Université Internationale de Casablanca

LAUREATE INTERNATIONAL UNIVERSITIES

Filière GI

Première Année du Cycle Ingénieur

Support de cours

Microprocesseurs

Encadré par : A. ERRAMI

Année 2017 - 2018

Chapitre 1
Introduction et Concepts de base

Tout nombre entier positif (de n chiffres a_i) peut être représenté, en base b, par une expression de la forme :

$$x = a_{n-1} * b^{n-1} + a_{n-2} * b^{n-2} + ... + a_1 * b^1 + a_0 * b^0$$

Exemples de système de numérotation :

- système décimal : B=10, $C_i \in$ { 0,1,2,….,9}
- système hexadécimal : B=16, $C_i \in$ { 0,1,2,….,9,A,B,C,D,E,F}
- système binaire : B=2, $C_i \in$ { 0,1}

335 en base 10

$$335 = \mathbf{3} * 10^2 + \mathbf{3} * 10^1 + \mathbf{5} * 10^0$$

$$=> \mathbf{335}$$

335 en base 2

$$335 = \mathbf{1} * 2^8 + \mathbf{0} * 2^7 + \mathbf{1} * 2^6 \mathbf{0} * 2^5 + \mathbf{0} * 2^4 + \mathbf{1} * 2^3 + \mathbf{1} * 2^2 + \mathbf{1} * 2^1 + \mathbf{1} * 2^0$$

$$=> \mathbf{101001111}$$

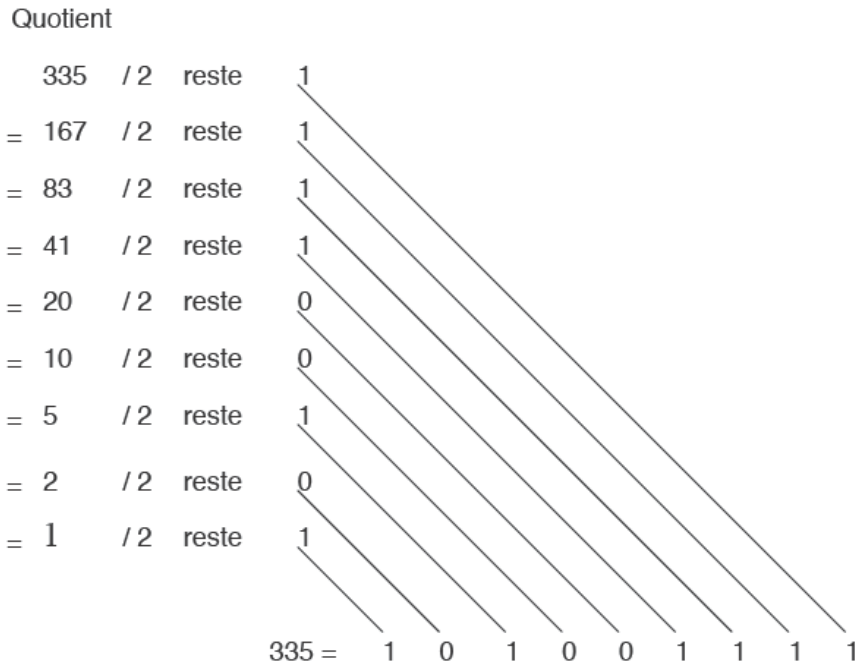
Représentation des nombres dans le système Binaire Codé Décimal BCD :

Chaque chiffre décimal composant le nombre est représenté en binaire sur 4 bits :

(1 3 8) $_{10}$ = 0001 0011 1000

Passage d'un système de numérotation à un autre :

- **du décimal vers le binaire :** technique des divisions successives par 2



- **du binaire vers le décimal :** décomposition en somme des puissances successives de deux

Additionner les puissances de 2 correspondants aux bits de valeur 1.

$$101001111 = 2^8 + 2^6 + 2^3 + 2^2 + 2^1 + 2^0 = 256 + 64 + 8 + 4 + 2 + 1 = 335$$

- **du décimal vers l'hexadécimal :** technique des divisions successives par 16.
- **de l'hexadécimal vers le binaire :** décomposition en somme des puissances successives de 16
- **du binaire vers l'hexadécimal et vice versa :** regroupement par bloc de quatre bits

Représentation en binaire des nombres négatifs

Pour représenter les nombres négatifs, on utilise le codage du signe et la technique du complément à deux.
Le complément à deux d'un nombre est obtenu en rajoutant une unité au complément à un.

Le complément à un est obtenu par substitution des '0' par des '1' et des '1' par des '0'.
La valeur du signe est indiqué par le bit de poids fort (le bits MSB) :

- MSB = 0 : signe positif
- MSB = 1 : signe négatif

→ Si le nombre est positif, son MSB = 0 et les autres bits donnent la valeur en binaire du nombre
→ Si le nombre est négatif, son MSB = 1 et les autres bits donnent la valeur en complément à deux du nombre.

Représentation des caractères

Les caractères sont des données non numériques : il n'y a pas de sens à additionner ou multiplier deux caractères. Par contre, il est souvent utile de comparer deux caractères, par exemple pour les trier dans l'ordre alphabétique.

Les caractères, appelés *symboles alphanumériques*, incluent les lettres majuscules et minuscules, les symboles de ponctuation (& ~ , . ; # " - etc...), et les chiffres.

Un texte, ou *chaîne de caractères*, sera représenté comme une suite de caractères.

Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère.

Le code ASCII (*American Standard Code for Information Interchange*) étant le systèmes de codage le plus utilisé à cet effet par les systèmes de traitement de l 'information.

Le Code ASCII

Caractères nationaux

Jeu de commandes

BITS				b_7	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	à	P		p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	£	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	'	6	F	V	f	v
0	1	1	1	7	BEL	ETB	(	7	G	W	g	w
1	0	0	0	8	BS	CAN	)	8	H	X	h	x
1	0	0	1	9	HT	EM	.	9	I	Y	i	y
1	0	1	0	A	LF	SUB		:	J	Z	j	z
1	0	1	1	B	VT	ESC	,	;	K		k	é
1	1	0	0	C	FF	ES		<	L	ç	l	ù
1	1	0	1	D	CR	GS		¨	M	§	m	è
1	1	1	0	E	SO	RS		>	N	^	n	¨
1	1	1	1	F	SI	US	/	?	O	–	o	DEL

Les commandes du code ASCII

Symbole	Signification	
ACK	Acknowledge	Accusé de réception
BEL	Bell	Sonnerie
BS	Backspace	Retour arrière
CAN	Cancel	Annulation
CR	Carriage Return	Retour chariot
DC	Device control	Commande d'appareil auxiliaire
DEL	Delete	Oblitération
DLE	Data Link Escape	Caractère d'échappement
EM	End Medium	Fin de support
ENQ	Enquiry	Demande
EOT	End Of Transmission	Fin de communication
ESC	Escape	Echappement
ETB	End of Transmission Block	Fin de bloc de transmission
ETX	End Of Text	Fin de texte
FE	Format Effector	Commande de mise en page
FF	Form Feed	Présentation de formule
FS	File Separator	Séparateur de fichiers
GS	Group Separator	Séparateur de groupes
HT	Horizontal Tabulation	Tabulation horizontale
LF	Line Feed	Interligne
NAK	Negative Acknowledge	Accusé de réception négatif
NUL	Null	Nul
RS	Record Separator	Séparateur d'articles
SI	Shift IN	En code
SO	Shift Out	Hors code
SOH	Start Of Heading	Début d'en-tête
SP	Space	Espace
STX	Start Of Text	Début d'en-tête
SYN	Synchronous idle	Synchronisation
TC	Transmission Control	Commande de transmission
US	Unit Separator	Séparateur de sous-article
VT	Vertical Tabulation	Tabulation verticale

Figure 2.5 Le code ASCII.

Représentation des nombres réels : Technique de la virgule Flottante

Les nombres à virgule flottante sont utilisés pour représenter des quantités qui ne peuvent être formulées par des entiers, soit parce qu'elles contiennent des valeurs fractionnelles, soit parce qu'elles tombent en dehors de l'intervalle qui peut être représenté par la largeur de bit du système. Pratiquement tous les ordinateurs modernes utilisent la représentation à virgule flottante spécifiée dans le standard IEEE 754, qui spécifie que les nombres sont représentés par une mantisse et un exposant. À la manière des notations scientifiques, la valeur d'un nombre à virgule flottante est $mantisse \times 2^{exposant}$.

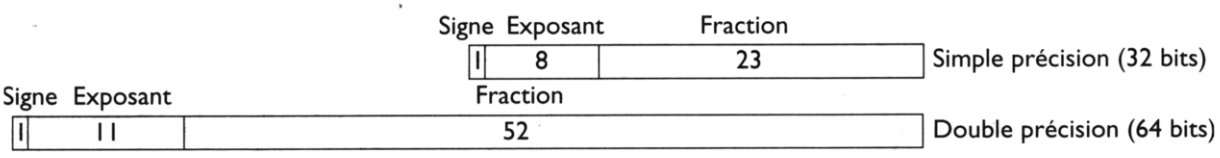


Figure 2.7 Formats à virgule flottante IEEE 754.

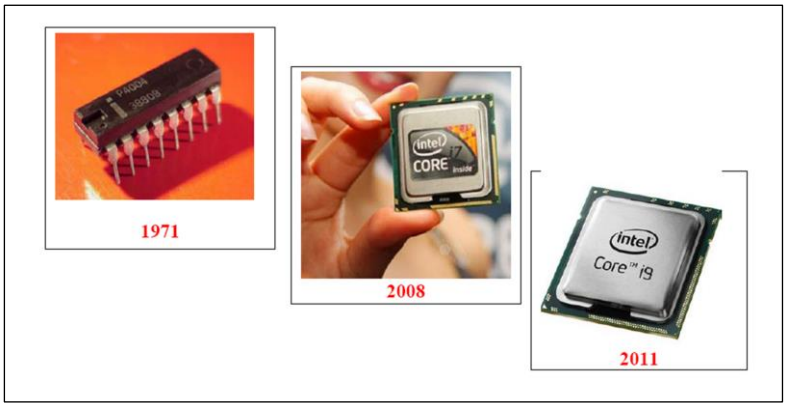
Les types de données fondamentaux en C

Type	Taille	Intervalle de valeurs	Précision
char	1 octet	-128 à 127	-
unsigned char	1 octet	0 à 255	-
int	2 ou 4 octets	comme short ou long	-
unsigned (int)	2 ou 4 octets	comme unsigned short ou long	-
short	2 octets	-32768 à 32767	-
unsigned short	2 octets	0 à 65535	-
long	4 octets	-2 147 483 648 à 2 147 483 647	-
unsigned long	4 octets	0 à 4 294 967 295	-
float	4 octets	$3.4 * 10^{-38}$ à $3.4 * 10^{38}$	6 chiffres
double	8 octets	$1.7 * 10^{-308}$ à $1.7 * 10^{308}$	15 chiffres
long double	10 octets	$1.2 * 10^{-4932}$ à $1.2 * 10^{4932}$	18 chiffres

Architecture interne et principe de fonctionnement d'un système à microprocesseur

Les microprocesseurs : c'est Quoi ?

Un ou plusieurs circuits LSI / VLSI qui réalisent des fonctions de traitement.

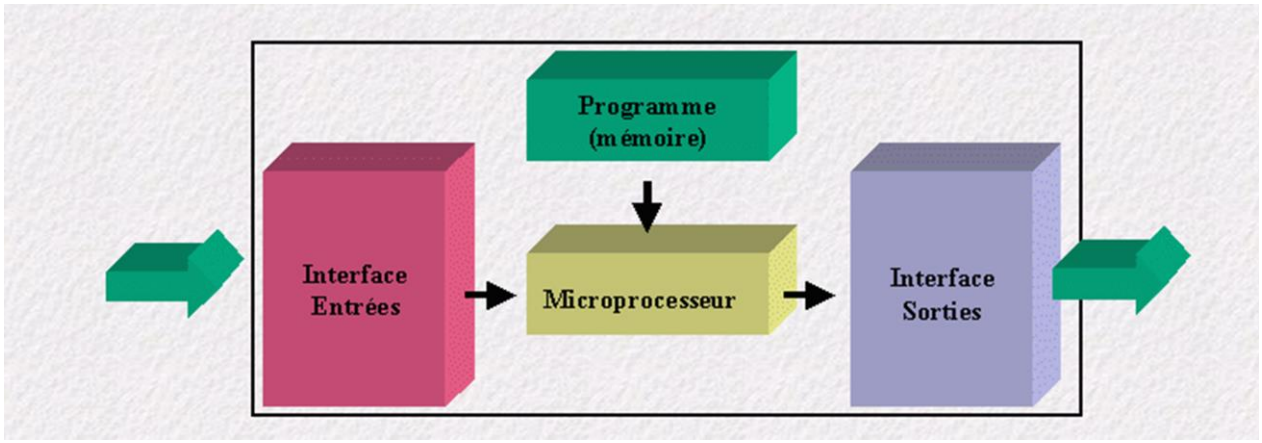


Les microprocesseurs : pourquoi ?

- il peut exécuter une grande variété de fonctions
  - parce qu'il est programmable. Il exécute une suite d'instructions qui peut être modifiée à souhait.
- son domaine d'application est pratiquement illimité
  - parce qu'on peut le coupler, via des interfaces d'entrée et de sortie, à une grande variété d'organes extérieurs
- production en très grandes séries et coût très faible.

