remarque** : le 8086 ne peut lire une donnée sur 16 bits en une seule fois, uniquement si l'octet de poids fort de cette donnée est rangé à une adresse impaire et l'octet de poids faible à une adresse paire (alignement sur les adresses paires), sinon la lecture de cette donnée doit se faire en deux opérations successives, d'où une augmentation du temps d'exécution du transfert dû à un mauvais alignement des données.

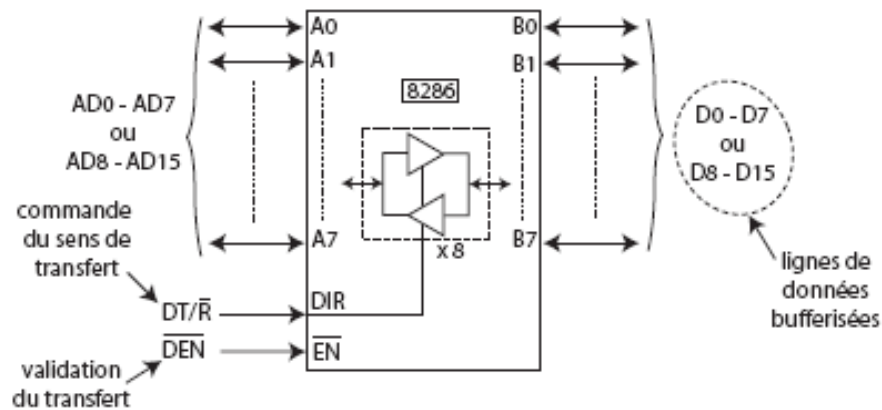
### 3) Conception de bus système de $\mu p$ 8086 :



Exemples de bascules D : circuits 8282, 74373, 74573.



Exemples de tampons de bus : circuits transmetteurs bidirectionnels 8286 ou 74245.



A partir de cette étude de bus système, on va interfacer le bloc mémoire et les périphériques d'interfaçages qui vont étudier par la suite.

#### 4) Interfaçage mémoires :

Une mémoire est un circuit à semi-conducteur permettant d'enregistrer, de conserver et de restituer des informations (instructions et variables). C'est cette capacité de mémorisation qui explique la polyvalence des systèmes numériques et leur adaptabilité à de nombreuses situations. Les informations peuvent être écrites ou lues. Il y a écriture lorsqu'on enregistre des informations en mémoire, lecture lorsqu'on récupère des informations précédemment enregistrées.

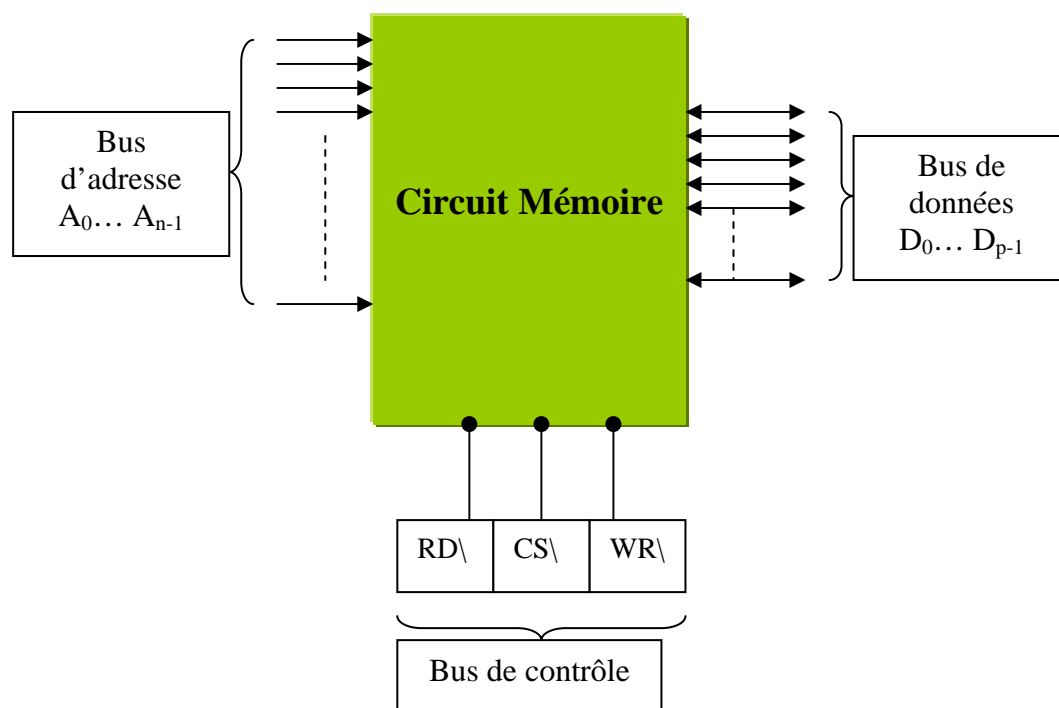
#### Organisation d'une mémoire :

Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs. Chaque tiroir représente alors une case mémoire qui peut contenir un seul élément : des **données**. Le nombre de cases mémoires pouvant être très élevé, il est alors nécessaire de pouvoir les identifier par un numéro. Ce numéro est appelé **adresse**. Chaque donnée devient alors accessible grâce à son adresse.

Avec une adresse de  $n$  bits il est possible de référencer au plus  $2^n$  cases mémoire. Chaque case est remplie par un mot de données (sa longueur  $m$  est toujours une puissance de 2). Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.

On peut donc schématiser un circuit mémoire par la figure suivante où l'on peut distinguer :



$RD\backslash$  : signal de lecture case mémoire.

$WR\backslash$  : signal d'écriture case mémoire.

$CS\backslash$  : signal de validation de circuit mémoire ou boîtier.

### Caractéristiques d'une mémoire :

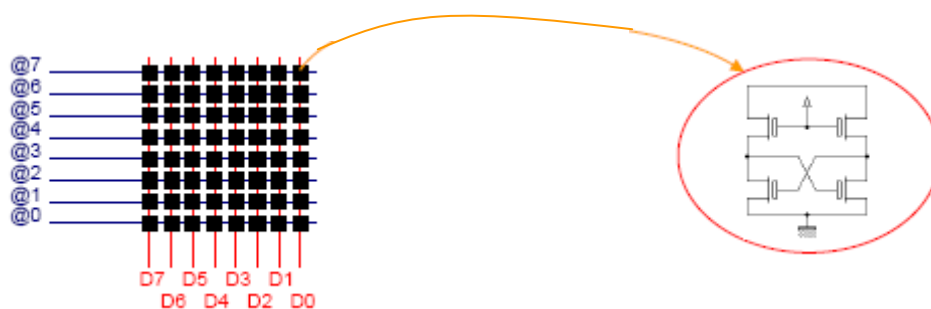
Capacité	C'est le nombre total de bits. Elle s'exprime aussi souvent en octet.
Format de données	C'est le nombre de bits que l'on peut mémoriser par case mémoire. On dit aussi que c'est la largeur du mot mémorisable.
Temps d'accès	C'est le temps écoulé entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
Temps de cycle	Il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.
Débit	C'est le nombre maximum d'informations lues ou écrites par seconde.
Volatilité	Elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

### Différents types de mémoire :

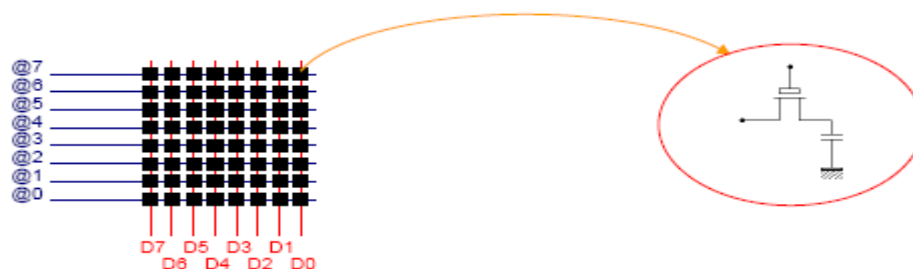
#### Mémoires vives (RAM) :

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure d'alimentation. Certaines d'entre elles, ayant une faible consommation, peuvent être rendues non volatiles par l'adjonction d'une batterie. Il existe deux grandes familles de mémoires RAM (RANDOM ACCESS MEMORY : mémoire à accès aléatoire) :

- Les RAM statiques.
  - Les RAM dynamique.
- 1- **Les RAM statiques** : le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule, chaque bascule contient entre 4 et 6 transistors.



- 2- **Les RAM dynamique** : dans cette mémoire, l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur.



### Avantages :

Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite.

### Inconvénients :

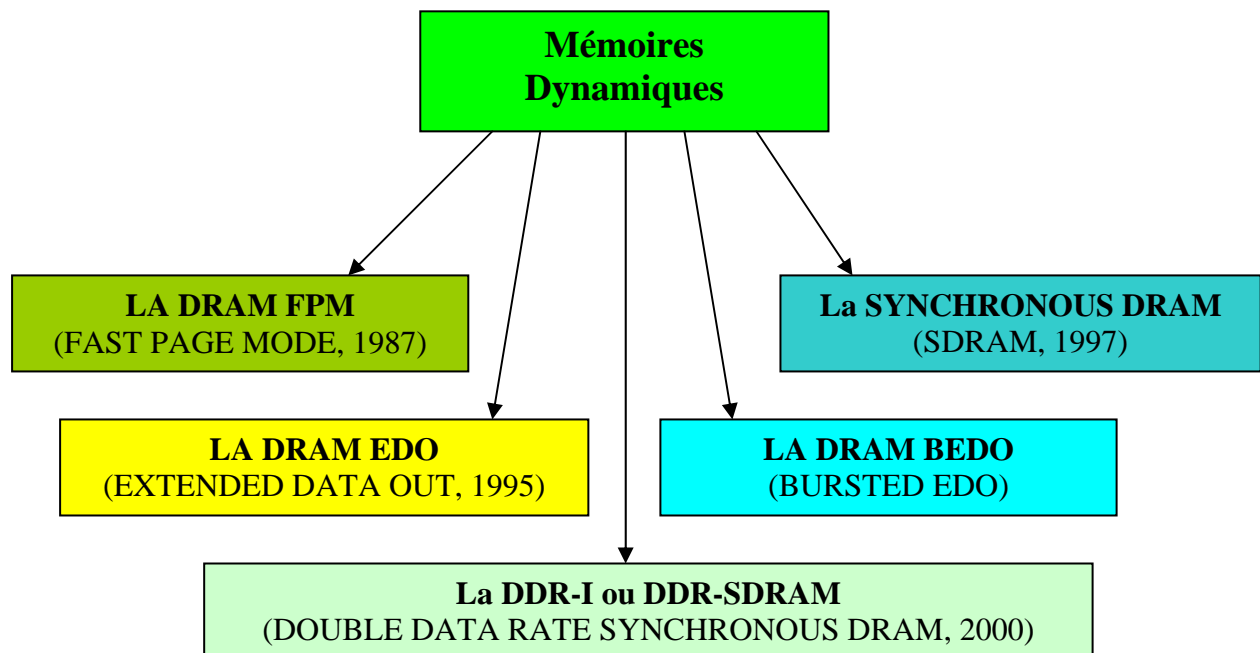
La présence de courants de fuite dans le condensateur contribue à sa décharge. Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur). Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger. Ce rafraîchissement indispensable a plusieurs conséquences :

- Il complique la gestion des mémoires dynamiques car il faut tenir compte des actions de rafraîchissement qui sont prioritaires.
- La durée de ces actions augmente le temps d'accès aux informations.

D'autre part, la lecture de l'information est destructive. En effet, elle se fait par décharge de la capacité du point mémoire lorsque celle-ci est chargée. Donc toute lecture doit être suivie d'une réécriture.

