الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Ziane Achour de Djelfa — جامعة الجلفة
Faculté des Sciences et de la Technologie — كلية العلوم و التكنولوجيا
Département de Génie Electrique — قسم الهندسة الكهربائية

Semestre: 1

Matières: Microprocesseurs et Microcontrôleurs

1er année Master ELT(Electrotechnique)

Chapitre 2

La programmation en assembleur

Contenu de la matière :

Généralités, Le jeu d'instructions, Méthode de programmation

Réalisé et présenté par :

Dr. Obeidi Thameur

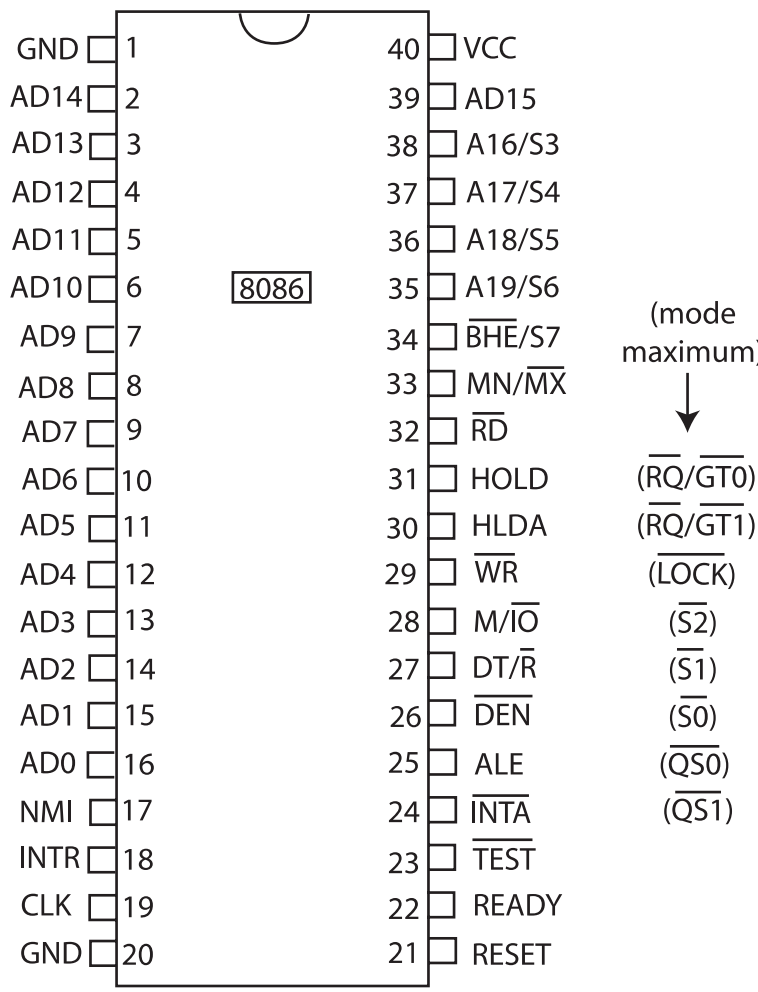
Année Universitaire : 2023 / 2024

Chapitre 2
La programmation en assembleur

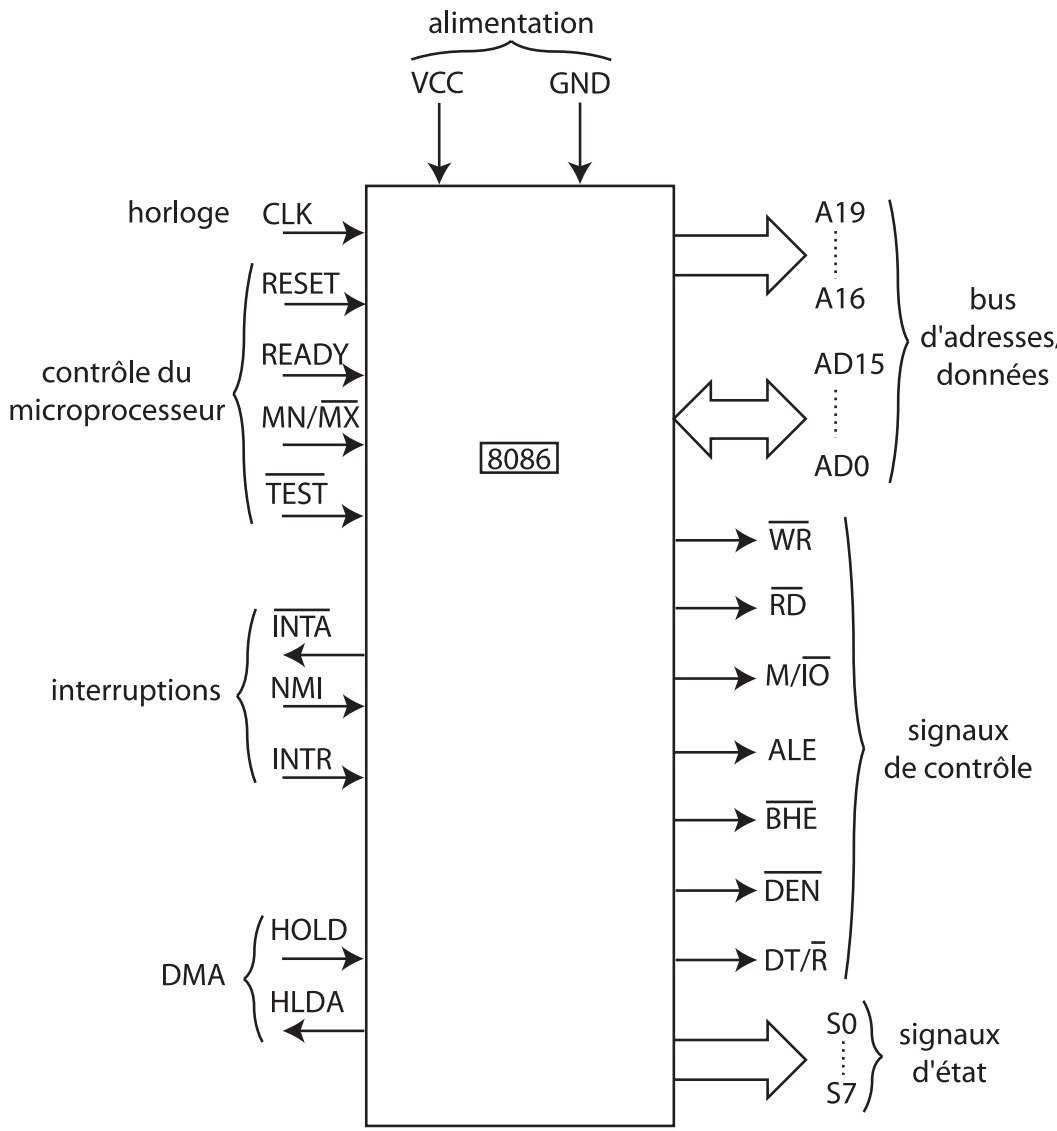
Le microprocesseur Intel 8086

1. Description physique du 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 :

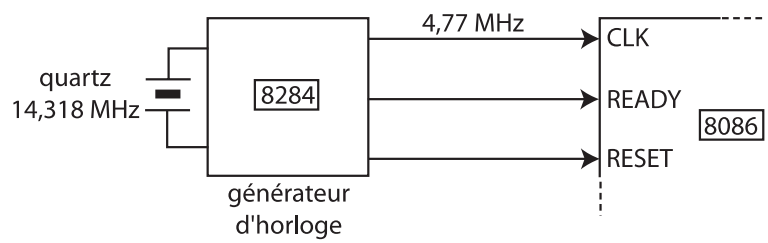


2. Schéma fonctionnel du 8086



3. Description et utilisation des signaux du 8086

CLK : entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un **générateur d'horloge** : le 8284.