Architecture de base d'un système à microprocesseur : Modèle de von Neumann

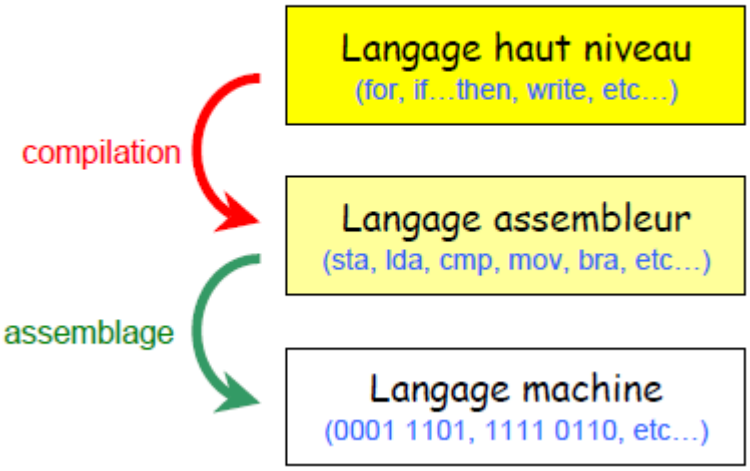
John von Neumann (1903 - 1957)

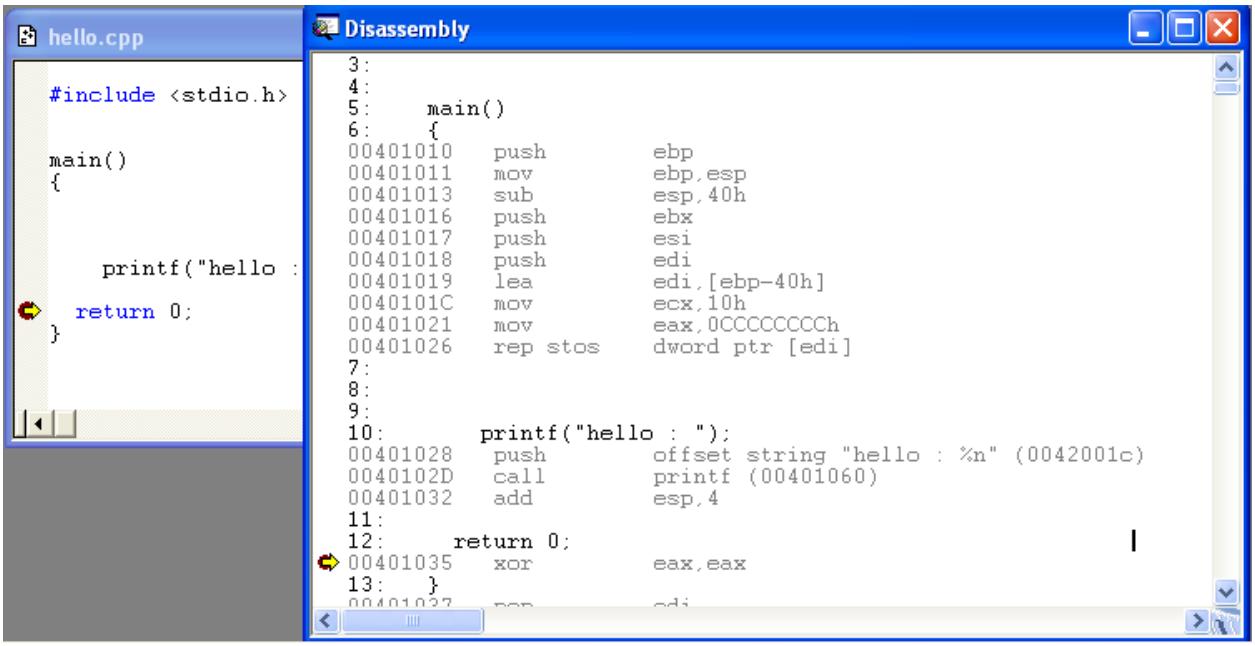


L'activité du microprocesseur est alors de répondre aux entrées pour produire des sorties, d'une façon déterminée par une ***séquence d'instructions*** (le programme) qui est stockée dans une **mémoire**.

L'ensemble des mots binaires, rangés en mémoire centrale, constitue le programme à exécuter. Cette exécution est réalisée par le CPU, qui reconnaît les instructions parmi le jeu d'instructions dont il est doté par le constructeur.

Langage de programmation





plusieurs types de programmation machine

L'introduction du programme dans la machine peut se faire sous plusieurs formes:

Sous forme binaire, c'est le langage machine :

C'est le langage compris par le microprocesseur

Sous forme hexadécimale :

Une forme un peu plus évoluée est le codage en hexadécimal. Il faut alors un clavier hexadécimal (0,.....,9,A,.....,F) pour introduire le programme en machine.

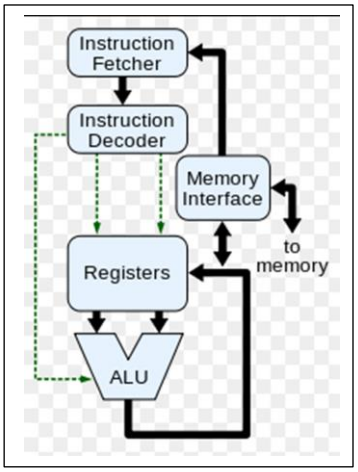
Par le langage d'assemblage :

Plus agréable : le langage d'assemblage, à partir d'un clavier alphanumérique (alphabet + chiffres), on introduit les instructions sous forme symbolique mnémonique. En langage assemblage, on écrit les instructions

mémoire	forme binaire	forme hexadécimale	forme d'assemblage	signification
n	10101001 b	A9 H	**ADD** A,09 H	Addition de 9 à A
n+1	01000000 b	40 H	**ROL** A	rotation à gauche de A
n+2	01100000 b	60 H	**MOV** (A9),A	mettre la valeur de A dans la case mémoire d'adresse A9H =(169)
n+3	10101001 b	A9 H		

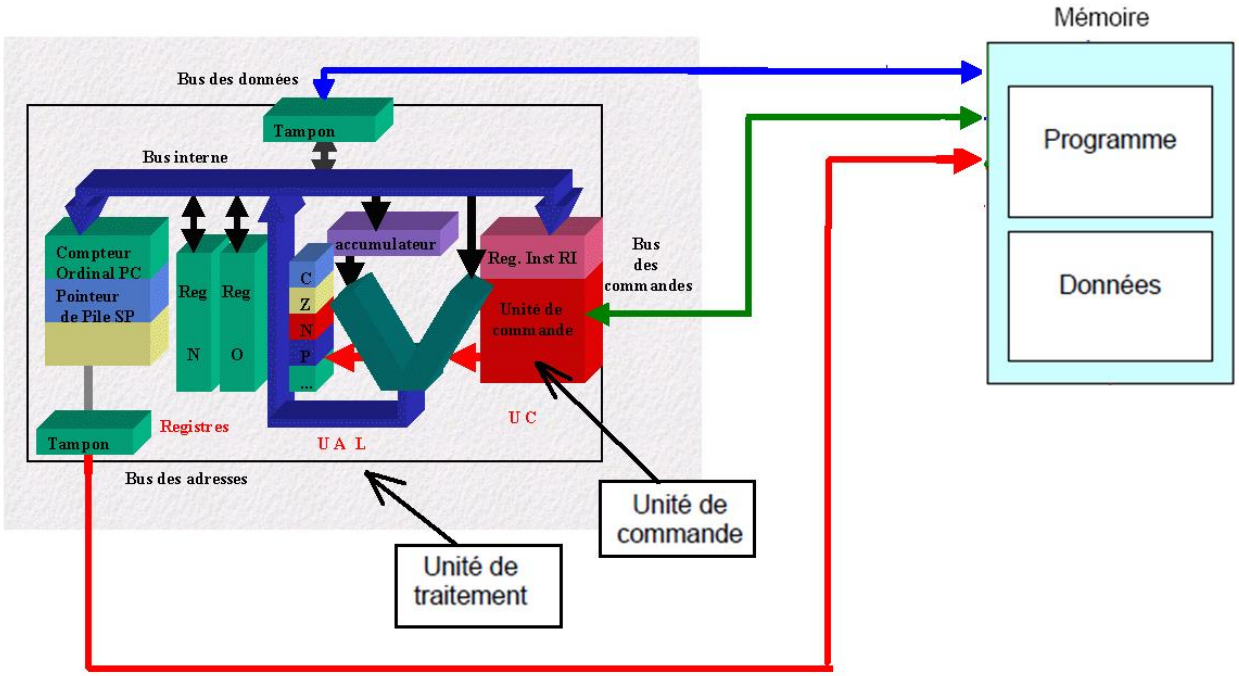
Procédure d'exécution d'une instruction

- Recherche de l'instruction (Fetch)
- Décodage (decode)
- Exécution (execute)



Architecture de base du microprocesseur

Le microprocesseur est constitué d'une unité de commande (UC) qui gère les codes et d'une unité de traitement qui exécute les instructions.



L'unité de commande :

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction. Comme chaque instruction est codée sous forme binaire, elle en assure le décodage pour enfin réaliser son exécution puis effectue la préparation de l'instruction suivante. Pour cela, elle est composée par :

- le compteur de programme constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de l'instruction à exécuter.
- le registre d'instruction et le décodeur d'instruction : chacune des instructions à exécuter est rangée dans le registre instruction puis est décodée par le décodeur d'instruction.
- Bloc logique de commande (ou séquenceur) : Il organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction des divers signaux de commande provenant du décodeur d'instruction ou du registre d'état par exemple. Il s'agit d'un automate réalisé soit de façon câblée (obsolète), soit de façon micro-programmée, on parle alors de microprocesseur.

L'unité de traitement

C'est le cœur du microprocesseur. Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions :

- L'Unité Arithmétique et Logique (UAL) est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage , etc…) ou arithmétique (Addition, soustraction).

- Le registre d'état est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateur d'état ou flag ou drapeaux. Dans un programme le résultat du test de leur état conditionne souvent le déroulement de la suite du programme. On peut citer par exemple les indicateurs de :
    - retenue (carry : C)
    - retenue intermédiaire (Auxiliary-Carry : AC)
    - signe (Sign : S)
    - débordement (overflow : OV ou V)
    - zéro (Z)
    - parité (Parity : P)

- Les accumulateurs sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.

Chapitre 2 : Introduction à la programmation en assembleur

Langage machine, langage interprété et langage compilé

⇨ CPU

⇨ cherche une instruction en mémoire, l'exécute, cherche la suivante…

⇨ Une instruction est une suite de bits contenant le code de l'information (addition, soustraction…) et ses opérandes (arguments de l'opération

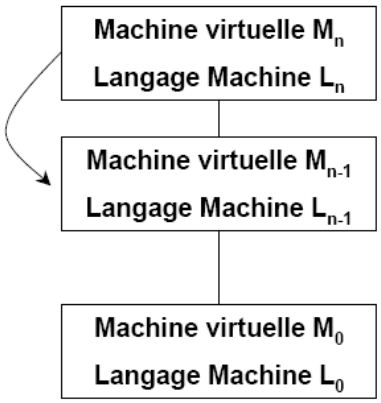
⇨ Pour éviter d'avoir à connaître ce langage on définit des langages de plus haut niveau

⇨ Interprétation :

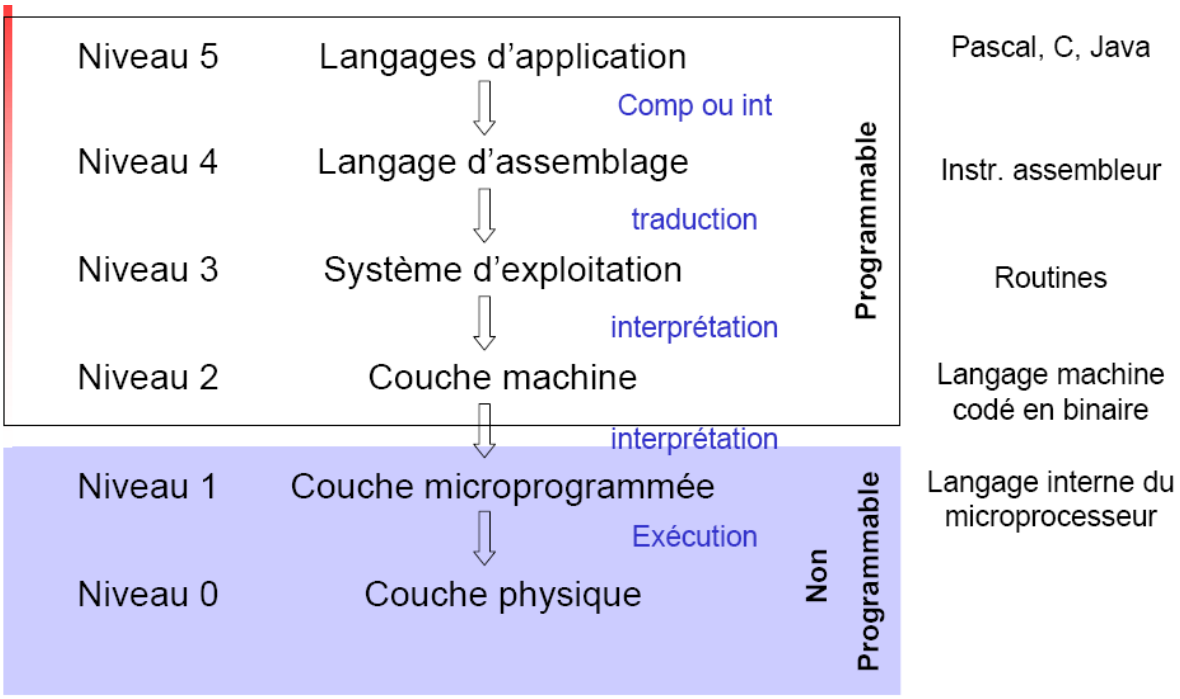
⇨ chaque instruction du langage Ln est traduite en séquence de L_{n-1} exécutable par M_{n-1}. Ré interprétation à chaque fois.

⇨ Compilation

⇨ programme de M_n traduit en entier en un programme en L_{n-1} exécutable par M_{n-1}

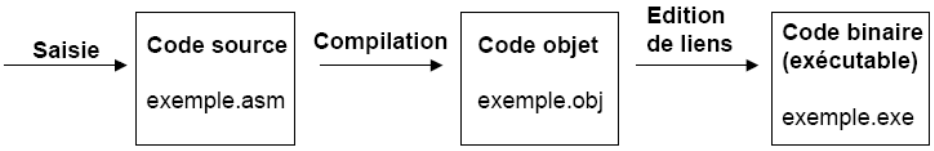


Fonctionnement en couches et niveaux de programmation



Processus d'assemblage

- **3 phases :**



1. **Saisie du code source avec un éditeur de texte**
2. **Compilation du code source**
3. **Edition des liens**
   - » **permet de lier plusieurs codes objets en un seul exécutable**
   - » **permet d 'inclure des fonctions prédéfinies dans des bibliothèques**

Exemple simple

Code-source et code-objet

Remplir une zone de mémoire de nombres entiers allant de 0 à la valeur de la variable l_tab-1

	Adr.	*Code généré*	*Instruction*			*Commentaire*
1	0000		m_seg	SEGMENT		; début du segment logique "m_seg"
2				ASSUME	CS:m_seg,	; reg. de base <-> segment logique
3					DS:m_seg	
4	= 004D		l_tab	EQU	77	; définir constante: longueur de la table
5	0000	4D [	table	DB	l_tab DUP(?)	; rés.de la mémoire pour la table
6		??]				
7	004D	B8 ---- R	debut:	MOV	AX,m_seg	; première instruction à exécuter
8	0050	8E D8		MOV	DS,AX	; initialiser DS = base du segment
9	0052	BB 00 4D		MOV	BX,l_tab	; initialiser BX = pointeur dans la table
10	0055	4B	rempl:	DEC	BX	; mise-à-jour pointeur (vers le début)
11	0056	88 9F 0000 R		MOV	table[BX],BL	; stocker (BL = byte inférieur de BX)
12	005A	83 FB 00		CMP	BX,0	; 0 = début de la table
13	005D	75 F6		JNZ	rempl	; continuer sauf si au début
14	005F	B4 4C		MOV	AH,4CH	; prép. la requête (4CH = fin exécution)
15	0061	CD 21		INT	21H	; faire la requête à MS/DOS
16	0063		m_seg	ENDS		; fin du segment
17				END	debut	; fin, définir l'adresse du début