### Conclusions :

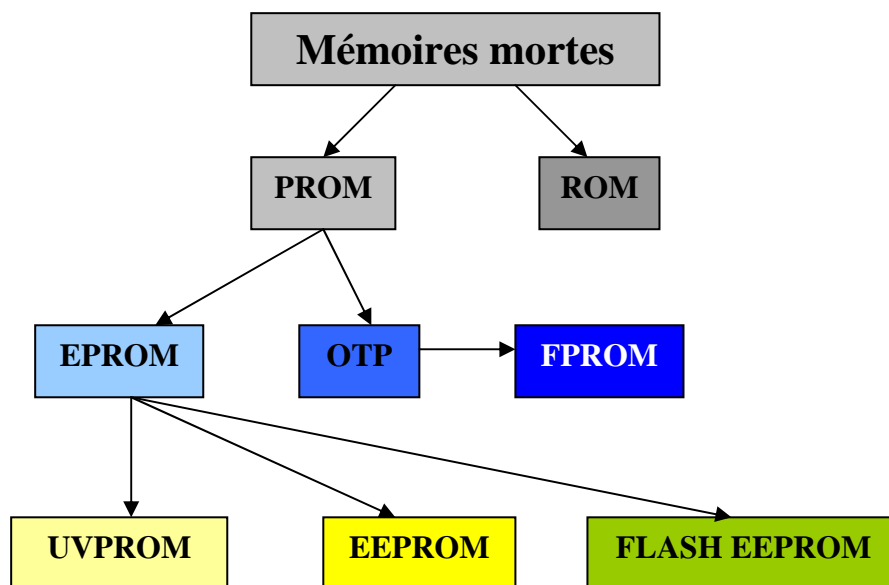
En général les mémoires dynamiques, qui offrent une plus grande densité d'information et un coût par bit plus faible, sont utilisées pour la mémoire centrale, alors que les mémoires statiques, plus rapides, sont utilisées lorsque le facteur vitesse est critique, notamment pour des mémoires de petite taille comme les caches et les registres.



### Les mémoires mortes (ROM) :

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule (ROM : READ ONLY MEMORY). Ces mémoires sont non volatiles.

Ces mémoires, contrairement aux RAM, ne peuvent être que lues. L'inscription en mémoire des données reste possible mais est appelée programmation. Suivant le type de ROM, la méthode de programmation changera. Il existe donc plusieurs types de ROM :

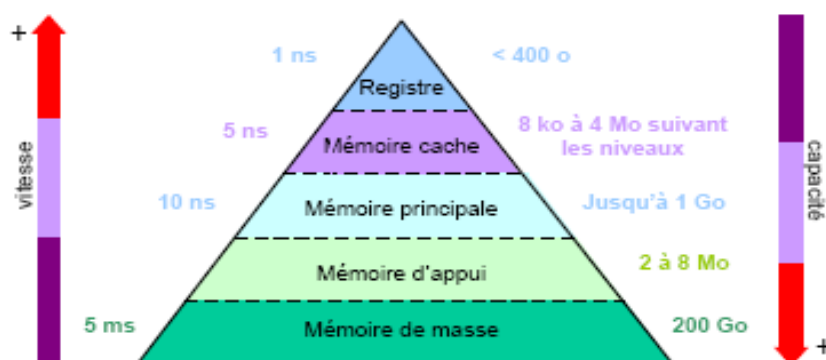


### Critères de choix d'une mémoire :

Les principaux critères à retenir sont :

- Capacité.
- Vitesse.
- Consommation.
- Coût.

Pyramide de vitesse et de capacité des éléments de stockages d'informations :



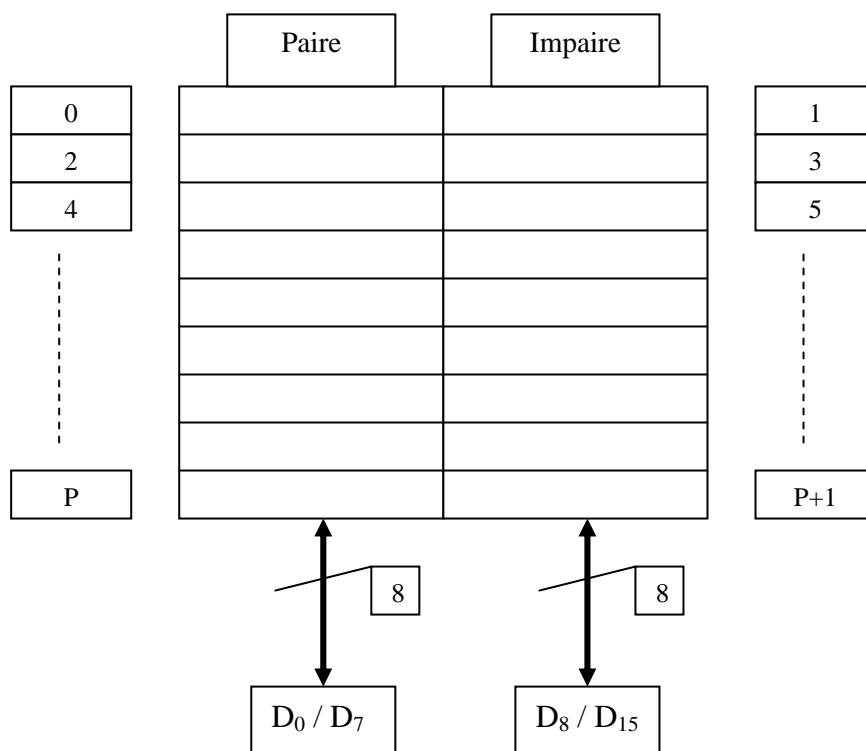
- **Les registres** sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- **La mémoire cache** est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- **La mémoire principale** est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.
- **La mémoire d'appui** sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- **La mémoire de masse** est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

Après cette longue définition des différents types de mémoires, on passe au cœur du cours, c'est l'interface d'un circuit mémoire au bus système 8086.

Avant de commencer, il faut d'abord citer les différents lignes utilisées pour interfacier un boîtier mémoire :

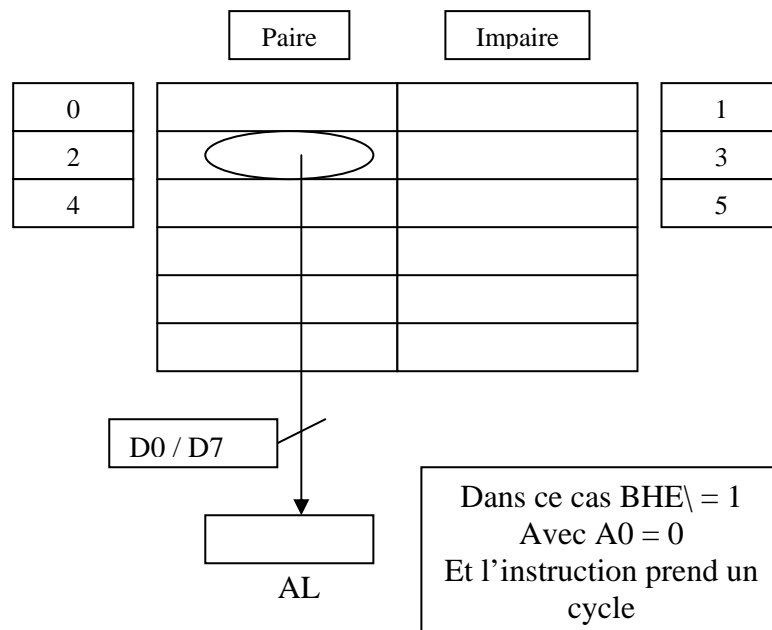
- Les lignes d'adresses  $A_0 \dots A_{19}$ .
- Les lignes de données  $D_0 \dots D_{15}$ .
- Le signal  $\text{BHE}\backslash$ .
- Le signal  $\text{M}/\text{IO}\backslash$ .
- Le signal  $\text{RD}\backslash$ .
- Le signal  $\text{WR}\backslash$ .

A partir de la taille de bus d'adresse de 20 bits, on distingue que l'espace mémoire adressable par le  $\mu\text{p}$  8086 est de capacité  $1 \text{ Mo} = 2^{20} * 8$ . Cet espace est divisé en deux parties, partie paire de 512Ko et partie impaire de 512Ko. La partie paire est connectée au bus de données  $D_0/D_7$ , et la partie impaire est connectée au bus  $D_8/D_{15}$ .

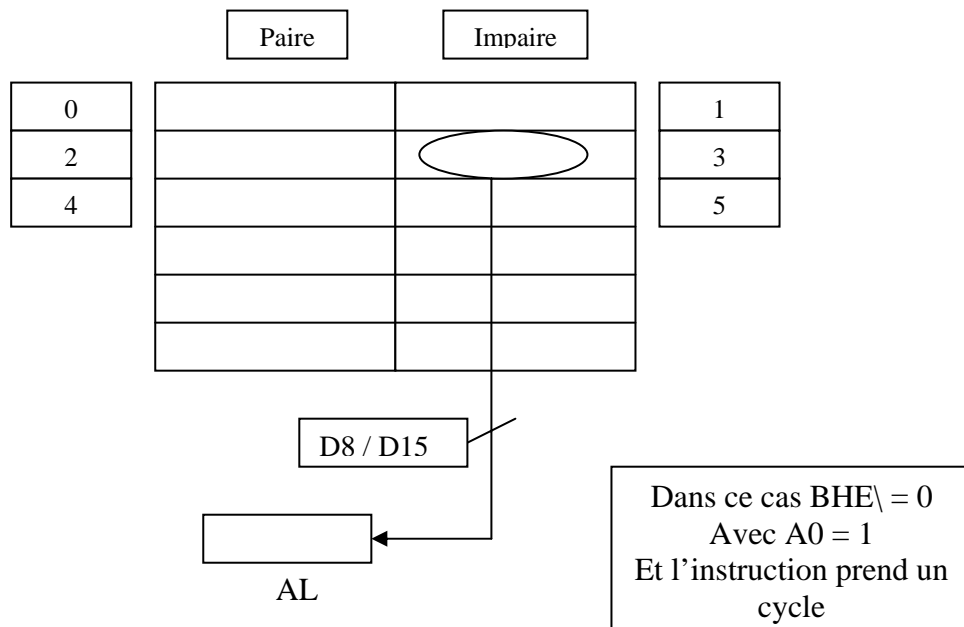


Pour bien comprendre le fonctionnement de la ligne BHE $\setminus$ , on exécute les codes sources suivants :

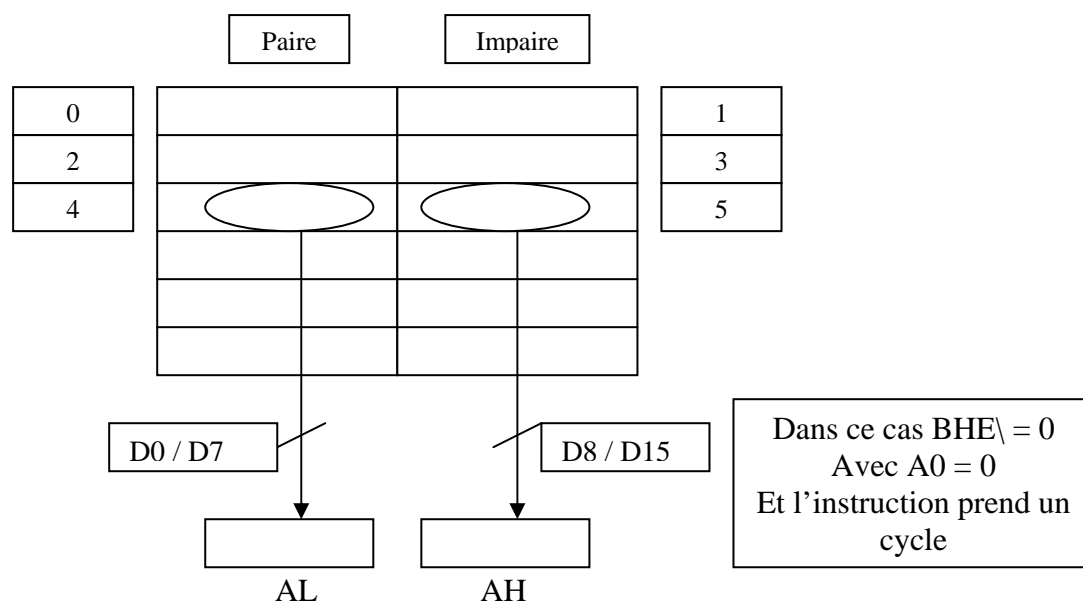
MOV AL, (0002h)



MOV AL, (0003h)



MOV AX, (0004h)



A partir de ces exemples en constate le tableau d'accès suivant :

BHE\	A0	Accès
0	0	16 bits adresse paire
0	1	8 bits adresse impaire
1	0	8 bits adresse paire
1	1	Illégale

Il reste l'accès 16 bits adresse impaire, dans ce cas, on prend l'exemple :

MOV AX, (0001h)

Par analogie AL sera chargé par le contenu de l'adresse (0001h) et AH par le contenu de l'adresse (0002h), mais on remarque que ADR (0001h) est une adresse impaire → on prend un cycle pour charger AL, et un autre cycle pour charger AH.