RESET : entrée de remise à zéro du microprocesseur. Lorsque cette entrée est mise à l'état haut pendant au moins 4 périodes d'horloge, le microprocesseur est réinitialisé : il va exécuter l'instruction se trouvant à l'adresse FFFF0H (adresse de bootstrap). Le signal de RESET est fourni par le générateur d'horloge.

READY : entrée de synchronisation avec la mémoire. Ce signal provient également du générateur d'horloge.

$\overline{\textbf{TEST}}$: entrée de mise en attente du microprocesseur d'un événement extérieur.

MN/$\overline{\textbf{MX}}$: entrée de choix du mode de fonctionnement du microprocesseur :

- mode minimum (MN/$\overline{\text{MX}} = 1$) : le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande ($\overline{\text{RD}}$, $\overline{\text{WR}}$, ...) ;
- mode maximum (MN/$\overline{\text{MX}} = 0$) : ces signaux de commande sont produits par un **contrôleur de bus**, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs.

NMI et **INTR** : entrées de demande d'interruption. INTR : interruption normale, NMI (Non Maskable Interrupt) : interruption prioritaire.

$\overline{\textbf{INTA}}$: Interrupt Acknowledge, indique que le microprocesseur accepte l'interruption.

HOLD et **HLDA** : signaux de demande d'accord d'accès direct à la mémoire (DMA).