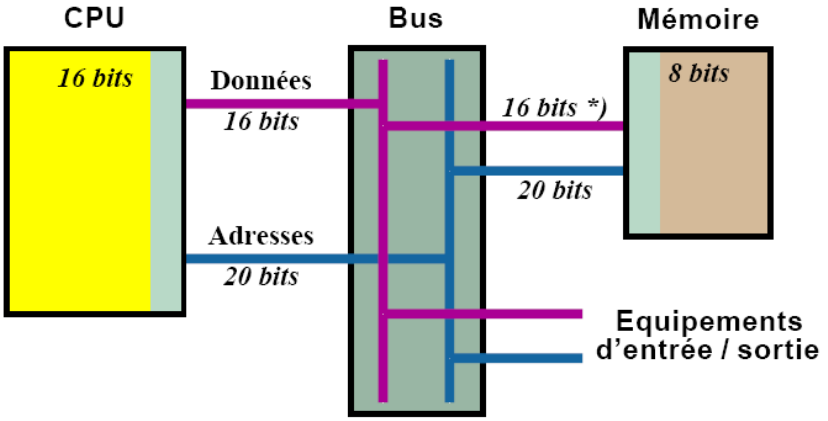
Segments et adresses

(impression du résultat de l'édition de liens)

```
Microsoft MACRO Assembler Version 3.00

Segments and Groups:

Name                    Size        Align       Combine Class
M_SEG                   0063        PARA        NONE

Symbols:

Name                    Type        Value       Attr
REMPL                   L NEAR      0055        M_SEG
DEBUT                   L NEAR      004D        M_SEG
TABLE                   L BYTE      0000        M_SEG       Length=004D
L_TAB                   Number      004D
49708 Bytes free

Warning      Severe
Errors       Errors
0            0
```

Architecture de la famille Intel 80xx:

- CPU à 16 bits
- mémoire organisée en bytes (8 bits)
- espace-mémoire avec des adresses représentées sur 20 bits



Note : Evolution ultérieure de la taille des mots: 16 bits ⇒ 32 (80386)⇒ 64 (Pentium) - et de la taille de l'espace d'adresses: 1M ⇒ 64 M (80386) ⇒ 4 G (80486, Pentium)

**) 8088 … 8 bits; 8086 … 16 bits; etc.*

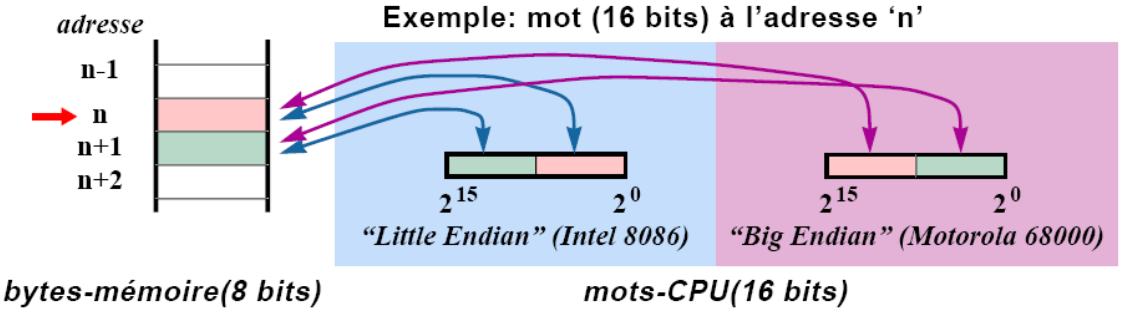
Organisation et fonctionnement de la mémoire

➔ **adresses** (= numérotation des positions de stockage)
en général bornées entre 0 et une limite supérieure

➔ **limite supérieure** de la mémoire (la plus grande adresse)
normalement imposée par la plus grande valeur qui peut être représentée en machine:

- soit dans un registre,
- soit dans une instruction.

La *séquence des bytes* stockés pour *représenter un mot en mémoire* dépend du type de la machine

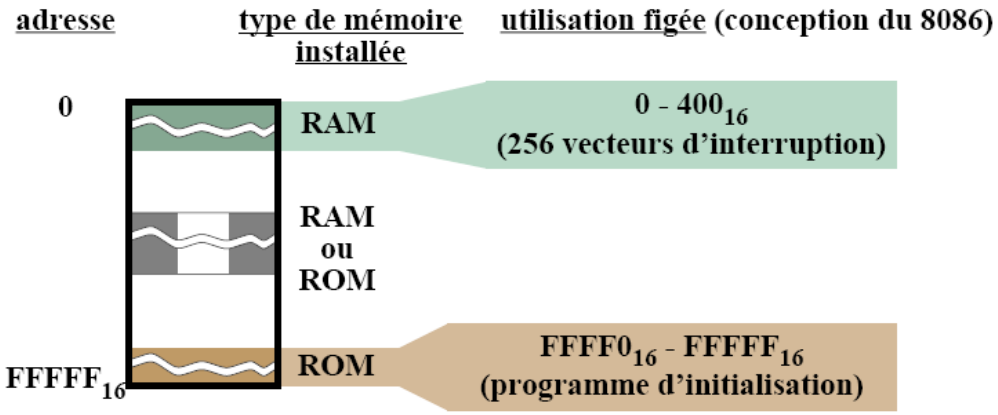


Organisation et fonctionnement de la mémoire

Implantation physique

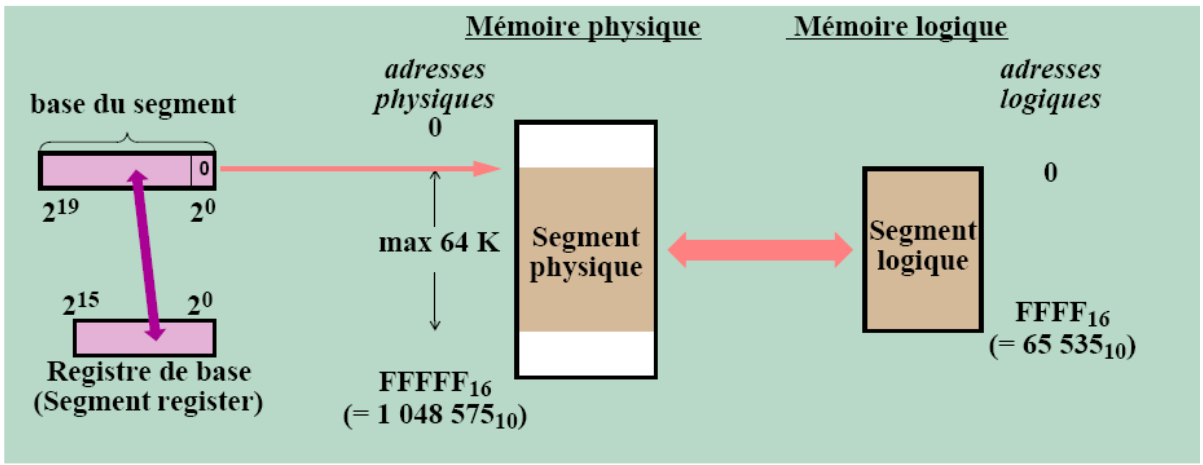
Implantation physique de la mémoire =
sous-ensemble de l'espace d'adresses,
effectivement équipé de mémoire (vive ou morte), organisé en zones contiguës

Plan de la mémoire (Intel 8086)



Segment de mémoire =
concept pour désigner une partie contiguë de la mémoire

Mémoire physique et mémoire logique



adresse physique		*adresse logique*		*base*
a_p	=	a_l	+	(registre de base $*$ 16_{10})

Représentation externe des adresses

convention de Intel — base : adresse-logique

exemple: 1C395 = 1000 : C395 ou 1B29 : 1105

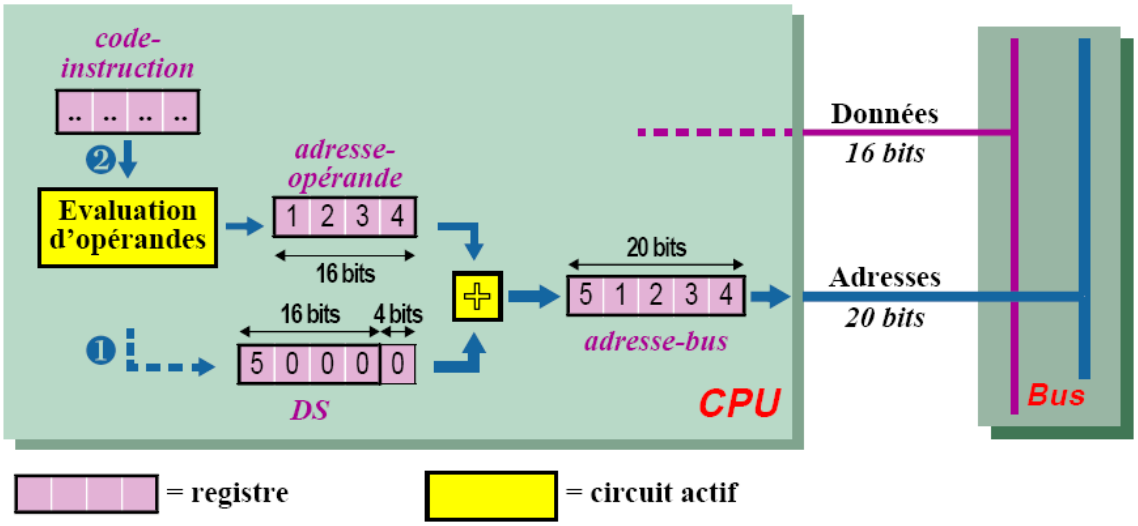
Organisation et fonctionnement de la mémoire

Génération une adresse-bus dans le CPU

Modèle fonctionnel

illustration:

```
                1  MOV   AX,5000H     ; préparer la valeur pour la base dans AX
étape ❶         2  MOV   DS,AX        ; définir la base du segment DS = 50000H
                   ...
étape ❷         3  MOV   AL,valeur    ; 'valeur' représente une donnée dont la place est
                                      ; réservée à l'adresse 1234H du 'data segment'
```



Caractéristiques du 8086

- Bus de données : 16 bits
- Bus d'adresse : 20 bits
- Registres : 16 bits

- 4 accumulateurs 16 bits
  - Accumulateur (AX)
  - Base (BX)
  - Counter (CX)
  - Accumulateur auxiliaire (DX)

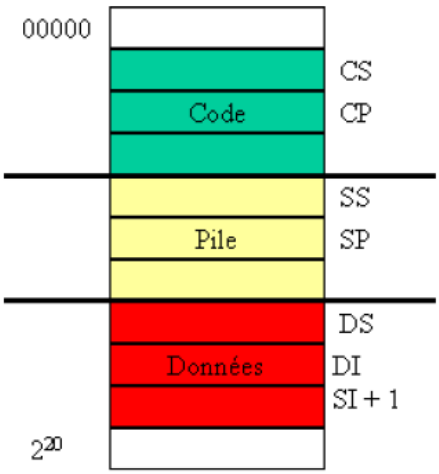
Certains registres 16 bits peuvent être utilisés comme deux registres 8 bits

AH	AL
BH	BL
CH	CL
DH	DL

- Registres accessibles sous forme de 2 info 8 bits
  - AX se décompose en AH (poids fort) et AL (poids faible de AX)…

Caractéristiques du 8086

- 4 accumulateurs 16 bits
  - AX, BX, CX ,DX
- Registres d'index :
  - Pointeur d'instruction (IP)
  - Index source ou destination(SI, DI)
  - Pointeur de Pile ou de base (SP, BP)
- 3+1 registres segment :
  - Segment de code (CS) : contient le prog en cours d'exécution
  - Segment data (DS) : contient les données du programme
  - Segment stack (SS) : contient des données particulières
  - Extra Segment (ES)



Segmentation de la mémoire

- Largeur du bus d'adresse = 20 bits
  - Possibilité d'adressage mémoire = 2^{20} = 1 Mo
- Le pointeur d'instruction fait 16 bits
  - Possibilité d'adresser 2^{16} = 64 Ko (ce qui ne couvre pas la mémoire)
- On utilise deux registres pour indiquer une adresse au processeur
  - Chaque segment débute à l'endroit spécifié par un registre spécial nommé registre segment.
  - Le déplacement permet de trouver une information à l'intérieur du segment.
  - CS:IP : lecture du code d'une instruction (CS registre segment et IP déplacement)
  - DS : accès aux données (MOV AX,[1045] = lecture du mot mémoire d'adresse DS:1045H)

Segmentation de la mémoire

- Registres de déplacement = sélectionner une information dans un segment.
  - Dans le segment de code CS : le compteur de programme IP joue ce rôle. CS:IP permet d'accéder à une information dans le segment de code.
  - Dans les segments de DS : les deux index SI ou DI jouent ce rôle. Le déplacement peut être aussi une constante. DS:SI ou DS:DI permettent d'accéder à une information dans le segment de données.
  - Dans le segment de pile SS le registre SP (stack pointer) et BP (base pointer) jouent ce rôle. SS:SP ou SS:BP permettent d'accéder à une information dans le segment de pile.

Les modes d'adressage

Méthodes d'accès aux données

- Adressage immédiat : l'opérande est une constante

```
mov ah,10h ; => Opcode B410
```

- Adressage direct : l'opérande est une case mémoire (registre ds par défaut)

```
mov al,[10h]    ; <=> mov al,[ds:10h] => Opcode A01000
mov ax,[es:10h] ;                     => Opcode 26A11000
```

- Adressage basé : l'opérande est une case mémoire dont l'adresse est donnée par bx (avec ds par défaut) ou bp (avec ss par défaut)

```
mov ah,[bx] ; <=> mov ah,[ds:bx] => Opcode 8A27
mov al,[bp] ; <=> mov al,[ss:bp] => Opcode 8A4600
```

- Adressage indexé : l'opérande est une case mémoire dont l'adresse est donnée par si ou di (avec ds par défaut, sauf mnemonic spécifique)

```
mov ah,[si] ; <=> mov ah,[ds:si] => Opcode 8A24
```

- Adressage basé et indexé

```
mov ah,[bx+di] ; <=> mov ah,[ds:bx+di] => Opcode 8A21
mov [bp+si],ah ; <=> mov [ss:bp+si],ah => Opcode 8822
```

- Adressage basé avec déplacement

```
mov ah,[bx+123h] ; <=> mov ah,[ds:bx+123h] => Opcode 8AA72301
```

Organisation d'un programme en assembleur
Cas de l'assembleur : flat assembler

Structure d'un programme en assembleur pour générer un programme format ' .com '

COM template - simple and tiny executable file format, pure machine code.

La directive ' name' indique le titre du programme

La directive ' org' indique l'adresse de début du programme

Zone des variables

Point d'entrée du programme

Zone programme

Instruction de retour à l'OS

```
; The easiest way to print out "Hello, World!"

name "hi"

org 100h

jmp start       ; jump over data declaration

msg:    db      "Hello, World!", 0Dh,0Ah, 24h

start:  mov     dx, msg  ; load offset of msg into dx.
        mov     ah, 09h  ; print function is 9.
        int     21h      ; do it!

        mov     ah, 0
        int     16h      ; wait for any key....

ret ; return to operating system.
```

Structure d'un programme en assembleur pour générer un programme format ' .exe '

EXE template - advanced executable file. header: relocation, checksum.