AL ← (0001h) : A0 = 1 et BHE\ = 0. (1 cycle).

AH ← (0002h) : A0 = 0 et BHE\ = 1. (1 cycle).

On tous deux cycles.

On passe maintenant après cette description de la ligne BHE\ à la liaison d'un bloc mémoire de taille 1Ko à l'adresse physique (00000h) à l'aide les boîtiers de RAM de taille 256\*8.

Avant d'interfacer ce bloc, on va suivre les étapes suivantes :

- 1- Calculez le nombre de boîtiers utilisés pour réaliser le bloc mémoire demandé.

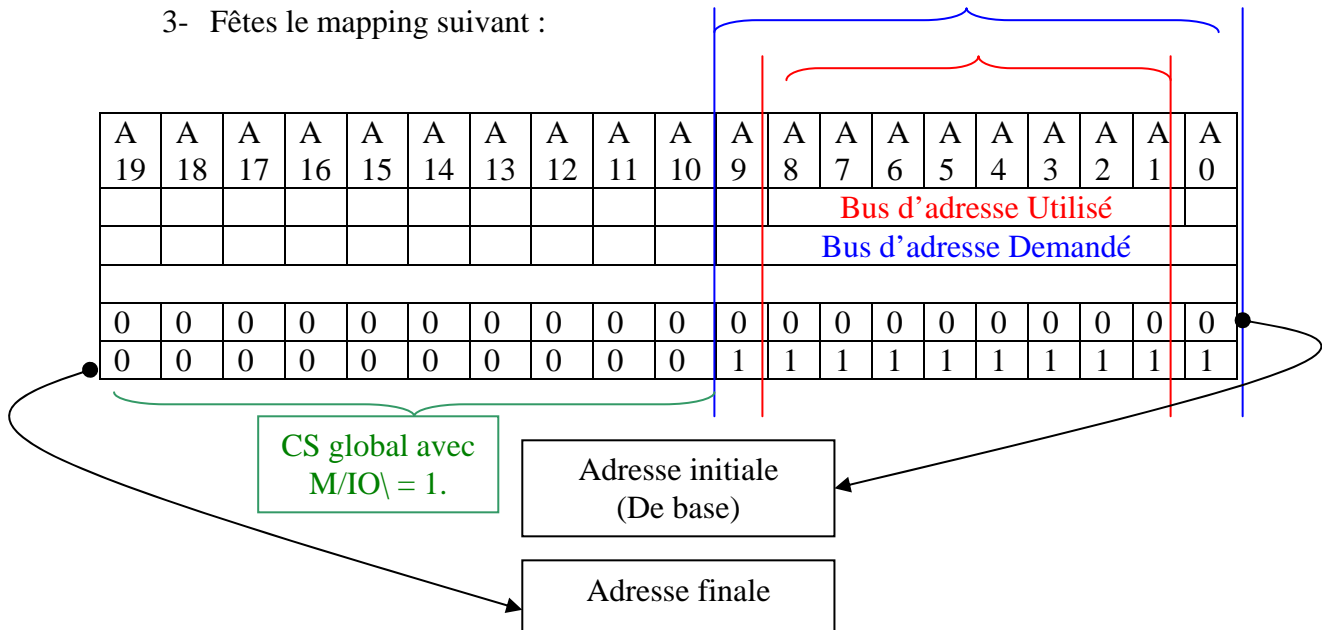
$$\text{Nombre de boîtiers} = \frac{1 * 1024 * 8}{256 * 8} = 4.$$

2- Calculez le nombre de lignes d'adresse du bloc demandé et de boîtier utilisé.

$$\text{Nombre de lignes d'adresse demandé} = \frac{\lg(1*1024)}{\lg(2)} = 10 \text{ lignes.}$$

$$\text{Nombre de lignes d'adresse utilisé} = \frac{\lg(256)}{\lg(2)} = 8 \text{ lignes.}$$

3- Faites le mapping suivant :



Comment calculer l'adresse finale : tous simplement par la relation suivante :

Adresse finale = adresse initiale + (valeur max sur le nombre de lignes d'adresse de bloc demandé).

Dans notre exemple :

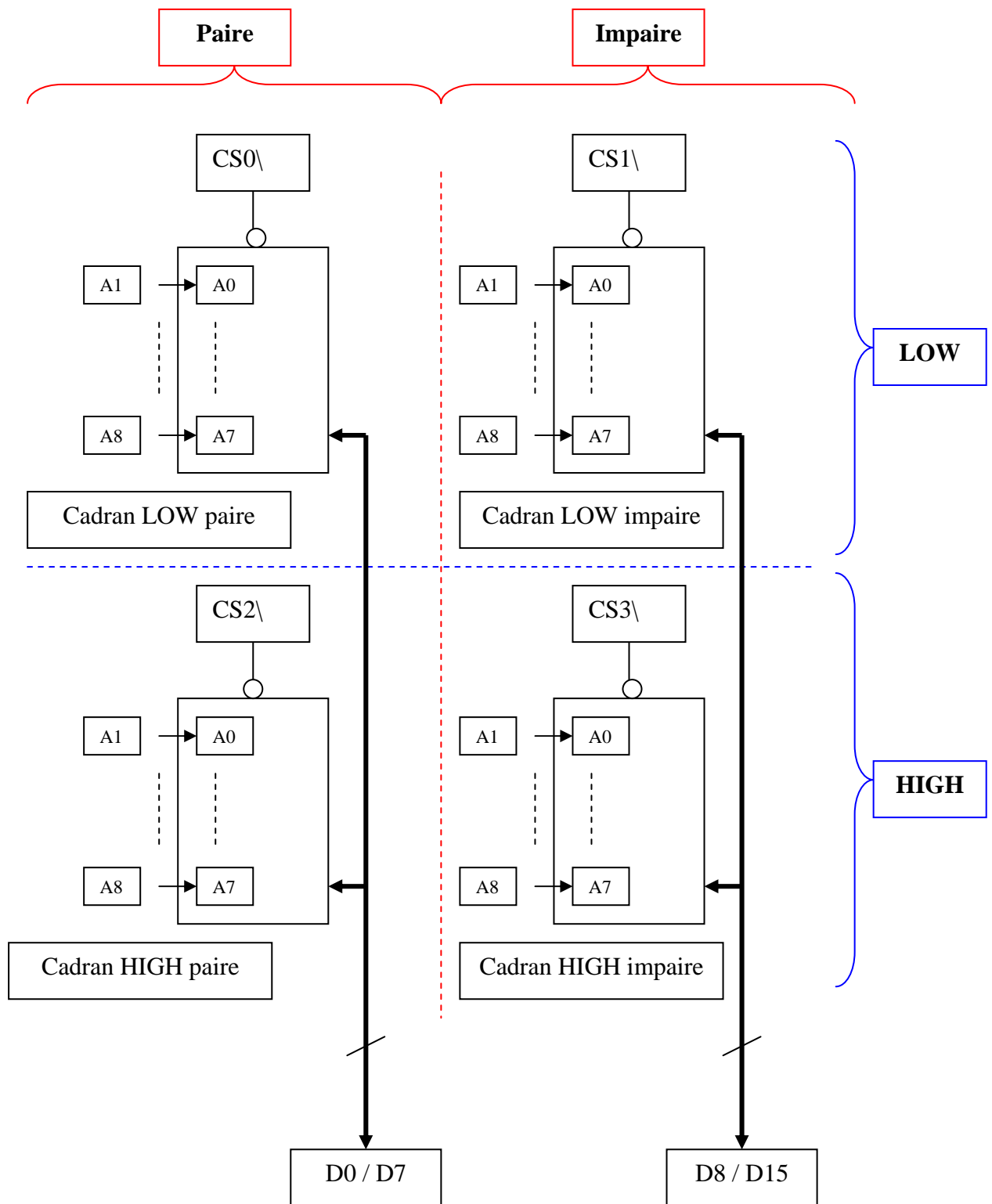
Adresse finale = 00000h + (valeur max sur 10 lignes = 3FFh).

Adresse finale = 003FFh.

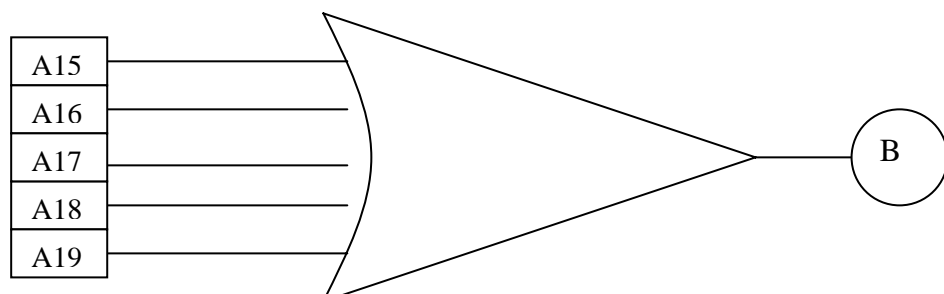
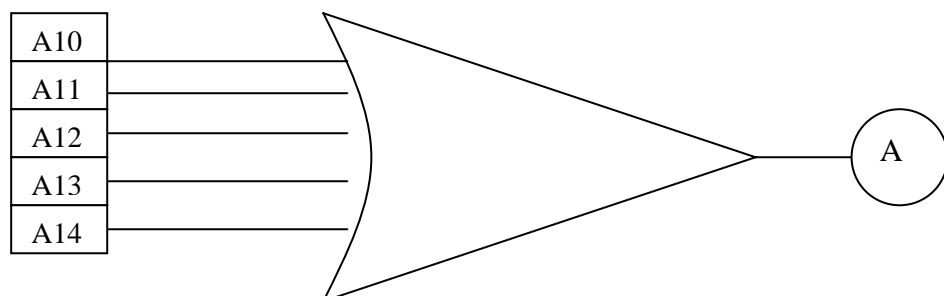
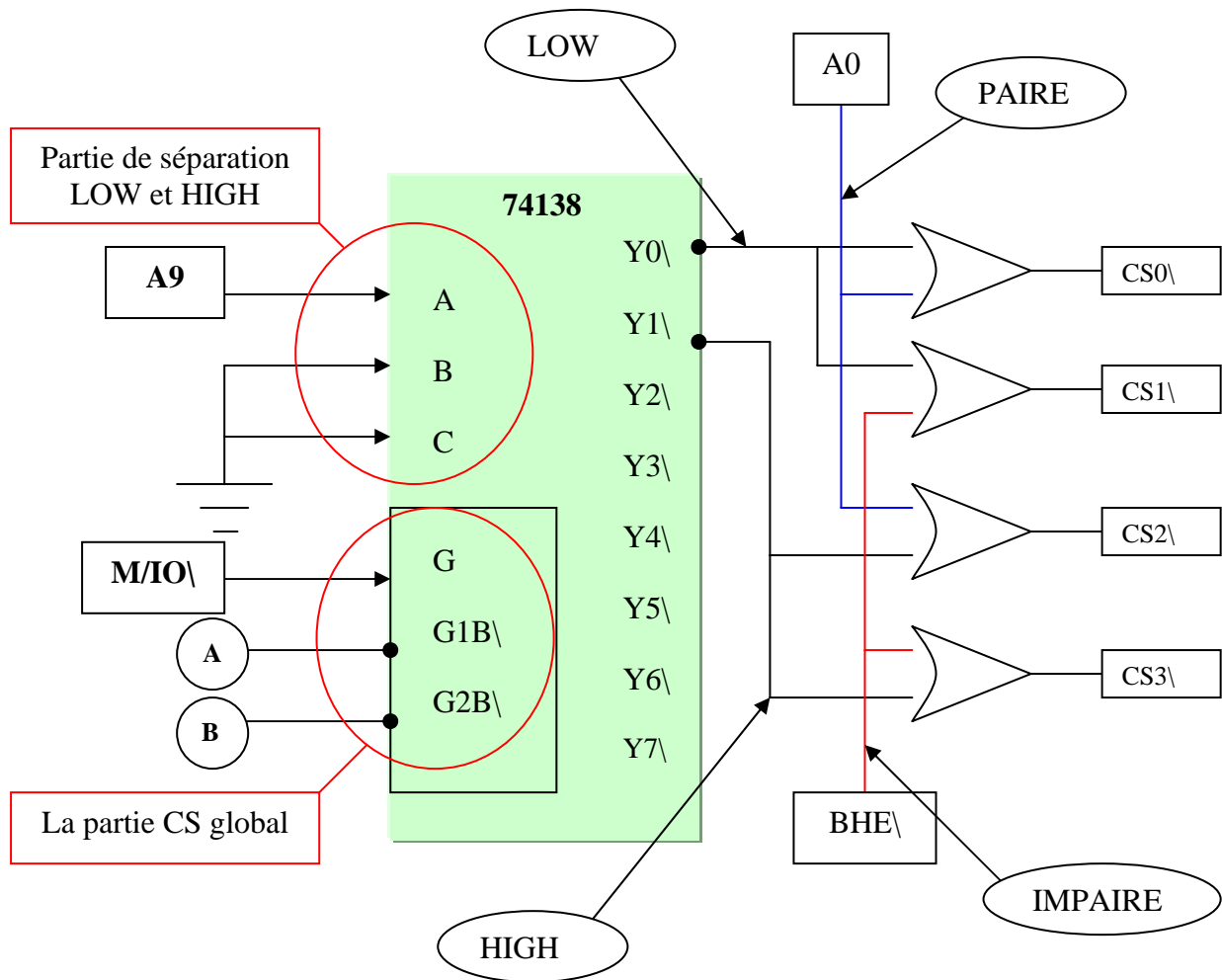
On remarque que les lignes d'adresse A10 → A19 et  $M/IO = 1$  prend la même valeur pour n'importe quelle adresse de l'espace demandé de 1 Ko dans l'adresse de base spécifiée → c'est le CS (CHIP SELECT) global de cette zone. Et en plus la ligne d'adresse A0 est non utilisée dans les lignes d'adresse utilisée par ce que cette dernière est utilisée avec la ligne BHE\ pour la séparation paire et impaire du bloc demandé de 1 Ko. Il reste la ligne A9, cette ligne on utilise pour la séparation de l'espace mémoire demandée LOW (A9 = 0), et de l'espace mémoire demandée HIGH (A9 = 1).