La directive ' name' indique le titre du programme

La directive ' data segment' indique la zone du segment de données pour les variables

La directive ' stack segment' indique la zone du segment de la pile

La directive ' code segment' indique la zone du segment de code

Point d'entrée du programme

Zone du programme

La directive 'ends' indique la fin d'un segment

La directive END est toujours la dernière ligne du code source.

```
; The easiest way to print out "Hello, World!"

name "hi"

data segment
    msg:    db      "Hello, World!", 0Dh,0Ah, 24h
ends

stack segment
    dw   128  dup(0)
ends

code segment

start:

    ; add your code here


        mov     dx, msg  ; load offset of msg into dx.
        mov     ah, 09h  ; print function is 9.
        int     21h      ; do it!
        mov     ah, 0
        int     16h      ; wait for any key....


        mov ax, 4c00h    ; exit process
        int 21h

ends

end start
```

Déclaration des variables

Les principaux types de données

Size (bytes)	Define data
1	db file
2	dw du
4	dd
6	dp df
8	dq
10	dt

Déclaration des données BYTE

```
valeur1     db    "A"
valeur2     db    0
valeur3     db    255
valeur4     db    ?

liste       db    10,20,30,40

zone1       db    20 DUP(0)
zone2       db    20 DUP( ?)
zone3       db    20 DUP("A")
```

Déclaration des données WORD et SWORD

```
mot1:       dw    65535
mot2:       dw    ?
```

Déclaration des chaînes de caractères

```
Bienvenue:  db    "Bonjour"
            db    " veuillez activer"
            db    " une touche pour"
            db    " démarrer. " 0dh,0ah
```

Appels système BIOS et MSDOS

Pour réaliser les opérations standards (affichage, saisie, …) le système d'exploitation (ici DOS) fournit des fonctions pré-écrites :

– Affichage d'un caractère	Mov DL,"A"	; caractère
	Mov AH, 2	; fonction n°2
	Int 21h	; appel système
– Saisie d'un caractère (avec écho)	Mov AH,1	
	Int 21h	; (résultat dans AL)
– Saisie d 'un caractère (sans écho)	Mov AH,7	
	Int 21h	; (résultat dans AL)
– arrêt du programme	Mov AH, 4Ch	; A mettre a la fin de
	Int 21h	; chaque programme

Jeux d'instructions (affectation, arithmétique, logique)

⇨ Instruction d'affectation *(Transfert CPU Mémoire)*	: MOV
⇨ Instructions arithmétiques *(Opération Acc / Donnée)*	: INC (incrémentation) : DEC (décrementation) : ADD (addition) : SUB (soustraction) : CMP (soustraction sans sauvegarde) : NEG
⇨ Instructions logiques	: NOT, OR, XOR : AND, TEST (= AND sans sauvegarde) : SHL (SHR), SAL (SAR) : ROL (ROR), RCL (RCR)

⇨ Instructions pour l'accès aux entrées sorties : **IN et OUT**

Jeux d'instructions (décalage et rotation)

⇨ Décalage vers la gauche ou vers la droite les bits de l'accumulateur.

- ⇨ opérations utilisées pour décoder bit à bit des données
- ⇨ ou pour diviser ou multiplier rapidement par une puissance de 2.
- ⇨ En effet : décaler AX de n bits vers la gauche revient à le multiplier par 2^n
- ⇨ De même, un décalage vers la droite revient à diviser par 2^n.

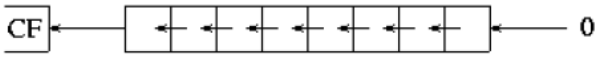
⇨ Ces opérations peuvent opérer sur les registres AX ou BX (16 bits) ou sur les registres de 8 bits AH, AL, BH et BL.

- ⇨ SHL, SHR, ROL, ROR, RCL, RCR…

Les instructions de décalage

SHL *registre*, 1 *(Shift Left)*

Décale les bits du registre d'une position vers la gauche. Le bit de gauche est transféré dans l'indicateur CF. Les bits introduits à droite sont à zéro.



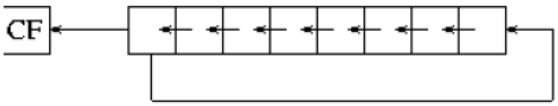
SHR *registre*, 1 *(Shift Right)*

Comme SHL mais vers la droite. Le bit de droite est transféré dans CF.
SHL et SHR peuvent être utilisé pour multiplier/diviser des entiers *naturels* (et non des relatifs car le bit de signe est perdu)

Les instructions de rotation

ROL *registre*, 1 *(Rotate Left)*

Rotation vers la gauche : le bit de poids fort passe à droite, et est aussi copié dans CF. Les autres bits sont décalés d'une position.

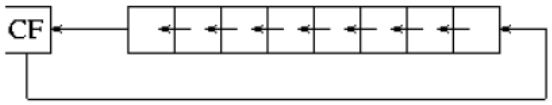


ROR *registre*, 1 *(Rotate Right)*

Comme ROL, mais à droite.

RCL *registre*, 1 *(Rotate Carry Left)*

Rotation vers la gauche en passant par l'indicateur CF. CF prend la place du bit de poids faible; le bit de poids fort part dans CF.



RCR *registre*, 1 *(Rotate Carry Right)*

Comme RCL, mais vers la droite.

RCL et RCR sont utiles pour lire bit à bit le contenu d'un registre[7].

Jeux d'instructions (les branchements)

- Branchement : JMP
- Branchements conditionnels : JE/ JZ (JNE/ JNZ) : Jump if zero
  : JO (JNO) : Jump if overflow
  : JS (JNS) : Jump if sign
- Comparaison de valeurs

CMP AX, BX suivi d 'un test :

	(entiers naturels)	(complément à 2)
AX > BX ?	JA (>= JAE)	JG (JGE)
AX < BX ?	JB (JBE)	JL (JLE)

Principe des instructions de branchement

- Le processeur exécute une instruction en mémoire puis passe à celle qui suit en mémoire : séquentiel
  - Besoin de faire répéter au processeur une suite d'instructions
  - Besoin de déclencher une action qui dépend d'un test
- Utilisation d'une instruction de branchement ou saut
  - On indique au processeur l'adresse de la prochaine instruction
- On distingue deux catégories de branchements
  - le saut est toujours effectué (sauts *inconditionnels*)
  - il est effectué seulement si une condition est vérifiée (sauts *conditionnels*).

Branchement inconditionnel

- Principale instruction de saut inconditionnel = **JMP**
- L'opérande de JMP est un *déplacement*, c'est à dire une valeur qui va être ajoutée à IP. L'action effectuée par **JMP** est :
  - IP = IP + *déplacement*
- Le déplacement est un entier relatif sur codée 8 bits. La valeur du déplacement est :
  - *déplacement* = adr. instruction visée - adr. instruction suivante

---

- Exemple : le programme suivant écrit indéfiniment la valeur 0 à l'adresse 0140H.
- La première instruction est implantée à l'adresse 100H.

```
Adresse Contenu MP Langage Symbolique Explication en français

0100    B8 00 00      MOV AX, 0            met AX a zéro
0103    A3 01 40      MOV [140], AX        écrit à l'adresse 140
0106    EB FC         JMP 0103             branche en 103
0107                  xxx -> instruction jamais exécutée
```

- Le déplacement est ici égal à FCH, c'est à dire -4 (=103H-107H).

Branchement conditionnel

⇨ Les instructions de branchement conditionnels utilisent les *indicateurs*,

⇨ bits spéciaux positionnés par l'UAL après certaines opérations.
⇨ nous étudierons ici les indicateurs nommés ZF, CF, SF et OF.

ZF : *Zero Flag*

Cet indicateur est mis à 1 lorsque le résultat de la dernière opération est zéro. Sinon, ZF est positionné à 0.

CF : *Carry Flag*

C'est l'indicateur de report (retenue). Il est positionné par les instructions ADD, SUB et CMP (entiers naturels).
CF = 1 s'il y a une retenue

⇨ **SF** : *Sign Flag*

SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1; sinon SF=0. SF est utile lorsque l'on manipule des entiers relatifs, car le bit de poids fort donne alors le signe du résultat.

⇨ **OF** : *Overflow Flag (*Indicateur de débordement)

OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable *en relatif* dans l'accumulateur (par exemple si l'addition de 2 nombres positifs donne un codage négatif).

⇨ CMP = SUB, mais ne stocke pas le résultat de la soustraction (positionner les indicateurs)

⇨ CMP AX, 5 : ZF = 1 si AX contient 5, et ZF = 0 si AX ≠ 5.

Exemples d'instructions de branchement conditionnel

ZF	SF	CF	OF	PF	opération	condition de branchement (à la suite de l'exécution de l'instruction "CMP A,B")
1					JE,JZ	A=B
0					JNE,JNZ	A≠B
	1				JS	
	0				JNS	
		1			JB,JNAE	A<B
		0			JNB,JAE	A≥B
CF∨ZF = 1					JBE,JNA	A≤B
CF∨ZF = 0					JNBE,JA	A>B
SF∀OF = 1					JL,JNGE	A<B
SF∀OF = 0					JNL,JGE	A≥B
(SF∀OF)∨ZF = 1					JLE,JNG	A≤B
(SF∀OF)∨ZF = 0					JNLE,JG	A>B
			1		JO	dépassement arithmétique
			0		JNO	pas de dépassement arithmétique
				1	JP,JPE	parité paire ("even")
				0	JNP,JPO	parité impaire ("odd")

comparaison de valeurs sans signe

comparaison de valeurs avec signe

∨ = *"ou" inclusif*
∀ = *"ou" exclusif*

Notes : Caractères mnémoniques : *Equal, Zero, Below, Above, Less-than, Greater-than, Not*

Branchements pour itérations

Instruction	Actions (phases)	
	1. mise-à-jour de CX	*2. branchement si :*
LOOP	CX := CX-1	CX≠0
LOOPZ,LOOPE	CX := CX-1	(CX≠0)∧(ZF=1)
LOOPNZ,LOOPNE	CX := CX-1	(CX≠0)∧(ZF=0)
JCXZ	pas d'action	CX=0

Exemples d'utilisation des instructions de branchement conditionnel

Recherche du plus grand de deux nombres entiers non signés : (le résultat dans DX)

```
        mov     dx,ax
        cmp     ax,bx       ; si AX >=BX alors
        jae     L1          ; sauter à L1
        mov     dx,bx
L1 :
```

Recherche du plus petit de deux nombres entiers non signés : (le résultat dans DX)

```
        mov     ax, V1
        cmp     ax,V2
        jbe     L1
        mov     ax,V2
L1 :
        cmp     ax,V3
        jbe     L2
        mov     ax,V3
L2 :
```

Traduction en assembleur des structures conditionnelles classiques

Instruction IF

```
If  ( expression )
   Bloc instructions 1
else
   Bloc instructions 2
```

Exemple :

```
if ( op1 == op2)
{
      X = 1 ;
      Y= 2 ;
}
else
{
      X = 2 ;
      Y = 1 ;
 }
```

Exemple de traduction en assembleur

```
        mov     ax,op1
        cmp     ax,op2
        je      L1
        mov     X,2
        mov     Y,1
        jmp     L2
L1 :    mov     X,1
        mov     Y,2
L2 :
```

Boucle WHILE

```
while ( expression )
   Bloc instructions 1
```

Exemple :

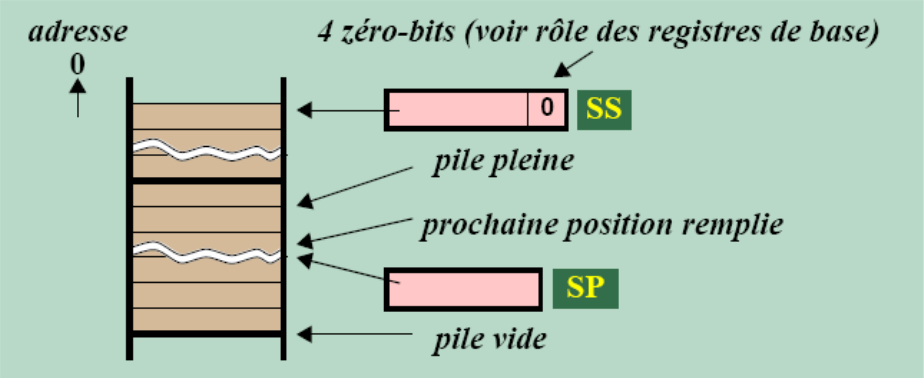
```
while ( val1 < val2)
{
      val1++  ;
      val2-- ;
 }
```

Exemple de traduction en assembleur

```
        mov     ax,val1
L1 :
        cmp     ax,val2
        jnl     L2
        inc     val1
        dec     val2
        jmp     L1
L2 :
```

Opérations utilisant la pile

Utilisation d'une pile



Ajouter une valeur sur la pile :

1. SP := SP - 2
2. déposer la valeur à l'adresse déterminée par SP

Retirer une valeur de la pile :

3. Retirer la valeur de l'adresse déterminée par SP
4. SP := SP + 2

Note : la pile "grandit" vers le bas de la mémoire (Intel 8086!) - choix du constructeur!

Opérations utilisant la pile

`PUSH` *opérande-mot*	1. SP := SP - 2 2. cont (SS:SP) := "*op.-mot*" [1)]
`POP` *opérande-mot*	1. "*op.-mot*" := cont (SS:SP) [1)] 2. SP := SP + 2
`CALL` *opérande-near*	1. Push IP 2. IP := "op.-near"
`CALL` *opérande-far*	1. Push CS 2. Push IP 3. CS:IP := "*op.-far*"
`INT` *valeur*	1. Push flags [2)] 2. Push CS 3. Push IP 4. IF := 0 TF := 0 5. CS:IP := cont ("*valeur*" * 4) [1) 3)]
`RET`	1. Pop IP 2. Pop CS (seulement si "FAR")
`IRET`	1. Pop IP 2. Pop CS 3. Pop flags [2)]

1) *cont* (*pos*) = contenu de la position-mémoire "*pos*"

2) *flags* = contenu du registre d'état ("*flag register*")

3) adresse absolue (valeur * 4)

Les procédures en assembleur

- **Déclaration d'une procédure :**
  - appel proche (near) : la procédure et le programme sont dans le même segment de code
  - appel lointain (far) : la procédure et le programme sont dans deux segments de code différents

```
Moyenne PROC [NEAR/FAR]
        instructions
Moyenne ENDP
```

- **Appel de la procédure dans le programme**

```
Call Moyenne
```

Comment l'unité de traitement arrive-t-elle à retourner au programme principal à la fin de la fonction ?