On distingue dans ce cas que l'espace demandé est divisé en quatre cadrans sont :

- Cadrant LOW paire.
- Cadrant LOW impaire.
- Cadrant HIGH paire.
- Cadrant HIGH impaire.



Maintenant on réalise la logique de chaque  $CS_i$  par le circuit décodeur suivant :



### 5) Les Interfaces parallèles :

L'interface parallèle est une carte électronique I/O qui fait l'échange de donnée entre le  $\mu$ p et un périphérique en parallèle selon la taille de bus de données (8 ou 16 bits). Les points d'accès aux I/O sont appelés PORTS.

#### Gestion des ports :

Le  $\mu$ p 8086 fournies un espace adressable pour l'accès ports de taille 64Ko → 16 lignes d'adresse et 16 lignes de données plus quelque lignes de contrôles. Mais on remarque que la mémoire et les ports I/O partage le même bus de données et d'adresse, alors comment différencier l'accès entre eux. Tous simplement si on rappelle le cours de l'interfaçage mémoires dans la partie « CHIP SELECT » de décodeur 74138 l'entrée « G », cette entrée est attaquée par le signal « M/IO\ » de bus de contrôle.

- Pour l'accès mémoire : M/IO\ = 1.
- Pour l'accès ports : M/IO\ = 0.

Les instructions utilisées pour R/W d'un port sont « IN/OUT ». Il existe deux types d'accès, accès direct et un autre indirect. Les registres utilisés sont l'accumulateur « AL/AX » et le registre de données « DX ».

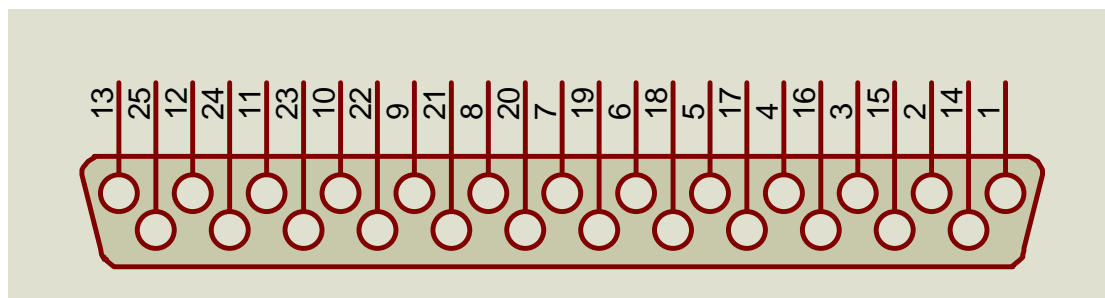
On peut citer dans ce stade les 2 interfaces parallèles standard « PORT PARALLELE » et « COUPLEUR PARALLELE PPI 8255 ».

#### Le port parallèle :

Cette interface est généralement utilisée par les imprimantes, car elles utilisent beaucoup de données. Cependant cette interface à une limite physique lié aux problèmes de diaphonie entre les fils, ce qui à pour principale conséquence de limiter la longueur des câbles parallèles. Au delà de 10 m on commence à avoir une atténuation significative du signal. Ceci dit il existe des amplificateurs de signal qui prennent généralement la forme de boîtier de partage auto commuté.

#### Description physique:

Le port parallèle des PC se présente sous la forme d'une prise DB25 femelle.



Description du brochage du connecteur DB25 F :

Broche	Description	E/S
1	-STROBE	Sortie
2	+Bit de données 0	Sortie
3	+Bit de données 1	Sortie
4	+Bit de données 2	Sortie
5	+Bit de données 3	Sortie
6	+Bit de données 4	Sortie
7	+Bit de données 5	Sortie
8	+Bit de données 6	Sortie
9	+Bit de données 7	Sortie
10	-Accusé de réception	Entrée
11	+Occupé	Entrée
12	+Plus de papier	Entrée
13	+Sélection	Entrée
14	-Chargement automatique	Sortie
15	-Erreur	Entrée
16	-Initialisation de l'imprimante	Sortie
17	-Sélection de l'entrée	Sortie
18	-Retour du bit de données 0 (masse)	Entrée
19	-Retour du bit de données 1 (masse)	Entrée
20	-Retour du bit de données 2 (masse)	Entrée
21	-Retour du bit de données 3 (masse)	Entrée
22	-Retour du bit de données 4 (masse)	Entrée
23	-Retour du bit de données 5 (masse)	Entrée
24	-Retour du bit de données 6 (masse)	Entrée
25	-Retour du bit de données 7 (masse)	Entrée

### Description des ports :

Il dispose 3 ports pour la communication avec l'imprimante, ces ports sont le port DATA pour la transmission des caractères, le port STATUS pour indiquer l'état de l'imprimante, le port CONTROL pour contrôler l'échange de données.

**Port DATA** : est un port de données de taille 8 bits configuré en sortie dans la version standard.

P9	P8	P7	P6	P5	P4	P3	P2
D7	D6	D5	D4	D3	D2	D1	D0

**Port STATUS** : est un port de 5 bits configuré en entrée.

P11	P10	P12	P13	P15	X	X	X
S7\	S6	S5	S4	S3			

**Port CONTROL** : est un port de 4 bits configuré en sortie.

X	X	X	X	P17	P16	P14	P1
				C3\	C2	C1\	C0\

Les adresses des ports parallèles implantés dans les PC compatible PC IBM :

Numéro de LPT	Port DATA	Port STATUS	Port CONTROL
LPT1	0378H	0379H	037AH
LPT2	0278H	0279H	027AH
LPT3	03BCH	03BDH	03BEH

### Exemple d'accès :

#### Lecture du port STATUS :

On remarque que l'adresse du port est une adresse sur 16 bits ➔ le mode d'accès indirect par l'utilisation de registre DX.

MOV DX, 0379h

IN AL, DX

#### Ecriture dans le port DATA ou CONTROL:

MOV AL, 55h

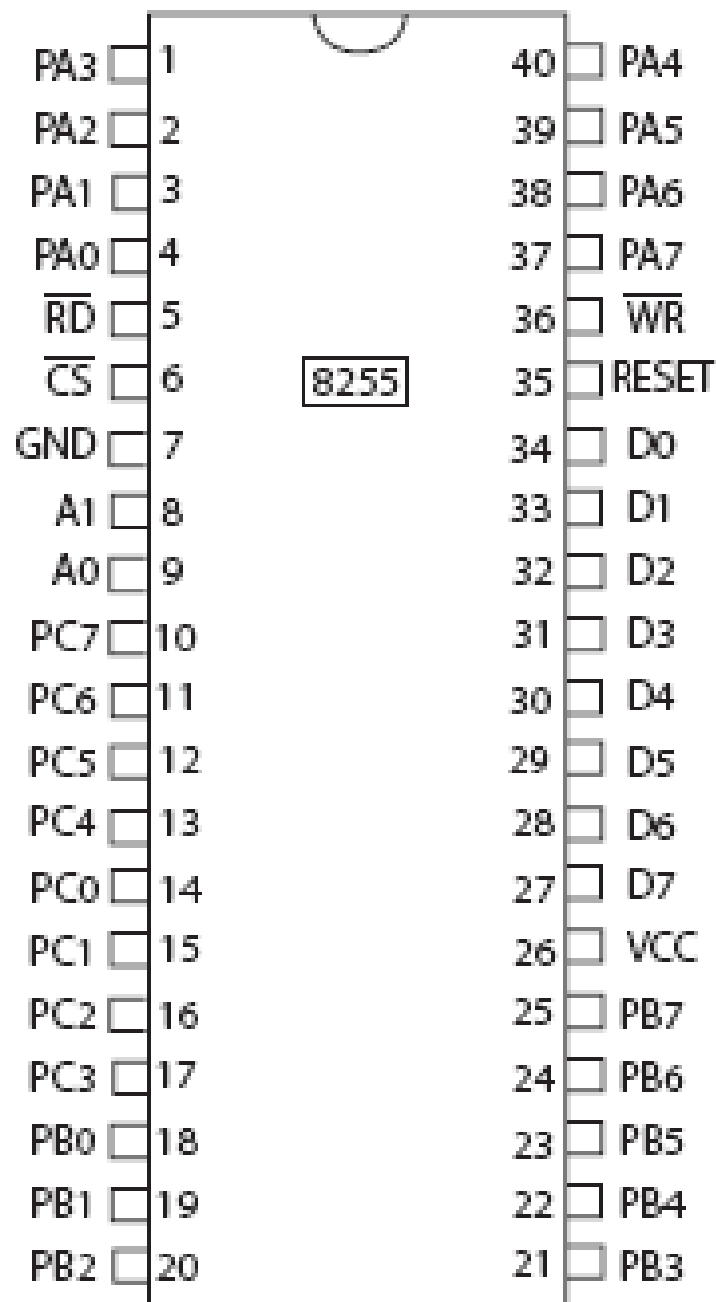
MOV DX, 0378h ; dans le cas de registre de control on mettre 037Ah

OUT DX, AL

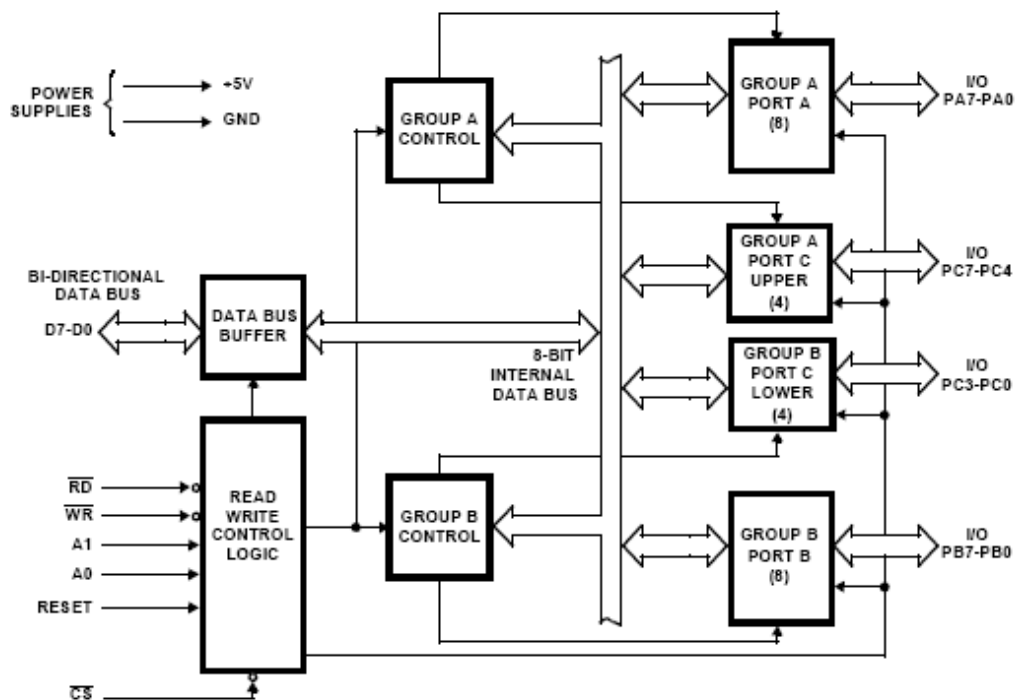
### Le coupleur parallèle PPI 8255:

Le rôle d'une interface parallèle est de transférer des données du  $\mu$ p vers un périphérique et l'inverse en parallèle. Le 8255 est une interface parallèle programmable, elle peut être configurée en entrée et/ou en sortie par programme.

### Brochage du 8255 :



### Schéma fonctionnel :



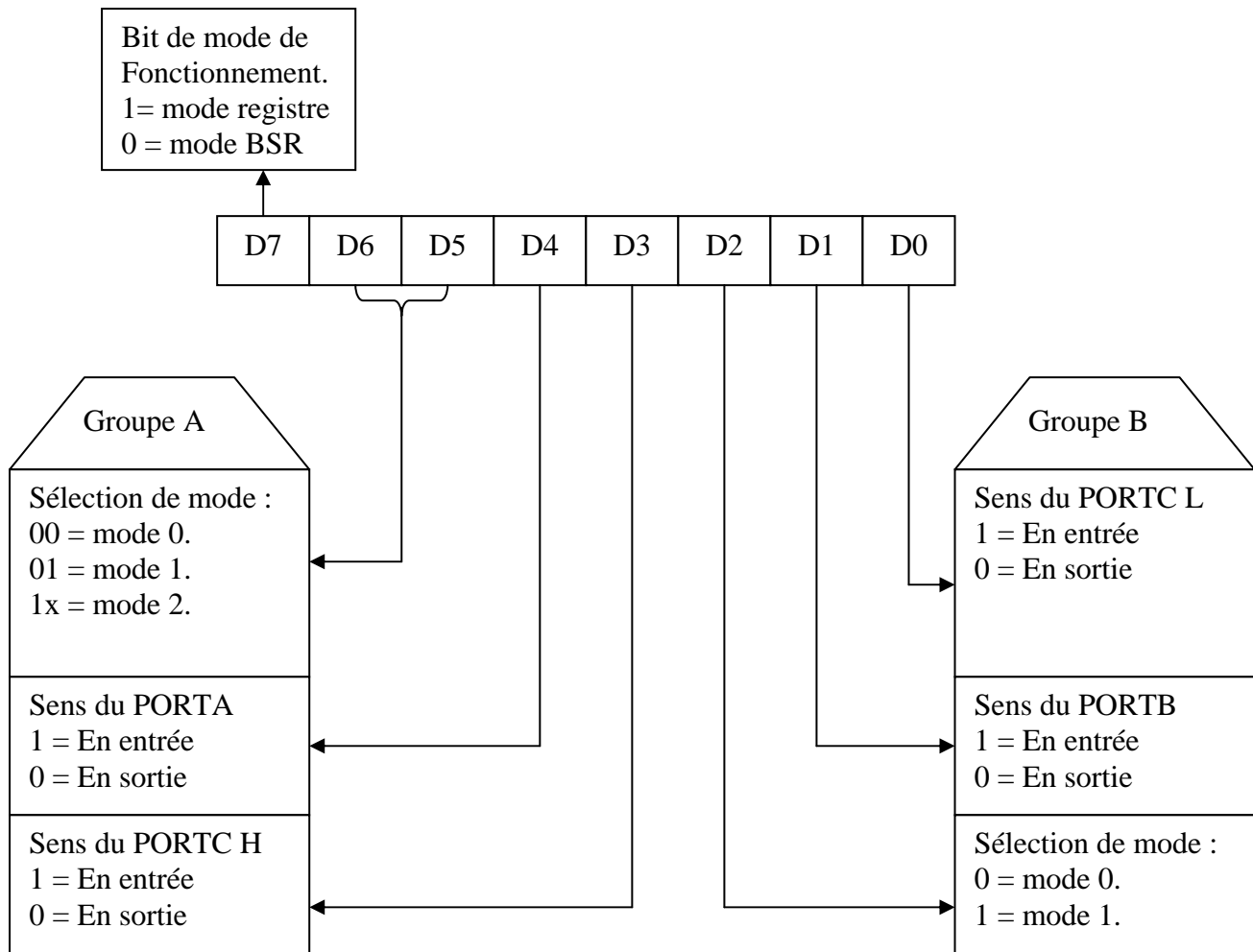
### Description des PINs et l'accès registres :

SYMBOL	PIN	TYPE	Description
VCC	26	-	Pin d'alimentation 5v
GND	7	-	Masse
D0-D7	27-34	I/O	Bus de données 8 bits bidirectionnels
RESET	35	I	Entrée de demande d'initialisation boîtier
CS\	6	I	Entrée d'activation boîtier
RD\	5	I	Entrée de demande de lecture registre
WR\	36	I	Entrée de demande d'écriture registre
A0-A1	8,9	I	Lignes d'adresse des registres
PA0-PA7	1-4,37-40	I/O	Pins du port A
PB0-PB7	18-25	I/O	Pins du port B
PC0-PC7	10-17	I/O	Pins du port C

### Accès registres :

A1	A0	RD\	WR\	CS\	Opération
0	0	0	1	0	PORTA → DATA BUS
0	1	0	1	0	PORTB → DATA BUS
1	0	0	1	0	PORTC → DATA BUS
1	1	0	1	0	CR → DATA BUS
0	0	1	0	0	DATA BUS → PORTA
0	1	1	0	0	DATA BUS → PORTB
1	0	1	0	0	DATA BUS → PORTC
1	1	1	0	0	DATA BUS → CR
X	X	X	X	1	DATA BUS Haute impédance
X	X	1	1	0	DATA BUS Haute impédance

### Structure de registre CR :

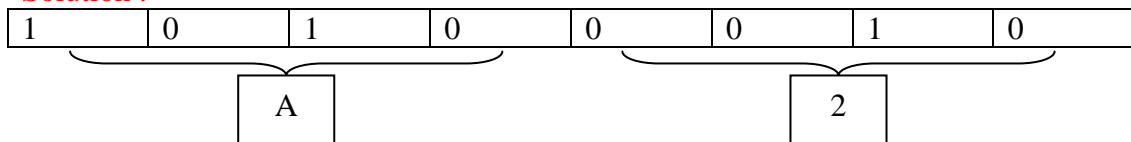


### Exemple :

Faites la configuration suivante :

- PORTA en mode 1 en sortie.
- PORTB en mode 0 en entrée.
- PORTC en sortie.

### Solution :



```
MOV AL, 0A2h      ; la valeur de CR qui assure la configuration.
MOV DX, ADR_CR    ; l'adresse de registre CR dans le registre DX.
OUT DX, AL        ; écrire dans le registre CR.
```

### Fonctionnement du PPI 8255 :

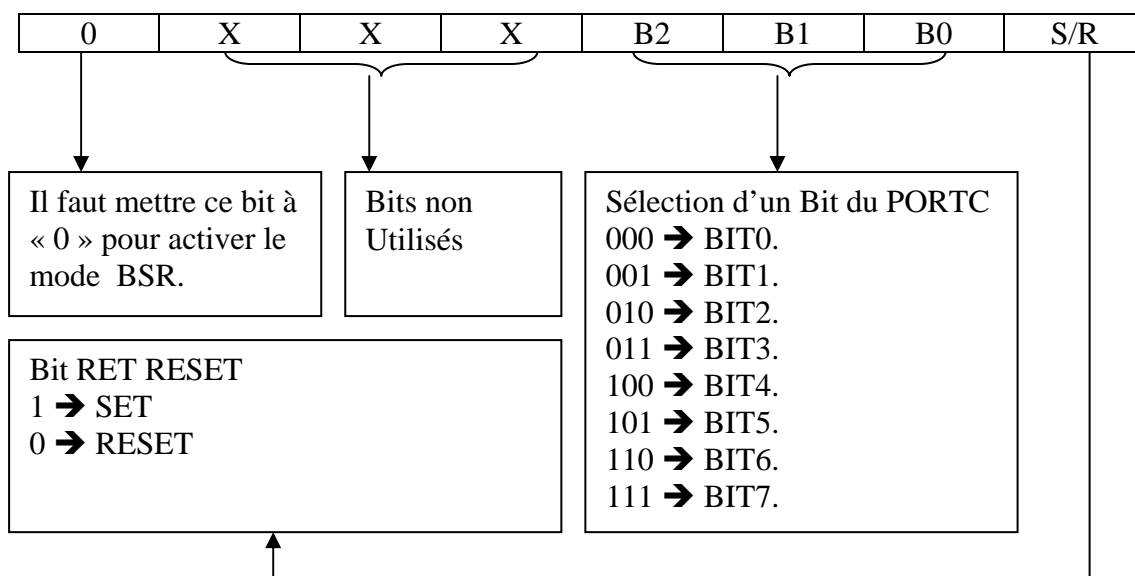
Le PPI 8255 fonctionne en 4 modes (mode0, mode1, mode2, mode BSR). Ces modes sont activés par la programmation de registre CR.

### Mode BSR (BIT SET RESET):

Ce mode utilise par le PORTC uniquement qui peuvent être mise à « 1 » ou « 0 » d'une ligne du PORTC individuellement. Dans ce mode la structure de registre CR est modifiée mais cette modification ne dérange pas le mode de fonctionnement des ports (A et B).

Remarque : n'oublie pas de configurer le PORTC en sortie dans le registre CR.

### Structure de registre CR en mode BSR :



### Exemple :

Mettez la PIN6 du PORTC à « 1 » et la PIN3 à « 0 ».

Configuration de PORTC en sortie CR ← B'10000000' = 80h.

Pour mettre la PIN6 à « 1 » le CR ← B'00001101' = 0Dh.

Pour mettre la PIN6 à « 1 » le CR ← B'00000110' = 06h.

; Configuration des registres et PORTC en sortie.

MOV DX, ADR\_CR

MOV AL, 80h

OUT DX, AL

; PIN6 ← 1

MOV AL, 0Dh

OUT DX, AL

; PIN3 ← 0

MOV AL, 06h

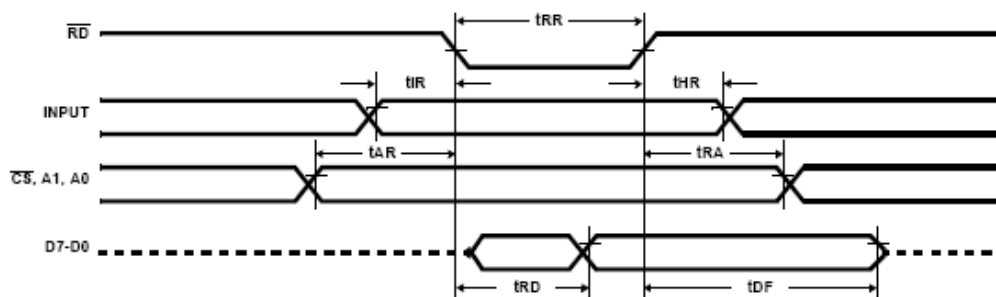
OUT DX, AL

### Mode 0:

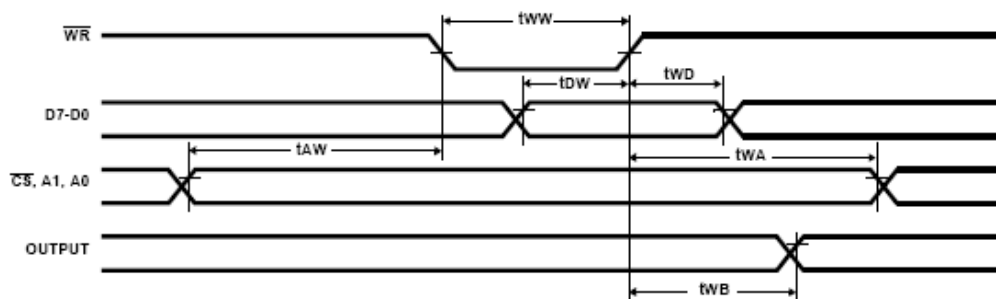
C'est le mode le plus simple : les ports sont configuré en entrées/sorties de base. Les données écrites dans les registres correspondants sont mémorisées sur les lignes de sorties ; l'état des entrées est recopié dans les registres correspondants et n'est pas mémorisé (bufférisées).