- **Au moment de l'appel de la fonction, l'adresse de l'instruction suivante est sauvegardée dans la pile :**
  - appel proche (near) : sauvegarde de **IP**
  - appel lointain (far) : sauvegarde de **CS** puis de **IP**

⇒ *A la fin de la procédure, l 'UT récupère les valeurs sauvegardées pour retourner au programme principal*

Exemple de programme avec procédures

```
TITLE Somme d un tableau                (SumArray.asm)

; Calcule la somme des valeurs d'un tableau de mots.

pile            segment  stack
                db  256 dup(0)
pile            ends

data            segment
array   dw 1000h,2000h,3000h,4000h
thesum  dw ?
data            ends

CODE          SEGMENT
              assume  cs:CODE,ds:data,ss:pile

debut:        mov ax,data               ; data segment address
              mov ds,ax                 ; copy to DS
              mov si,OFFSET  array      ; Adresse du tableau
              mov cx,0004               ; Compteur de boucle
              call SumArray             ; appel de la procédure de calcul
              mov thesum,ax             ; stockage du résultat

              mov ah,4ch                ; exit process
              int 21h                   ; call MS-DOS function
```

Le passage des paramètres entre le programme principal et les procédures en assembleur

- 2 méthodes de passage de paramètres : * par registre * par la pile

- Passage de paramètres par registre :
  - les paramètres d'entrée de la procédure sont mis dans des registres avant l 'appel de la procédure
  - les paramètres de sortie sont aussi rangés dans des registres
  - avantage = rapidité inconvénients = peu de registres pas de « portabilité »
  - Ex:

```
; programme principal
        ...
        Mov AX, N1
        Mov BX, N2
        Call Moyenne
        Mov Res, AX
        ...
```

```
Moyenne PROC
        Add AX, BX
        SHR AX, 1
        Ret
Moyenne ENDP
```

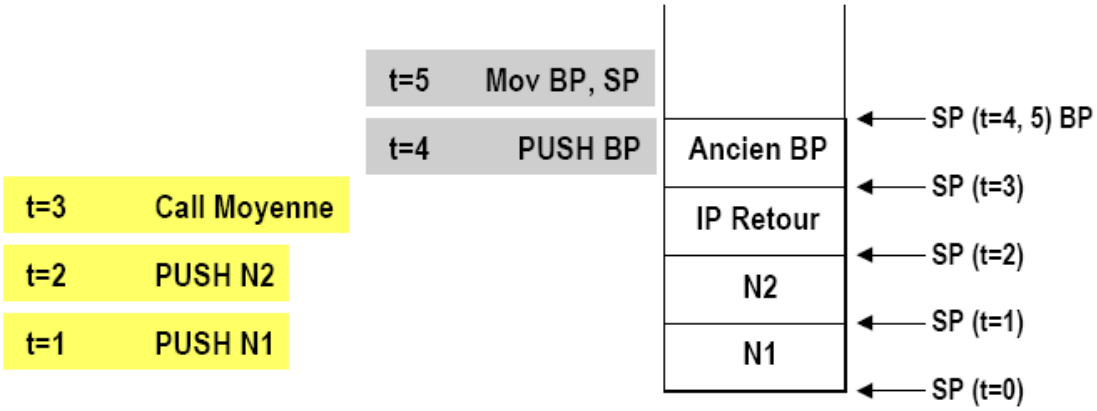
Ret = dépilement de IP pour retourner au programme principal

- Passage de paramètres par la pile :
  - les paramètres d'entrée sont empilés avant l 'appel de la procédure
  - les paramètres de sortie sont dépilés par le programme principal
  - avantage = « portabilité » inconvénients = récupération des paramètres plus « lourde »
  - Ex:

```
; programme principal
        ...
        PUSH N1
        PUSH N2
        Call Moyenne
        POP Res
        ...
```

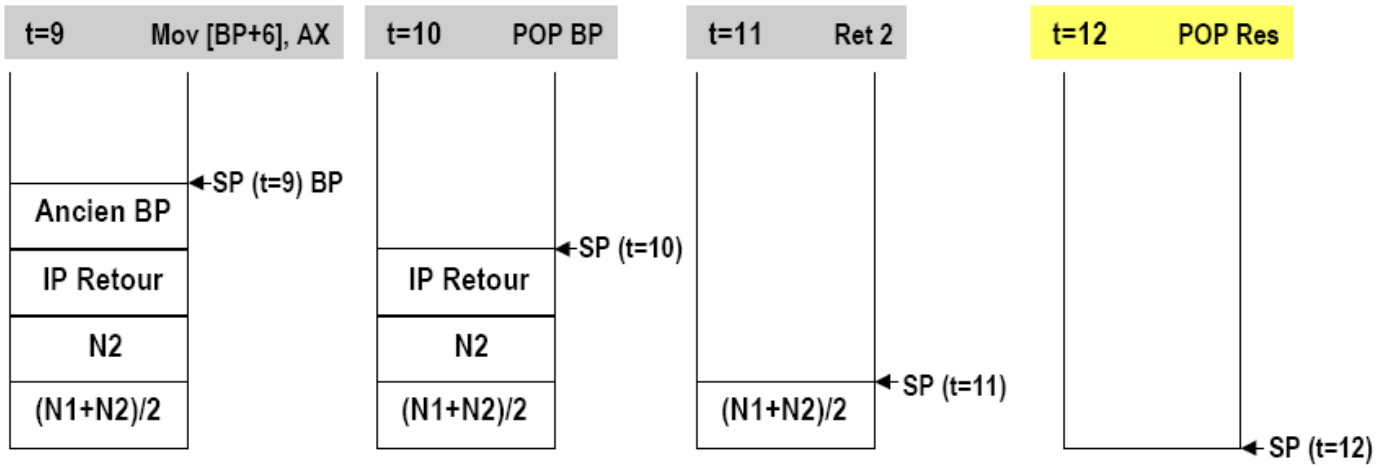
```
Moyenne PROC
        PUSH BP
        Mov BP, SP
        Mov AX, [BP+4]
        Add AX, [BP+6]
        SHR AX, 1
        Mov [BP+6], AX
        POP BP
        Ret 2
Moyenne ENDPc
```

- **Passage de paramètres par la pile : évolution de la pile**



- **A quoi sert BP ?**
  - **BP = référence de la pile au début de la procédure**
  - **permet d'avoir une référence fixe pour récupérer les paramètres**
    ici : [BP+4] pour N2 et [BP+6] pour N1

– **Sauvegarde du résultat et « nettoyage de la pile » :**



Sauvegarde du résultat dans le bas de la pile

BP reprend sa valeur initiale

Ret 2 dépile IP pour retourner au prog. principal et efface aussi la case suivante (Ret 2 = 2 cases)

Dépilement et rangement du résultat dans la variable Res

Les types de paramètres en assembleur

- On peut utiliser deux paramètres différents :
  - la valeur de la variable
  - l 'adresse de la variable

```
; passage des valeurs par registre
        ...
        Mov AX, N1
        Mov BX, N2
        Call Moyenne_par_valeur
        Mov Res, AX
        ...
```

```
Moyenne_par_valeur PROC
        Add AX, BX
        SHR AX, 1
        Ret
Moyenne_par_valeur ENDP
```

(Idem pour le passage par pile)

```
; passage des adresses par registre
        ...
        Mov SI, Offset N1
        Mov DI, Offset N2
        Mov BX, Offset Res
        Call Moyenne_par_adresse
        ...
```

```
Moyenne_par_adresse PROC
        Mov AX, [SI]
        Add AX, [DI]
        SHR AX, 1
        Mov [BX],AX
        Ret
Moyenne_par_adresse ENDP
```

ATTENTION : Une procédure ne doit pas modifier les registres du microprocesseur, sauf ceux qui sont utilisés pour retourner les paramètres.

Procédure sale = modifie les registres

```
Moyenne PROC
        PUSH BP
        Mov BP, SP
        Mov AX, [BP+4]
        Add AX, [BP+6]
        SHR AX, 1
        Mov [BP+6], AX
        POP BP
        Ret 2
Moyenne ENDP
```

A la fin de la procédure, AX est modifié

Procédure propre = ne modifie pas (ou restaure en fin de procédure) les registres

```
Moyenne PROC
        PUSH BP
        Mov BP, SP
        PUSH AX
        Mov AX, [BP+4]
        Add AX, [BP+6]
        SHR AX, 1
        Mov [BP+6], AX
        POP AX
        POP BP
        Ret 2
Moyenne ENDP
```

AX est sauvegardé en début de procédure et restauré à la fin

Les interruptions

- **La plupart des machines disposent d'un mécanisme pour indiquer les évènements extérieurs (clavier, disque dur, . . .), appellé interruptions**

- **Leur fonctionnement consiste à interrompre le programme en cours d'exécution et de lancer un autre bout de code (appellé gestionnaire) pour gérer l'évènement**

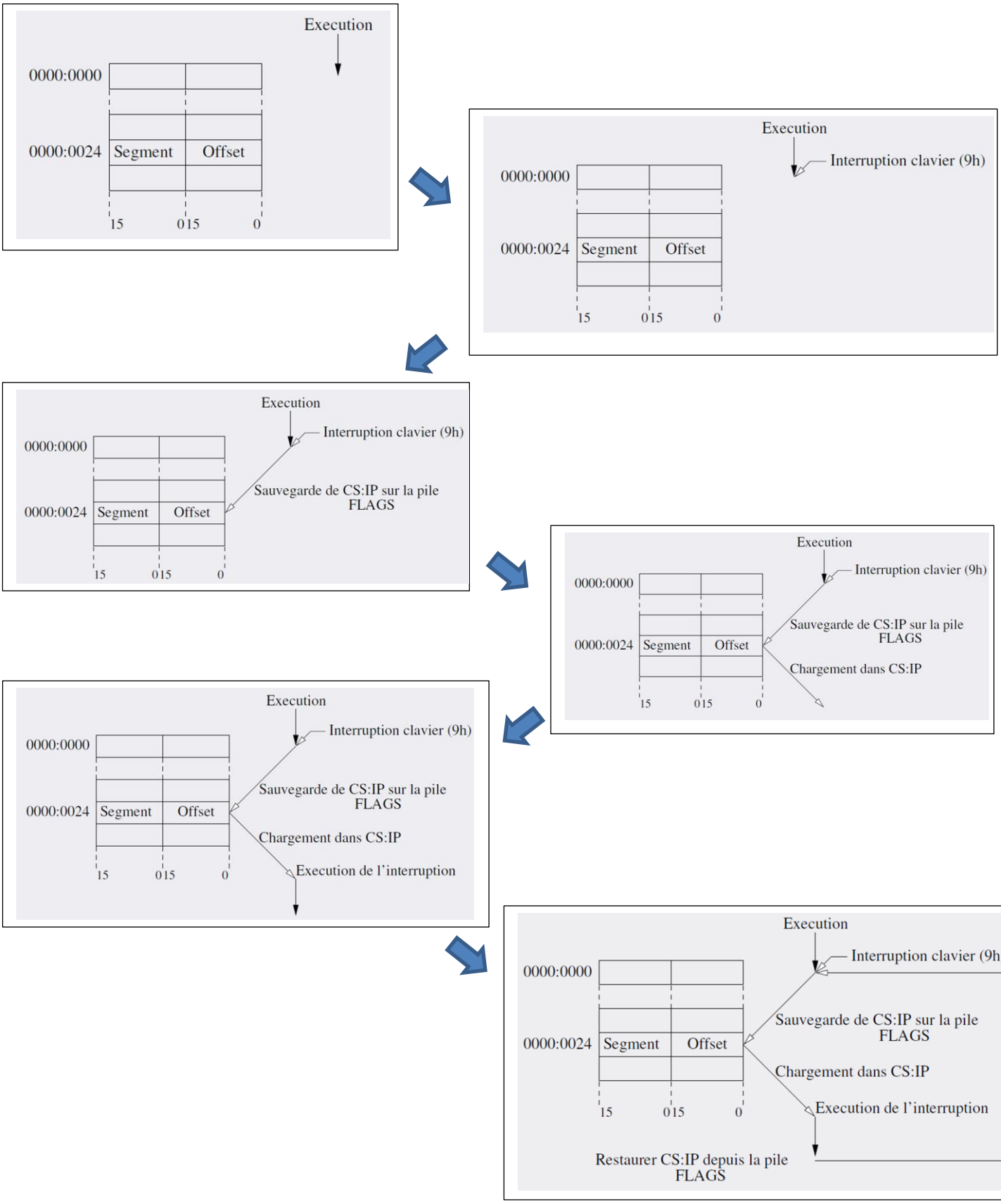
- **Sur le 8086, les interruptions sont spécifiées par un numéro sur 8 bits :il y'a donc 256 interruptions différentes**

- **Le programme à exécuter lors d'une interruption est donnée par une adresse segment : offset**

- **Une table des adresses des interruptions se situent dans la mémoire entre les adresses 0 et 1024 (inclus)**

- **Le numéro d'interruption sert d'indice dans cette table pour trouver l'adresse du programme à lancer**