Chronogramme d'accès :

Mode 0 (Basic Input)



Mode 0 (Basic Output)



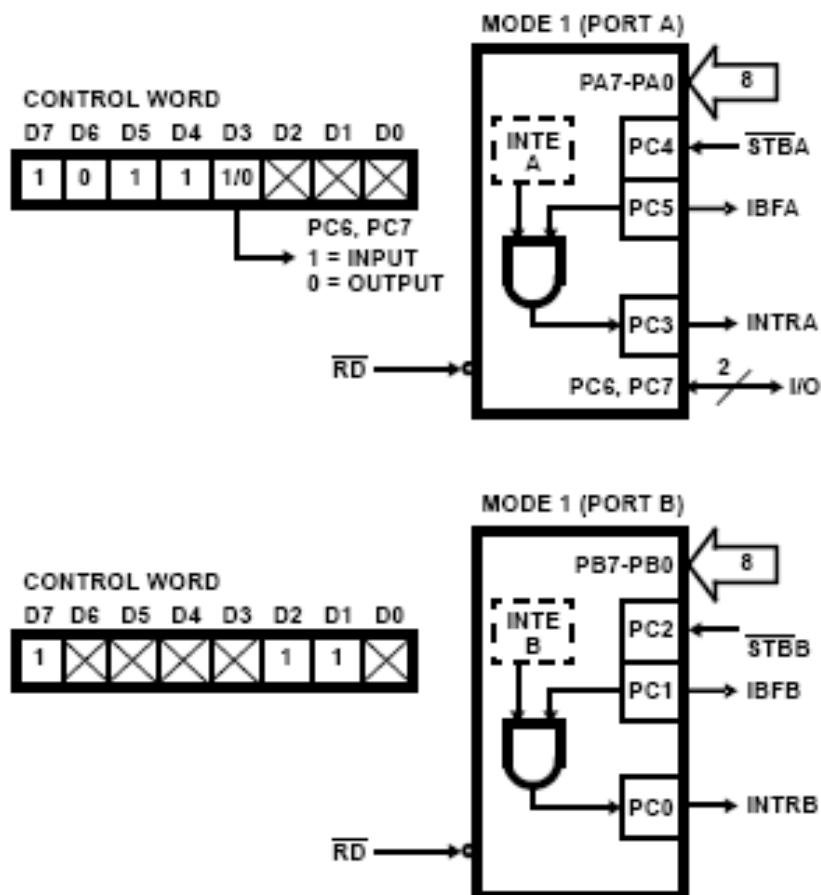
Différentes mots de configuration en mode 0 :

Valeur CR	Description
80h	A: OUT; B: OUT; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
82h	A: OUT; B: IN; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
81h	A: OUT; B: OUT; C <sub>L</sub> : IN; C <sub>H</sub> : OUT.
83h	A: OUT; B: OUT; C <sub>L</sub> : IN; C <sub>H</sub> : IN.
88h	A: OUT; B: IN; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
90h	A: IN; B: OUT; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
89h	A: OUT; B: IN; C <sub>L</sub> : IN; C <sub>H</sub> : OUT.
91h	A: IN; B: OUT; C <sub>L</sub> : IN; C <sub>H</sub> : OUT.
8Ah	A: OUT; B: IN; C <sub>L</sub> : OUT; C <sub>H</sub> : IN.
92h	A: IN; B: OUT; C <sub>L</sub> : OUT; C <sub>H</sub> : IN.
8Bh	A: OUT; B: IN; C <sub>L</sub> : IN; C <sub>H</sub> : IN.
98h	A: IN; B: IN; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
99h	A: IN; B: IN; C <sub>L</sub> : IN; C <sub>H</sub> : OUT.
93h	A: IN; B: OUT; C <sub>L</sub> : IN; C <sub>H</sub> : IN.
9Ah	A: IN; B: IN; C <sub>L</sub> : OUT; C <sub>H</sub> : OUT.
9Bh	A: IN; B: IN; C <sub>L</sub> : IN; C <sub>H</sub> : IN.

### Mode 1:

Il est utilisé pour le dialogue avec des périphériques nécessitant un asservissement (contrôle), dans ce mode les registres de 8255 sont regroupés en deux groupes (GROUPE A et GROUPE B) chaque groupe dispose le port du groupe plus quelques lignes du PORTC comme signaux du contrôle.

#### Mode 1 en entrée :



On remarque que le PORTC devient seulement les PINs PC6 et PC7 le reste des PINs seront des PINs du contrôle de groupe A et B.

Description des signaux du contrôle :

**STB\** (STROBE) : ce signal active au niveau bas délivrer par le périphérique qui indique que la donnée est présente dans le BUFFER d'entrée (PORTA ou PORTB).

**IBF** (INPUT BUFFER FULL) : ce signal active au niveau haut pour indiquer au périphérique la bonne réception de la donnée.

**INTR** (INTERRUPT REQUEST) : ligne active au niveau haut lorsque STB\ et INB sont aux niveau haut.

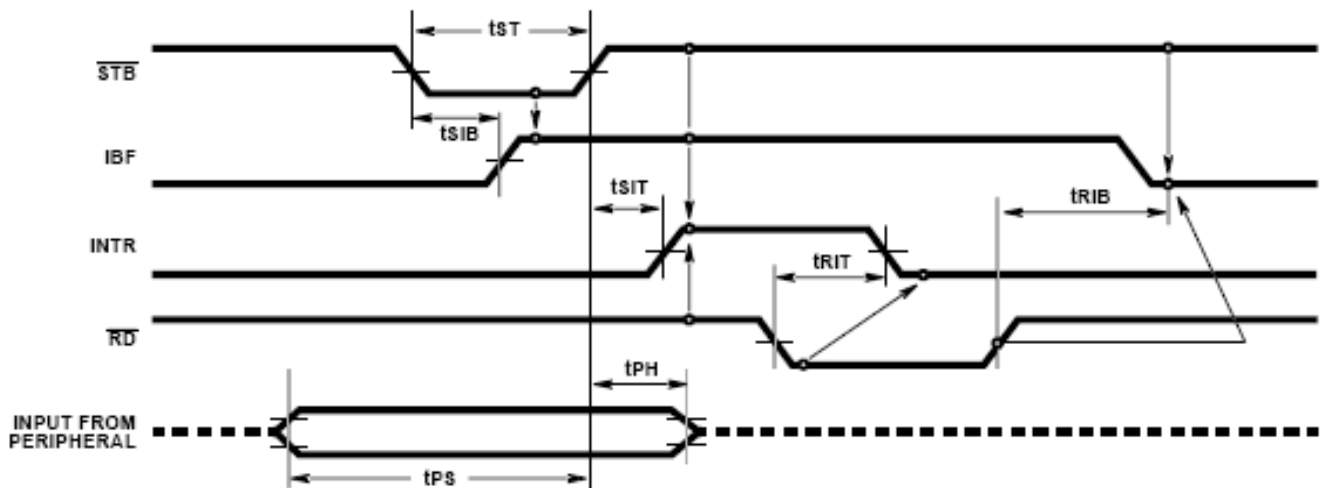
**INTE** (INTERUPT ENABLE) : c'est un bit utilisé pour autoriser le mode de fonctionnement sous interruption.

Le PORTC est appelé le mot d'état de groupe A et B :

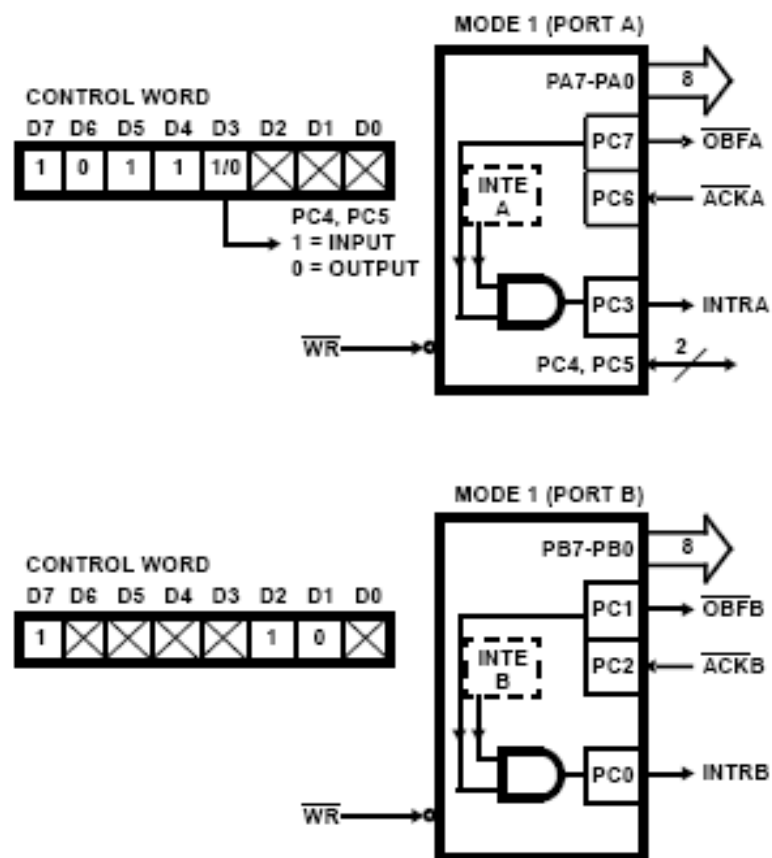
I/O	I/O	IBF A	INTE A	INTR A	INTE B	IBF B	INTR B
-----	-----	-------	--------	--------	--------	-------	--------

N'oubliez pas que les ports (A et B) sont des mémoires en entrée dans ce mode.

### Chronogramme d'accès :



### Mode 1 en sortie :



Dans ce cas la le PORTC devient les PINs PC5 et PC4.

Description des signaux du contrôle :

**OBF** (OUTPUT BUFFER FULL) : ce signal passe au niveau bas si le  $\mu$ p écrit une donnée dans le BUFFER de sortie, ce signal aussi indique au périphérique que il y a une donnée dans le BUFFER (les lignes du port A ou B).

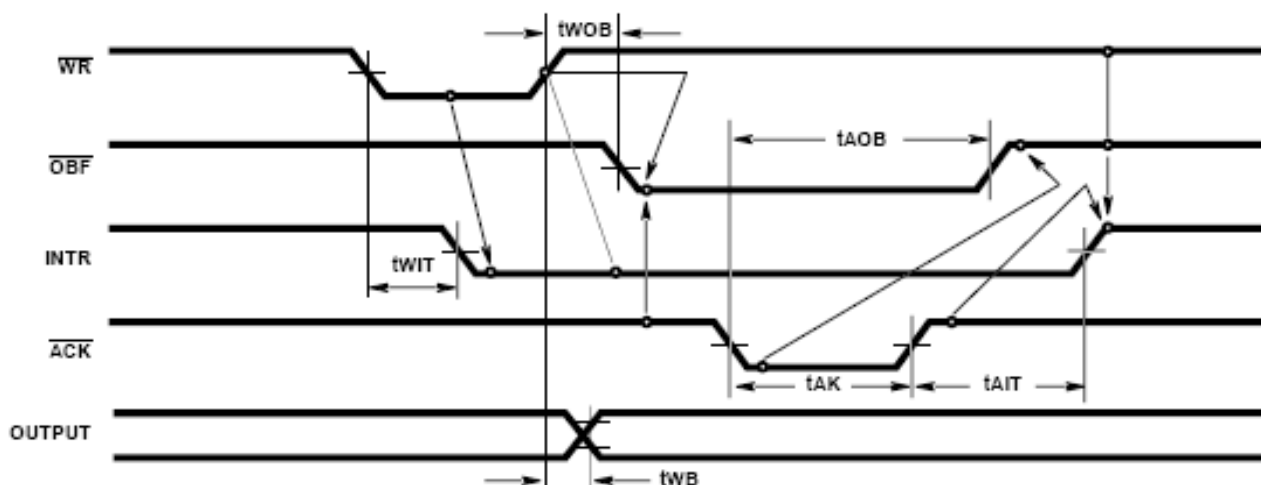
**ACK** (ACKNOWLEDGE INPUT) : ce signal délivré par le périphérique pour indiquer la bonne réception de la donnée sous forme impulsion négative.

**INTR** et **INTE** sont les mêmes signaux du mode 1 en entrée.

Le mot d'état du mode 1 en sortie:

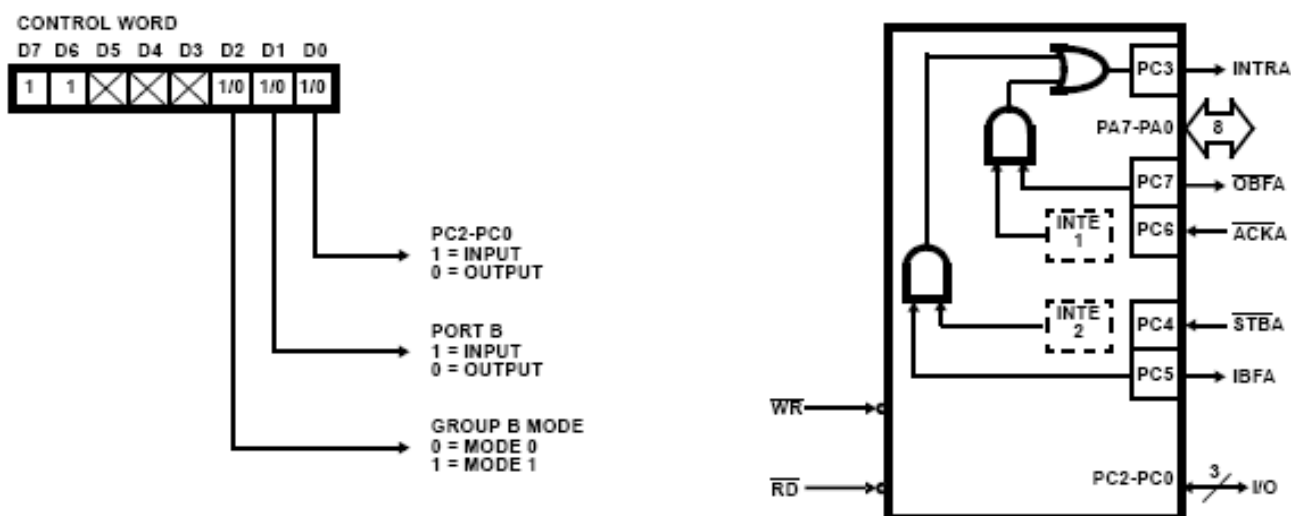
OBF\ A	INTE A	I/O	I/O	INTR A	INTE B	OBF\ B	INTR B
--------	--------	-----	-----	--------	--------	--------	--------

Chronogramme d'accès :



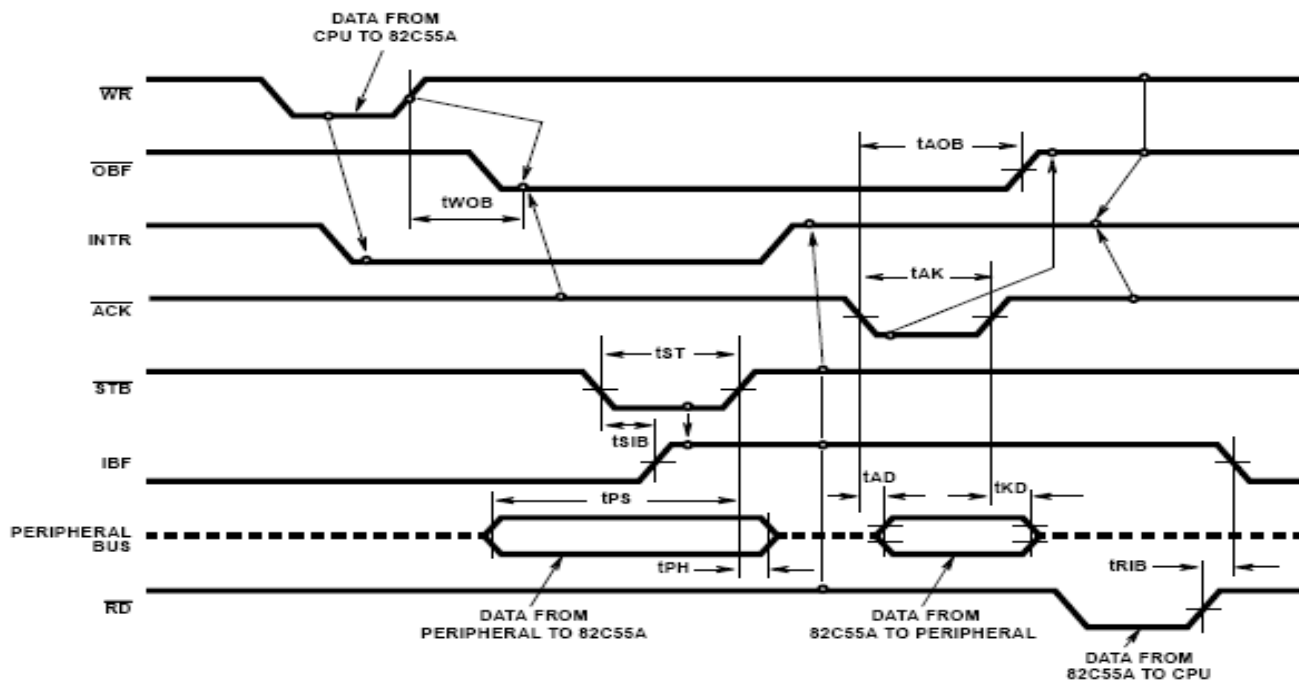
## Mode 2 :

Dans ce mode est utilisé seulement par le PORTA et peut être considéré comme un bus bidirectionnelle. Le groupe A sera donc le PORTA plus 5 lignes du PORTC, et le groupe B si on configure en mode 1 le PORTC est disparu par ce que le mode 1 extrait 3 lignes de ce dernier, mais dans le cas mode 0 le PORTC est de 3 lignes.



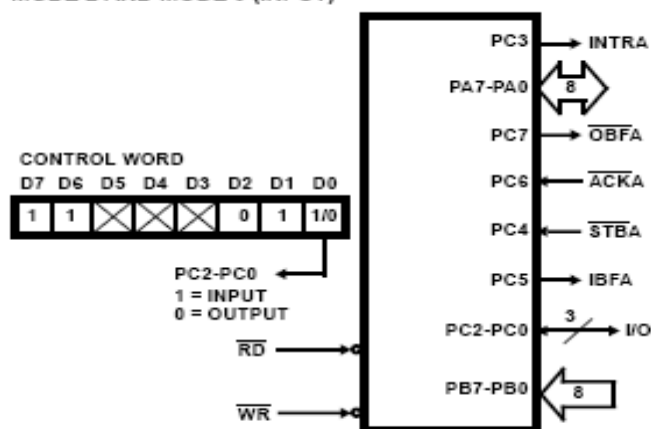
Les signaux du contrôle sont les mêmes que les signaux du mode 1 en I/O.

### Chronogramme d'accès :

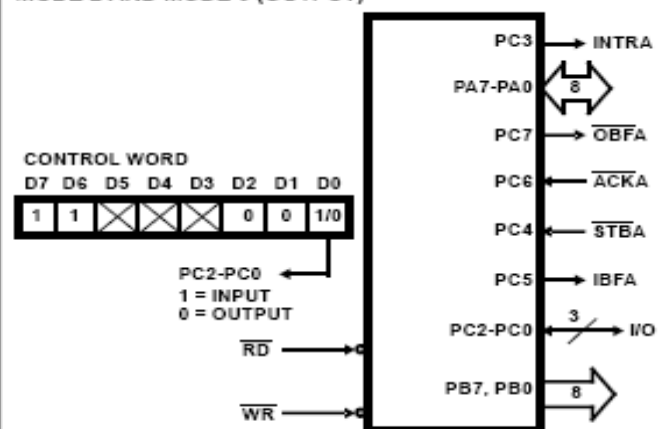


### Différentes combinaisons en mode 2 :

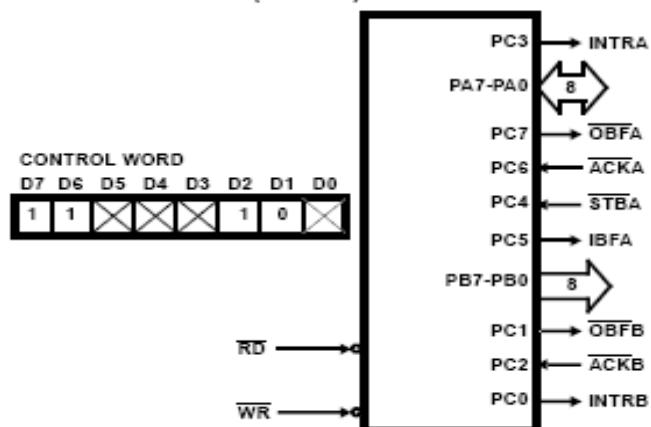
MODE 2 AND MODE 0 (INPUT)



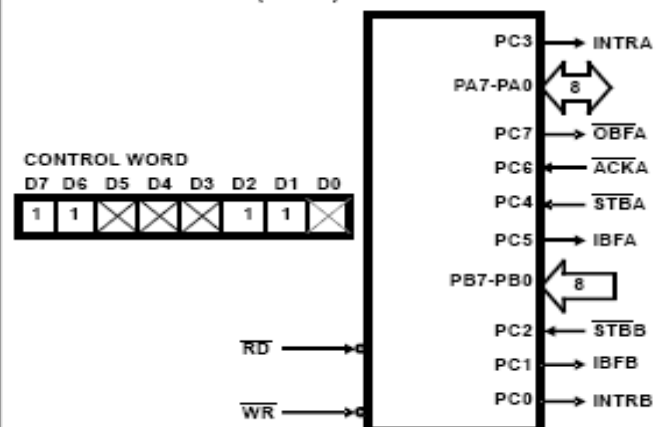
MODE 2 AND MODE 0 (OUTPUT)



MODE 2 AND MODE 1 (OUTPUT)



MODE 2 AND MODE 1 (INPUT)



Les mots d'état pour chaque combinaison :

**MODE 2 AND MODE 0 (INPUT) :**

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	I/O	I/O	I/O
--------	--------	-------	--------	--------	-----	-----	-----

**MODE 2 AND MODE 0 (OUTPUT) :**

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	I/O	I/O	I/O
--------	--------	-------	--------	--------	-----	-----	-----