Schéma d'exécution des interruptions

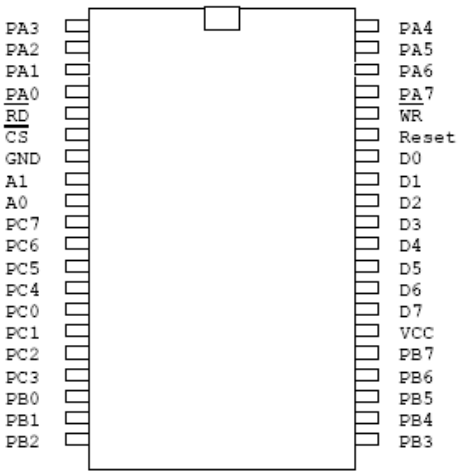


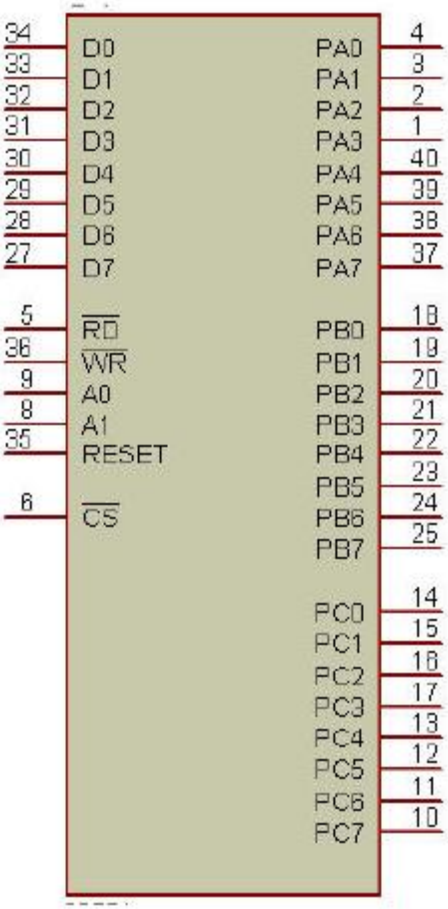
LE Circuit d'Entrée / sortie
PIA 8255A

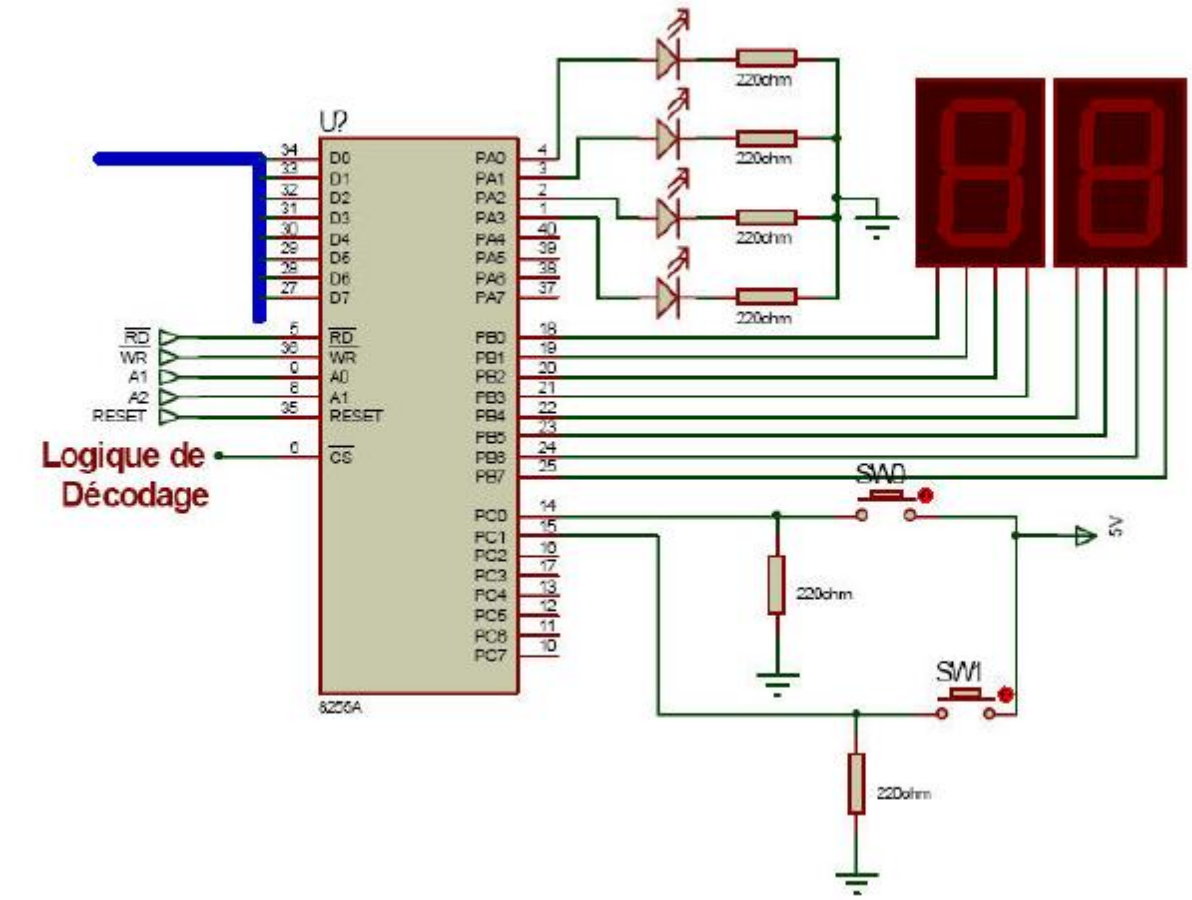
I/ Introduction:

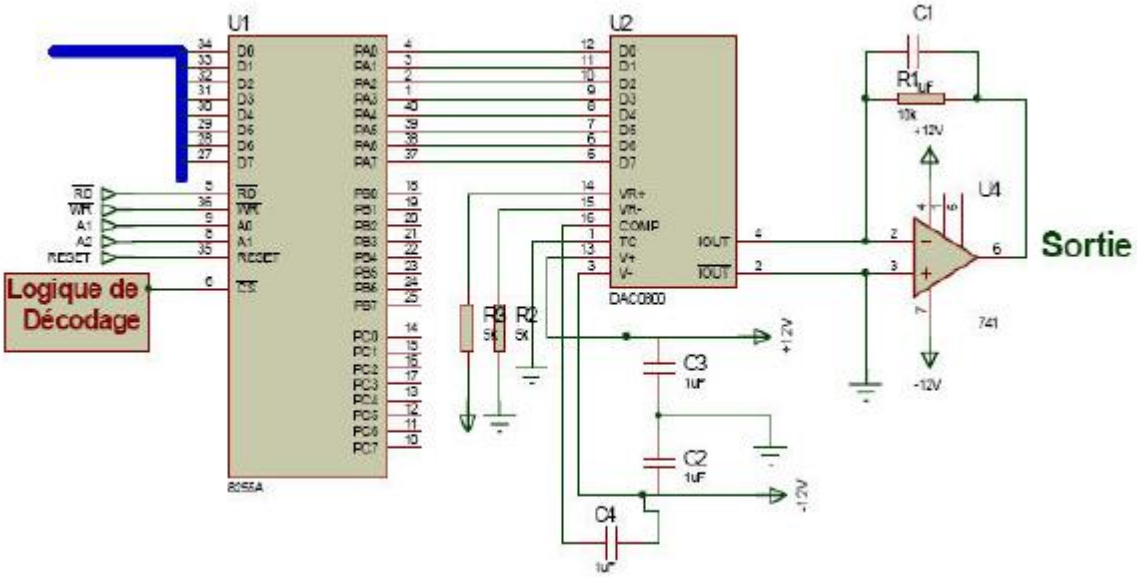
Le 8255A est un circuit d'E/S destiné à être utilisé avec les microprocesseurs Intel.
Il possède 24 broches qui peuvent être individuellement programmés en 2 groupes de 12 et il peut utiliser 3 modes fondamentaux de fonctionnement: mode 0, mode 1 et mode 2.

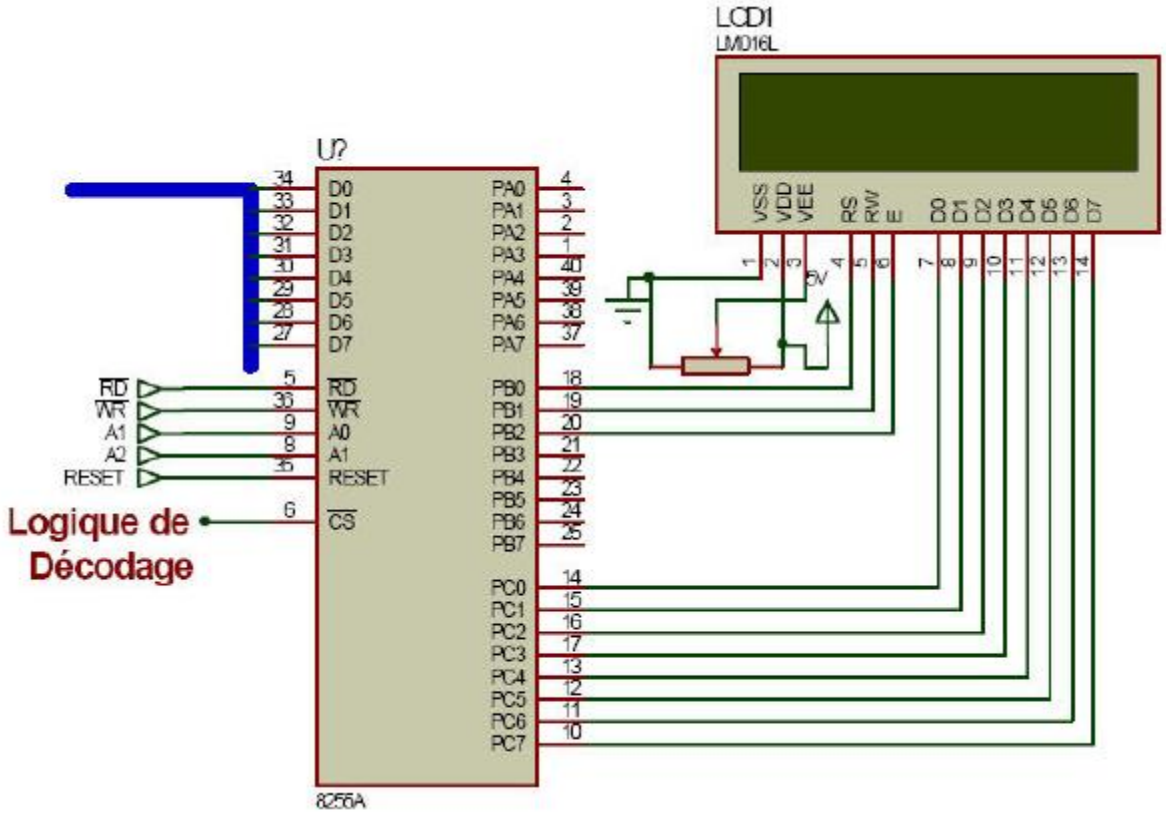
II/ Description fonctionnelle du 8255:

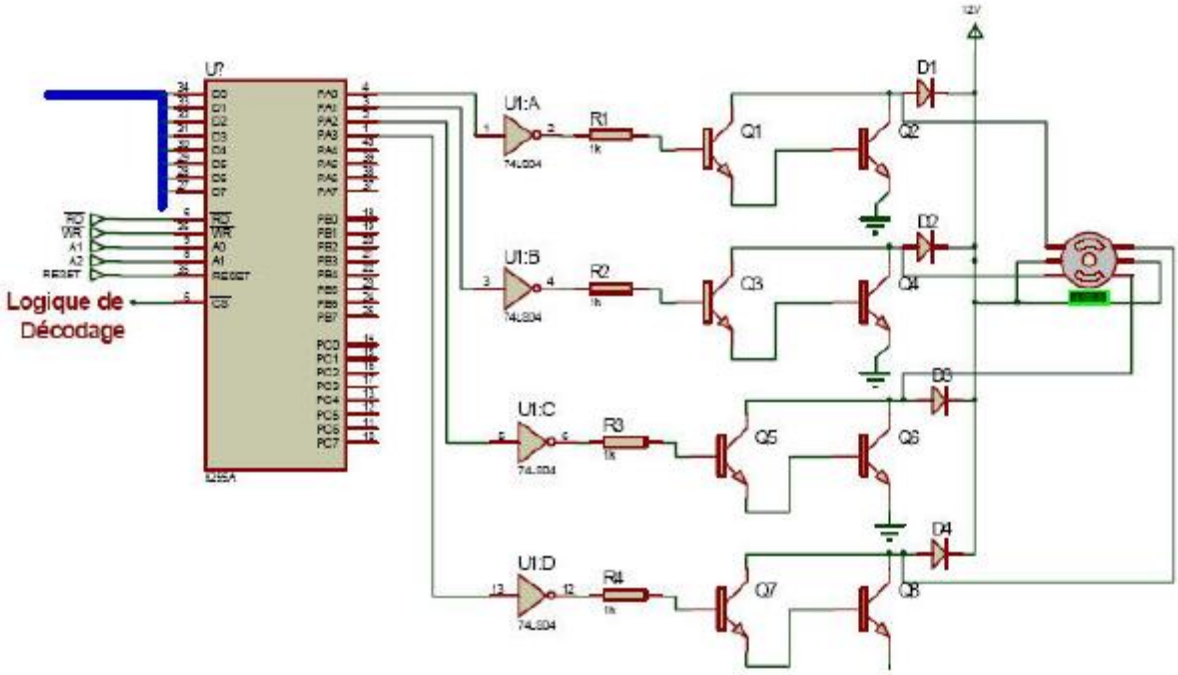





U?
D0
D1
D2
D3
D4
D5
D6
D7
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
RD
WR
A0
A1
RESET
CS
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
8255A
220ohm
220ohm
220ohm
220ohm
Logique de Décodage
SW0
SW1
5V
220ohm
220ohm


U1
U2
C1
R1
1uF
10k
+12V
U4
741
Sortie
-12V
DAC0800
IOUT
IOUT
COMP
VR+
VR-
VC
V+
V-
R3
5K
R2
5K
C3
1uF
C2
1uF
C4
1uF
+12V
-12V
Logique de Décodage
8255A

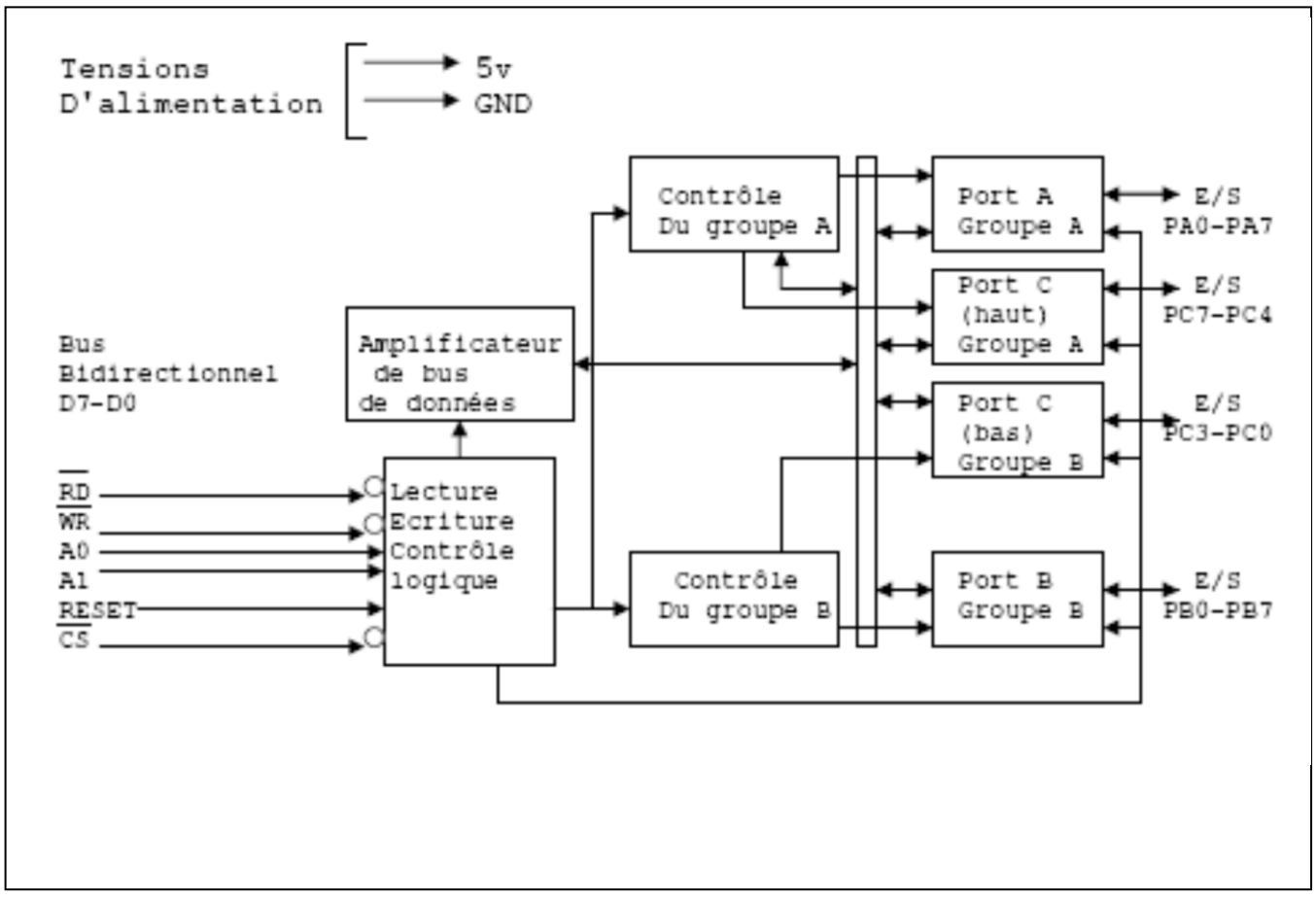

LCD1
LM016L
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
5V
U?
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A1
A2
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
Logique de
Décodage
8255A