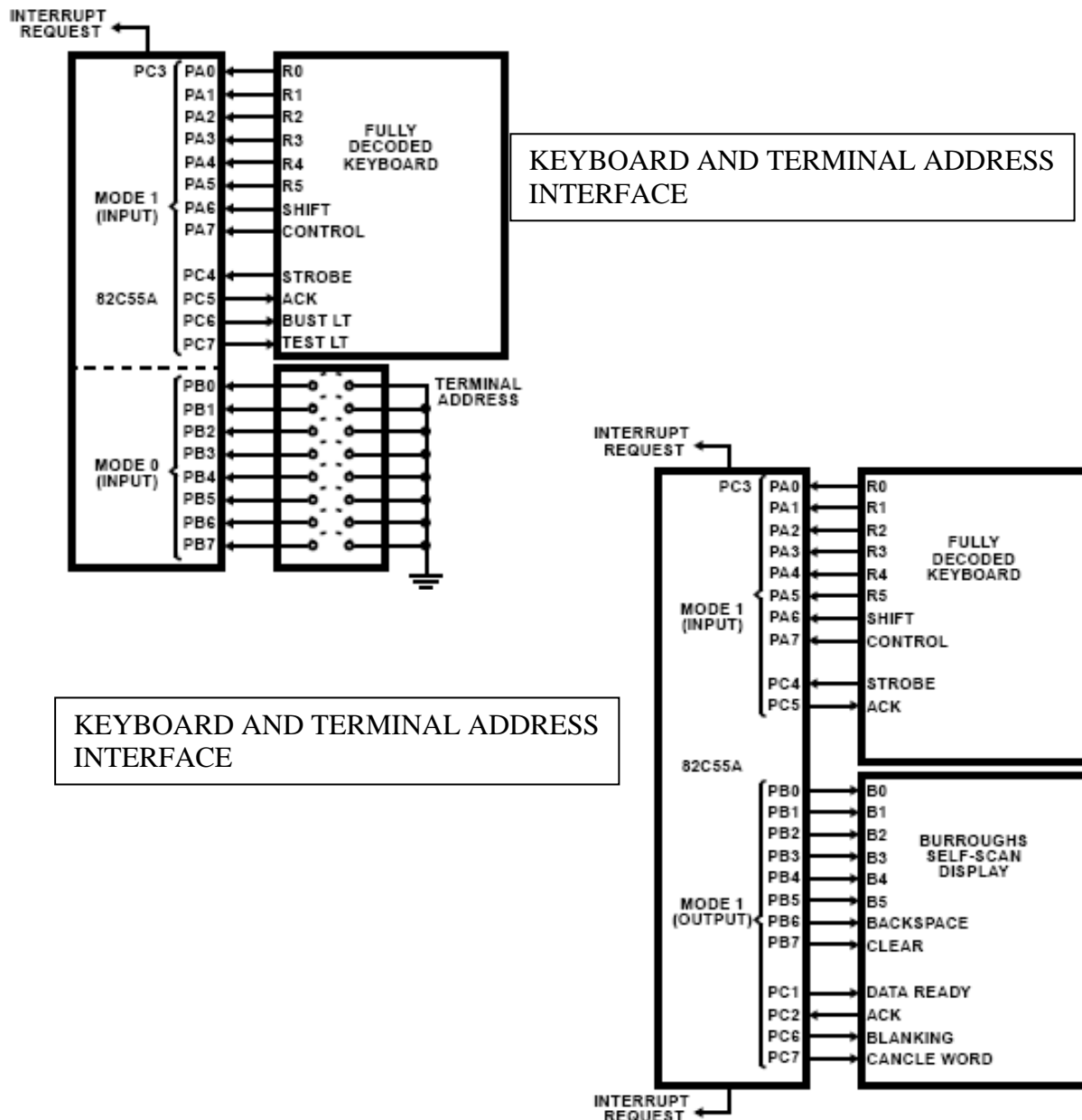
**MODE 2 AND MODE 1 (INPUT) :**

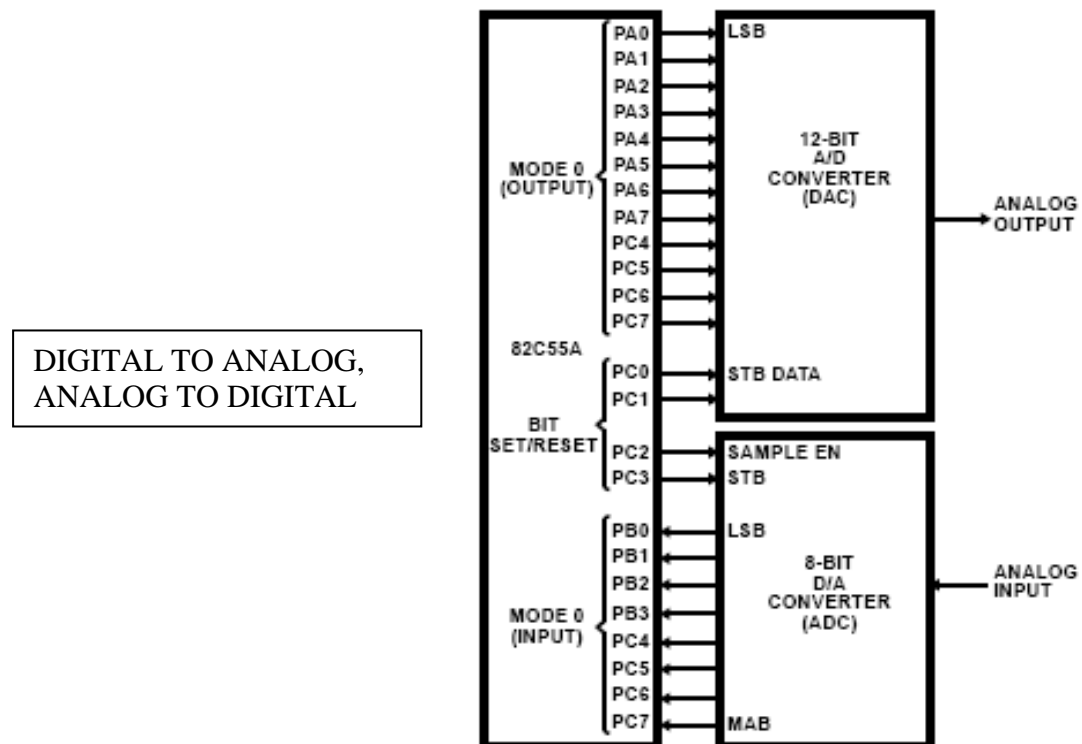
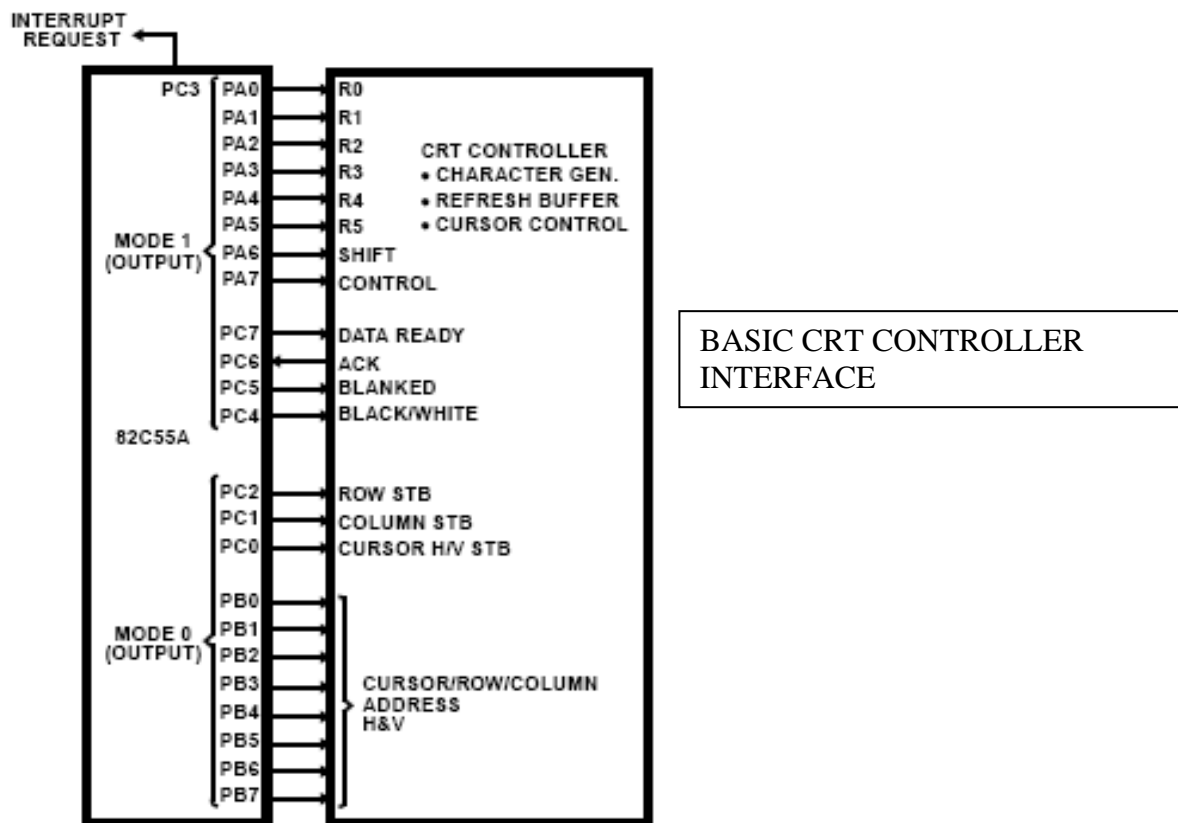
OBF\ A	INTE 1	IBF A	INTE 2	INTR A	INTE B	IBF B	INTR B
--------	--------	-------	--------	--------	--------	-------	--------

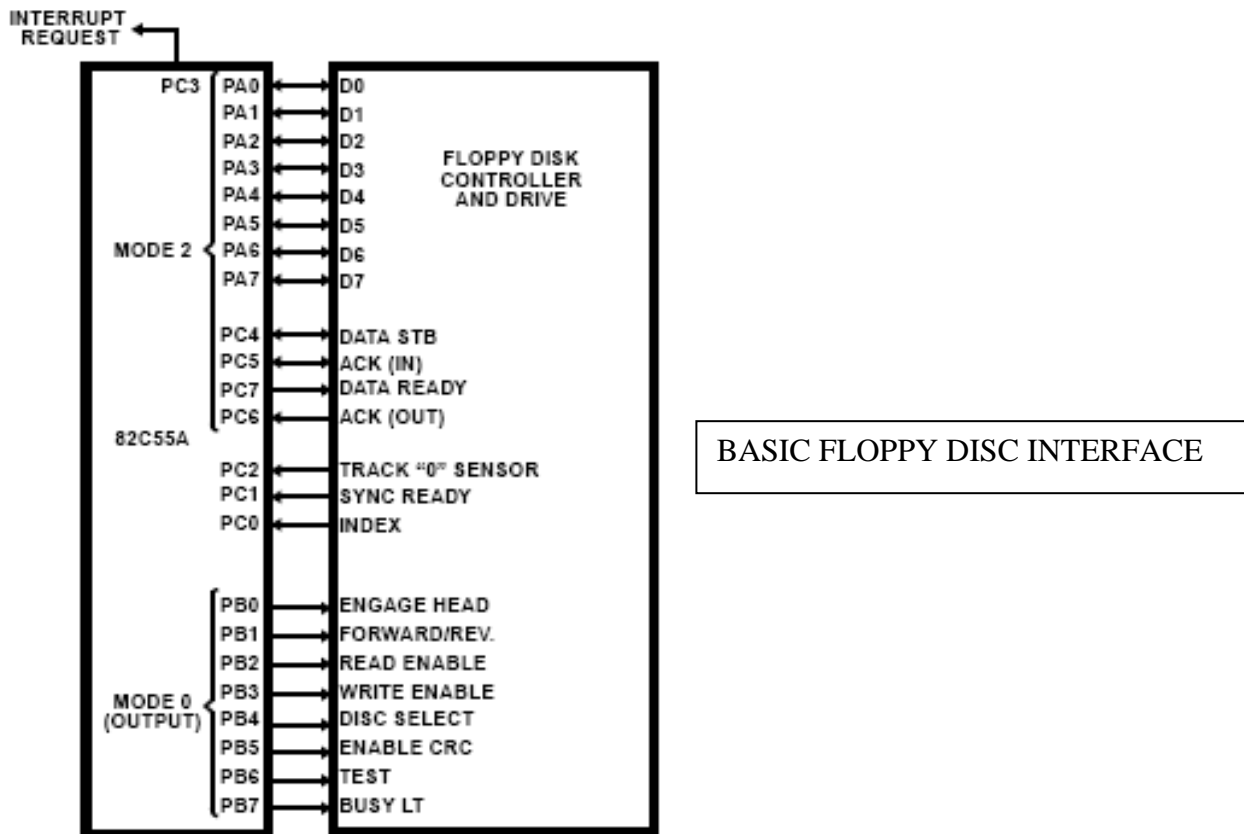
**MODE 2 AND MODE 1 (OUTPUT) :**

OBF\ A	INTE 1	IBF A	INTE 2	INTR A	INTE B	OBF\ B	INTR B
--------	--------	-------	--------	--------	--------	--------	--------

Applications :







MACHINE TOOL CONTROLLER INTERFACE