U?
Logique de
Décodage
U1:A
U1:B
U1:C
U1:D
74LS04
R1
R2
R3
R4
1k
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
D1
D2
D3
D4
12V

-1 Description générale des fonctions de broches:

N° Broche	Nom de broche	Fonction
5	RD/	Un signal bas sur cette entrée autorise le passage de donnée du 8255 vers la CPU
36	WR/	Un signal bas sur cette entrée autorise le passage de donnée de la CPU vers 8255
6	CS/	Un signal bas sur cette entrée sélectionne le 8255 et assure son dialogue avec CPU
8 & 9	A0 & A1	Ces deux broches représentent les deux bits les plus faibles du bus d'adresses et permettent la sélection d'un port du 8255
36	RESET	Un niveau logique 1 initialise le 8255
27,28,29,30, 31,32,33,34	Bus données	Ces broches représentent le bus de données du 8255A, en commençant du poid le plus faible jusqu'au plus fort
4,3,2,1,40,39, 38,37	PORT A	Ces broches représentent le port A du bit le plus faible à celui du plus fort
18,19,20,21, 22,23,24,25	PORT B	Ces broches représentent le port B du bit le plus faible à celui du plus fort
14,15,16,17, 13,12,11,10	PORT C	Ces broches représentent le port C du bit le plus faible à celui du plus fort
7	GND	C'est la masse du 8255A
26	VCC	C'est la broche d'alimentation du 8255

2- Schéma synoptique de 8255A:



a- Amplificateur Tampon de Bus de données:

Cet amplificateur de 8 bits, bidirectionnel à trois états est utilisé pour interfacer le 8255A au bus de données du système. La donnée est transmise ou reçu par l'amplificateur durant l'exécution des instructions IN et OUT du CPU. Les mots de contrôle et l'information d'état sont aussi transférés à travers l'amplificateur tampon du Bus de données.

b- Logique de contrôle et de lecture/écriture:

La fonction de ce bloc est de contrôler tous les transferts internes et externes des données et des mots de contrôle de CPU, il crée les signaux de commande des 2 groupes de contrôle.

- $\overline{CS}$: Chip Select (sélection de boîtier):

Un niveau "bas" sur cette entrée autorise la communication entre le 8255A et la CPU.

- $\overline{RD}$: Read (lecture):

Un niveau "bas" sur cette entrée autorise le 8255A à envoyer une donnée ou une information d'état au CPU par l'intermédiaire du bus de données. Autrement dit, cette entrée permet au CPU de lire une donnée provenant du 8255A.

- $\overline{WR}$: Write (écriture):

Un niveau "bas" sur cette entrée autorise la CPU à écrire une donnée ou un mot de contrôle dans le 8255A.

- A0 et A1: Port select 0 and Port select 1
  (Sélection du circuit d'entrée 0 et 1):

Ces signaux d'entrée associés aux signaux d'Entrée : $\overline{RD}$ et $\overline{WR}$, Contrôlent la sélection d'un des 3 circuits d'accès ou du registre de contrôle.

- Reset (remise à zéro):

Un niveau "haut" sur cette entrée met à 0 tous les registres internes y compris le registre de contrôle et met tous les ports d'E/S en mode 0 et en entrée.

c- Registres de Contrôle du groupe A et B:

La configuration fonctionnelle de chaque circuit d'accès est programmée par l'envoi d'un mot de contrôle au 8255A par la CPU.

Ce mot de contrôle contient une information telle que le mode, la mise à 1 d'un bit, la mise à 0 d'un bit,... Qui initialise la configuration fonctionnelle du 8255A.

Chacun des blocs de contrôle accepte les commandes de la logique de contrôle d'écriture/lecture, reçoit les mots de contrôle du bus interne et envoie les commandes propres à ces circuits d'accès associés.

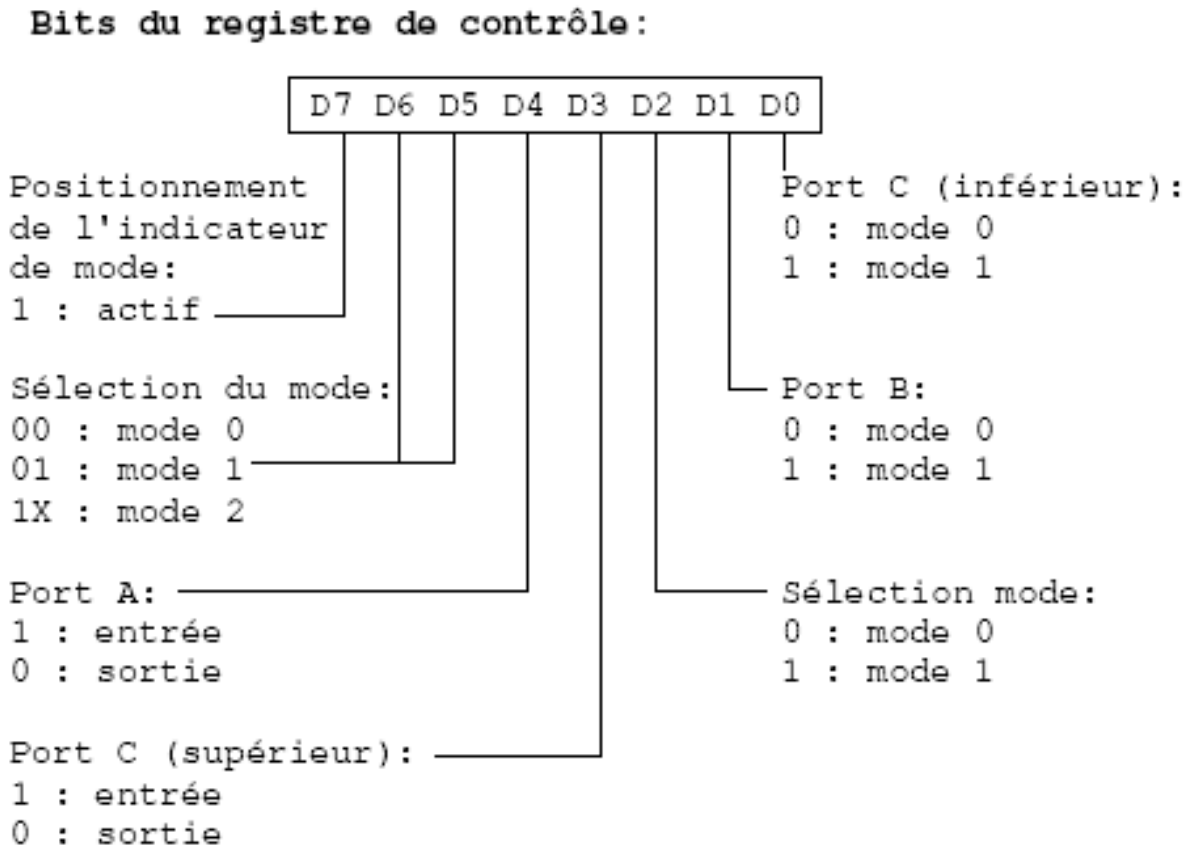
- Contrôle du groupe A: circuit d'accès A et circuit d'accès C (C4 à C7)
- Contrôle du groupe B: circuit d'accès B et circuit d'accès C (C0 à C3)

d- Les ports A, B et C:

Le 8255A possède 3 circuits d'accès de 8 bits (A,B et C). Qui peuvent être configurés sous 3 modes.

III/ Les modes de fonctionnement:

Le mode de fonctionnement est sélectionné par le bit 5 et 6 du registre de contrôle qui configure en plus les 3 ports en E ou en Sortie (bits 0,1,2,3 et 4) et d'actionner le PIA.



1- Mode 0: E/S classiques:

Ce mode permet d'utiliser 1 des 4 ports (A,B,C inf, et C sup) Comme entrée ou sortie, indépendamment les uns des autres.

Il existe 16 configurations entrée/sortie possibles qui peuvent être représentées dans le tableau suivant:

A	B	C4-C7	C0-C3	Hexa	Décimal
S	S	S	S	80	128
S	S	S	E	81	129
S	S	E	S	88	136
S	S	E	E	89	137
S	E	S	S	82	130
S	E	S	E	83	131
S	E	E	S	8A	138
S	E	E	E	8B	139
E	S	S	S	90	144
E	S	S	E	91	145
E	S	E	S	98	152
E	S	E	E	99	153
E	E	S	S	92	146
E	E	S	E	93	147
E	E	E	S	9A	154
E	E	E	E	9B	155

Direction des données: valeur du
E: entrée registre de
S: sortie contrôle

Programmation du PIA 8255 en mode 0

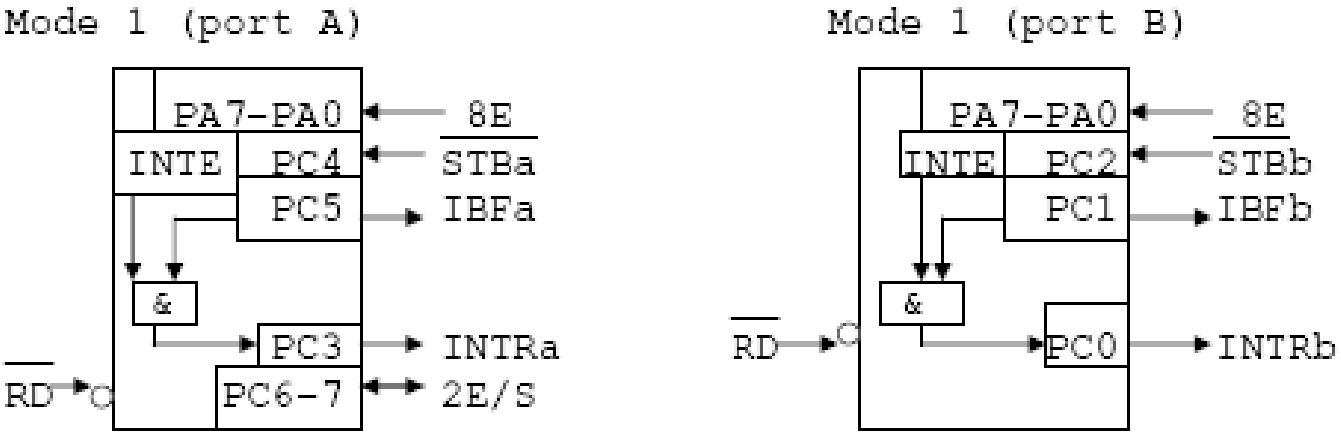
2- Mode 1:

Dans le mode 1, seuls les ports A et B sont disponibles pour le transfert parallèle des données, aussi bien les entrées que les sorties sont verrouillées. les 2x4 bits du port C servent aux signaux de dialogue (d'échange) pour les ports A et B. Deux configurations possibles l'Entrée et la Sortie:

a- Mode 1 en Entrée:

En entrée du mode 1, les ports A et B sont en entrées. Le port B utilise les bits C0,C1 et C2 pour l'asservissement et le port A utilise les bits C3,C4 et C5.

Le périphérique place une donnée de 8 bits sur A7-A0 (ou B7-B0) puis génère un STB (strobe) lequel charge la donnée dans le latch d'entrée. Ceci a pour effet l'activation du signal IBF (Input Buffer Full) à 1. Un niveau haut sur la sortie INTR peut être utilisé pour interrompre le Microprocesseur lors d'une demande du périphérique. L'INTR est activé quand STB=0,IBF=1 et INTE=1 et mise à 0 par le front descendant de RD.

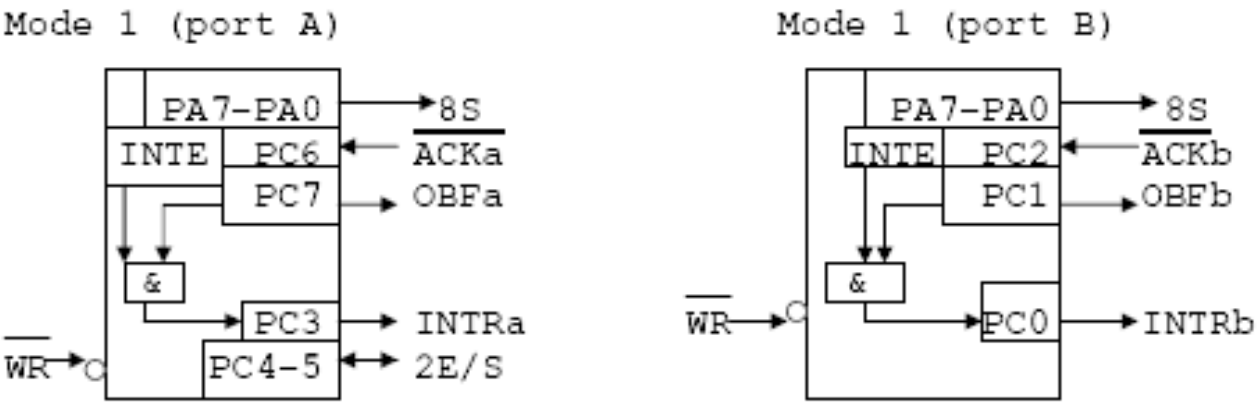


b- Mode 1 en Sortie:

Pour la sortie en mode 1, le microprocesseur écrit la donnée dans le port A (ou B). Le front montant de WR fait cesser l'interruption du microprocesseur INTR=0 et met OBF=0 pour signifier au périphérique que son buffer de sortie est plain et qu'il peut donc venir chercher la donnée.

ACK=0 est une réponse qui indique que le périphérique a reçu la donnée provenant de la CPU et qui a pour effet de faire cesser OBF=1, de nouveau ACK=1,OBF=1 et INTE=1.

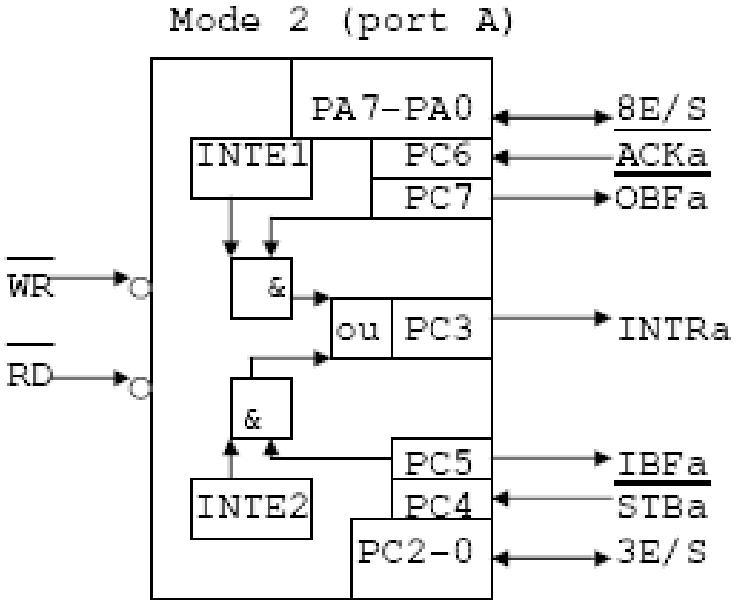
Fait interrompre le microprocesseur pour qu'il puisse écrire la donnée suivante.



3- Mode 2:

Le mode 2 permet d'avoir un bus bidirectionnel sur le port A. Cinq bits du port C sont utilisés pour le status et le contrôle du port A.

- Signaux de contrôle de bus bidirectionnel d'E/S:
  INTR=1 pour interrompre le microprocesseur lorsque le port a accepté ou transmis une donnée.
- Signaux de contrôle en Ecriture:
  OBF=0 indique que le microprocesseur sort une donnée par le port A.
- Signaux de contrôle en Lecture:
  STB=0 Charge la donnée dans le latch d'entrée.
  IBF=1 indique que la donnée a été chargée dans le latch d'entrée.



Annexe 1

Liste des instructions du CPU Intel 8086

Nom (mnémonique)		Action
AAA	ASCII Adjust for Addition	opération complexe, voir littérature
AAD	ASCII Adjust for Division	opération complexe, voir littérature
AAM	ASCII Adjust for Multi-plic.	opération complexe, voir littérature
AAS	ASCII Adjust for Subtr..	opération complexe, voir littérature
ADC	Add with Carry	*op1* + *op2* (+1) → *op1* (+1 si *CF* = 1)
ADD	Add	*op1* + *op2* → *op1*
AND		*op1 AND op2* → *op1*
CALL		appel de procédure
CBW	Convert Byte to Word	AL → AX (extension bit de signe)
CLC	Clear Carry flag	0 → *CF* ('Carry' bit, 'flag-register')
CLD	Clear Direction flag	0 → *DF* ('Direction' bit, 'flag-reg.')
CLI	Clear Interrupt flag	0 → *IF* ('Interrupt' bit, 'flag-register')
CMC	Complement Carry flag	*NOT CF* → *CF* ('Carry' bit, 'flag-reg.')
CMP	Compare	*op1* – *op2*
CMPB	Compare Byte	*zone-mémoire* – *zone-mémoire* (byte)
CMPW	Compare Word	*zone-mémoire* – *zone-mémoire* (mot)
CWD	Convert Word to Double	*AX* → *DX:AX* (extension bit de signe)
DAA	Decimal Adjust for Addi-tion	opération complexe, voir littérature
DAS	Decimal Adjust for Subtr.	opération complexe, voir littérature
DEC	Decrement	*op1* – 1 → *op1*
DIV	Divide	*DX:AX* / *op1* → *AX*, reste → *DX* *AX* / *op1* → *AL*, reste → *AH*
ESC	Escape	*op1* → bus (pas d'autre action CPU)
HLT	Halt	arrêter le CPU
IDIV	Integer Divide	*DX:AX* / *op1* → *AX*, reste → *DX* *AX* / *op1* → *AL*, reste → *AH* (op. abs.)
IMUL	Integer Multiply	*op1* * *AX* → *DX:AX* *op1* * *AL* → *AX* (op. abs.)
IN	Input	entrée/sortie → *AL*
INC	Increment	*op1* + 1 → *op1*
INT	Interrrupt	interruption par vecteur numéro *op1*
INTO	Interrupt Overflow	interr. par vecteur numéro 4 (si *OF* = 1)

Nom (mnémonique)		Action	
INW	Input Word	entrée/sortie → *AX*	
IRET	Interrupt Return	retour du traitement d'interruption	
JA	Jump if Above = JNBE	branchement cond. (*op2* > *op1*)	opérandes absolues
JAE	Jump if Above or Eq. = JNB	branchement cond. (*op2* Š≥ *op1*)	
JB	Jump if Below = JNAE	branchement cond. (*op2* < *op1*)	
JBE	Jump if Below or Eq. = JNA	branchement cond. (*op2* ≤ *op1*)	
JCXZ	Jump if CX = 0	branchement si *CX* = 0	
JE	Jump if Equal = JZ	branchement cond. (*op2* = *op1*)	
JG	Jump if Greater = JNLE	branchement cond. (*op2* > *op1*)	
JGE	Jump if Greater or Eq. = JNL	branchement cond. (*op2* ≥ *op1*)	
JL	Jump if Less = JNGE	branchement cond. (*op2* < *op1*)	
JLE	Jump if Less or Equal = JNG	branchement cond. (*op2* ≤ *op1*)	
JMP	Jump	branchement inconditionnel	
JNA	Jump if Not Above = JBE	branchement cond. (*op2* ≤ *op1*)	opérandes absolues
JNAE	Jump if Not Above or Eq.= JB	branchement cond. (*op2* < *op1*)	
JNB	Jump if Not Below = JAE	branchement cond. (*op2* ≥ *op1*)	
JNBE	Jump if Not Below or Eq.= JA	branchement cond. (*op2* > *op1*)	
JNE	Jump if Not Equal = JNZ	branchement cond. (*op2* ≠ *op1*)	
JNG	Jump if Not Greater = JLE	branchement cond. (*op2* ≤ *op1*)	
JNGE	Jump if Not Greater or Eq. =JL	branchement cond. (*op2* < *op1*)	
JNL	Jump if Not Less = JGE	branchement cond. (*op2* ≥ *op1*)	
JNLE	Jump if Not Less or Eq. = JG	branchement cond. (*op2* > *op1*)	
JNO	Jump if Not Overflow	branchement cond. (si pas 'overflow')	
JNP	Jump if Not Parity = JPO	branchement cond. (si parité impaire)	
JNS	Jump if Not Sign	branchement cond. (si valeur positive)	
JNZ	Jump if Not Zero = JNE	branchement cond. (si résultat¼ ¼≠ 0)	
JO	Jump if Overflow	branchement cond. (si 'overflow')	
JP	Jump if Parity = JPE	branchement cond. (si parité paire)	

Nom (mnémonique)			Action
JPE	Jump if Parity Even	= JP	branchement cond. (si parité paire)
JPO	Jump if Parity Odd	= JNP	branchement cond. (si parité impaire)
JS	Jump if Sign		branchement cond. (si valeur négative)
JZ	Jump if Zero	= JE	branchement cond. (si résultat¼ = 0)
LAHF	Load AH with Flags		bits arithmétiques du 'flag-reg.' → AH
LDS	Load pointer to DS		adresse de *op2* → *DS:op1*
LEA	Load Effective Addr.		adresse de *op2* → *op1*
LES	Load pointer to ES		adresse de *op2* → *ES:op1*
LOCK			réservation du bus pour > 1 cycle
LODB	Load Byte		*zone-mémoire* → *AL*
LODW	Load Word		*zone-mémoire* → *AX*
LOOP			branchement si *CX* = 0
LOOPE	Loop while Equal	= LOOPZ	branchement si *CX* = 0 et *ZF* = 1
LOOPNE	Loop while Not Eq.	= LOOPNZ	branchement si *CX* = 0 et *ZF* = 0
LOOPNZ	Loop while Not Zero	= LOOPNE	branchement si *CX* = 0 et *ZF* = 1
LOOPZ	Loop while Zero	= LOOPE	branchement si *CX* = 0 et *ZF* = 1
MOV	Move		*op2* → *op1*
MOVB	Move Byte		*zone-mémoire* → *zone-mémoire*
MOVW	Move Word		*zone-mémoire* → *zone-mémoire*
MUL	Multiply		*op1* * *AX* → *DX:AX* *op1* * *AL* → *AX*
NEG	Negate		0 – *op1* → *op1*
NOT			*NOT op1* → *op1*
OR			*op1 OR op2* → *op1*
OUT	Output		*AL* → entrée/sortie
OUTW	Output Word		*AX* → entrée/sortie
POP			des-empiler → *op1*
POPF	Pop Flags		des-empiler → 'flag-register'
PUSH			*op1* → empiler
PUSHF	Push Flags		'flag-register' → empiler
RCL	Rotate Left with Carry		décalage circulaire gauche par *CF*
RCR	Rotate Right with Carry		décalage circulaire droite par *CF*
REP	Repeat		pré-fixe: repétition sur zone-mémoire
RET	Return		retour d'une procédure

1ère action: *CX* – 1 → *CX*

Nom (mnémonique)			Action
ROL	Rotate Left		décalage circulaire gauche
ROR	Rotate Right		décalage circulaire droite
SAHF	Store AH to Flags		*AH* → bits arithm. du 'flag-register'
SAL	Shift Arithm. Left	= SHL	décalage à gauche (0-remplissage)
SAR	Shift Arithmetic Right		décalage à droite, extension du signe
SBB	Subtract with Borrow		*op1* – *op2* (–1) → *op1* (–1 si *CF* = 1)
SCAB	Scan Byte		*AL* – *zone-mémoire*
SCAW	Scan Word		*AX* – *zone-mémoire*
SHL	Shift Logical Left	= SAL	décalage à gauche (0-remplissage)
SHR	Shift Logical Right		décalage à droite (0-remplissage)
STC	Set Carry flag		1 → *CF* ('Carry' bit, 'flag-register')
STD	Set Direction flag		1 → *DF* ('Direction' bit 'flag-reg.')
STI	Set Interrupt flag		1 → *IF* ('Interrupt' bit, 'flag-register')
STOB	Store Byte		*AL* → *zone-mémoire*
STOW	Store Word		*AX* → *zone-mémoire*
SUB	Subtract		*op1* – *op2* → *op1*
TEST			*op1 AND op2*
WAIT			le CPU entre dans une boucle d'attente
XCHG	Exchange		*op1* ⇔ *op2*

Nom (mnémonique)		Action
XLAT	Translate	conversion de code par table de corresp.
XOR	Exclusive Or	*op1 XOR op2*

Annexe 2

Liste des interruptions BIOS

Les interruptions internes au processeur :

- `INT 00h` : Division par zéro. Le gestionnaire est appelé lorsqu'un essai de division par zéro est tenté. Le BIOS de l'IBM PC se contente de faire apparaître un message et suspend le système.
- `INT 01h` : Étape par étape. Cette interruption est utilisée par *debug* et d'autres débogueurs pour exécuter un programme ligne par ligne.
- `INT 03h` : Point d'arrêt. Cette interruption est utilisée par *debug* et d'autres débogueurs pour arrêter l'exécution d'un programme.
- `INT 04h` : Dépassement de capacité.

Les interruptions externes au processeur :

- `INT 05h` : Impression de l'écran.
- `INT 08h` : Temporisateur du système.
- `INT 09h` : Interruption du clavier avec de nombreuses fonctions.
- `INT 0Bh` : contrôle du premier port série COM1.
- `INT 0Ch` : contrôle du deuxième port série COM2.
- `INT 0Dh` : contrôle du deuxième port parallèle LPT2.
- `INT 0Eh` : contrôle des lecteurs de disquette. Signale une activité d'un des lecteurs, telle qu'une opération d'entrée-sortie.
- `INT 0Fh` : contrôle du premier port parallèle LPT1.
- `INT 10h` : interruption avec de nombreuses fonctions concernant le moniteur.
- `INT 11h` : détermination de l'équipement, utilisé en particulier lors du démarrage de l'ordinateur.
- `INT 12h` : détermination de la taille mémoire (jusqu'à 640 KiO, taille maximale gérée par MS-DOS), en particulier lors du démarrage de l'ordinateur.
- `INT 13h` : interruption avec de nombreuses fonctions concernant les entrées-sorties sur les disques.
- `INT 14h` : contrôle des communications séries avec de nombreuses fonctions.
- `INT 15h` : contrôle des services du système avec de nombreuses fonctions.
- `INT 16h` : entrée au clavier, avec de nombreuses fonctions.
- `INT 17h` : sortie sur l'imprimante, avec de nombreuses fonctions.
- `INT 18h` : entrée en BASIC situé en mémoire ROM. Cette interruption était en particulier appelée lorsqu'aucun disque système n'était trouvé. Cette interruption est inactive en général de nos jours.
- `INT 19h` : démarrage de l'amorce du système. Si un disque système est présent, le système est chargé, sinon il est fait appel à l'interruption `int 18h`.
- `INT 1Ah` : lit le temps ou le met à jour, avec un certain nombre de fonctions.
- `INT 1Bh` : prend le contrôle lors d'une interruption (*CTRL-break*) effectuée au clavier.

Annexe 3

Liste des interruptions MSDOS

Liste de fonctions de l'interruption ms-dos int 21H

AH=01H	LECTURE D'UN CARACTERE AU CLAVIER AVEC ECHO
AH=02H	ECRITURE D'UN CARACTERE A L'ECRAN
AH=03H	LECTURE D'UN CARACTERE SUR L'ENTREE AUXILIAIRE
AH=04H	ECRITURE D'UN CARACTERE SUR LA SORTIE AUXILIAIRE
AH=05H	ENVOI D'UN CARACTERE A L'IMPRIMENTE
AH=06H	LECTURE/ECRITURE DIRECTE
AH=07H	ENTREE DIRECTE D'UN CARACTERE SANS ECHO AU CLAVIER
AH=08H	LECTURE D'UN CARACTERE SANS ECHO AU CLAVIER
AH=09H	AFFICHAGE D'UNE CHAINE DE CARACTERS A L'ECRAN
AH=0AH	LECTURE D'UNE CHAINE DE CARACTERS AU CLAVIER
AH=0BH	TEST DU CLAVIER
AH=0CH	VIDE LE BUFFER STANDARD D'ENTREE ET LIT UN CARACTERE
AH=19H	DETERMINE LE DISQUE COURANT
AH=25H	MODIFIE UN VECTEUR D'INTERRUPTION
AH=2AH	LECTURE DE LA DATE SYSTEME
AH=30H	LECTURE DU NUMERO DE VERSION DU DOS
AH=31H	FIN DE PROGRAMME ET RESTE RESIDENT
AH=33H	CONTROLE DU BREAK
AH=35H	LECTURE D'UN VECTEUR D'INTERRUPTION
AH=36H	INFORMATION SUR LE DRIVE ET LA PALACE LIBRE RESTANTE
AH=62H	OBTENTION DE L'ADRESSE DU PSP