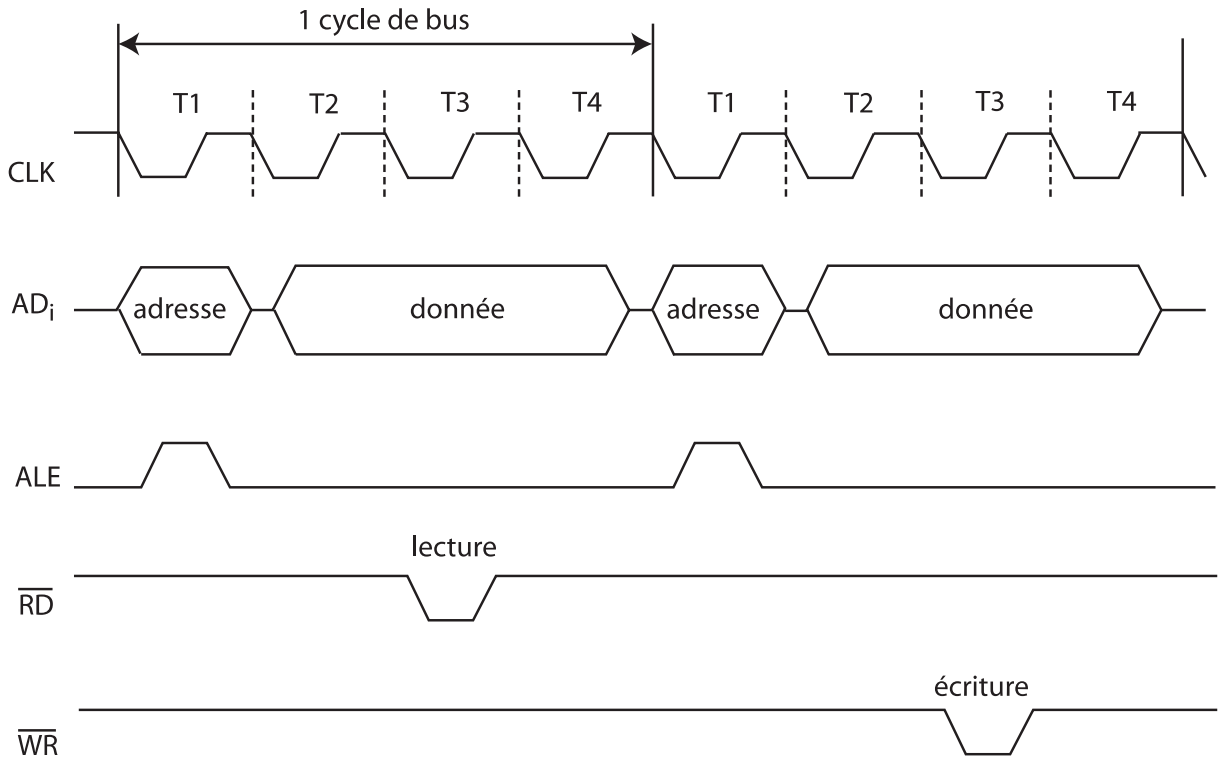
S0 à **S7** : signaux d'état indiquant le type d'opération en cours sur le bus.

A16/S3 à **A19/S6** : 4 bits de poids fort du bus d'adresses, **multiplexés** avec 4 bits d'état.

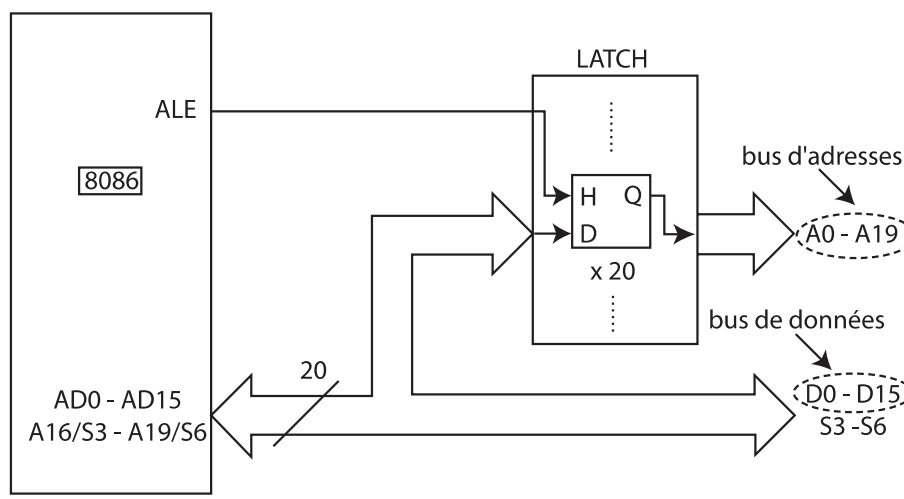
AD0 à **AD15** : 16 bits de poids faible du bus d'adresses, **multiplexés** avec 16 bits de données. Le bus A/D est multiplexé (multiplexage temporel) d'où la nécessité d'un **démultiplexage** pour obtenir séparément les bus d'adresses et de données :

- 16 bits de données (microprocesseur 16 bits) ;
- 20 bits d'adresses, d'où $2^{20} = 1$ Mo d'espace mémoire adressable par le 8086.

Chronogramme du bus A/D :



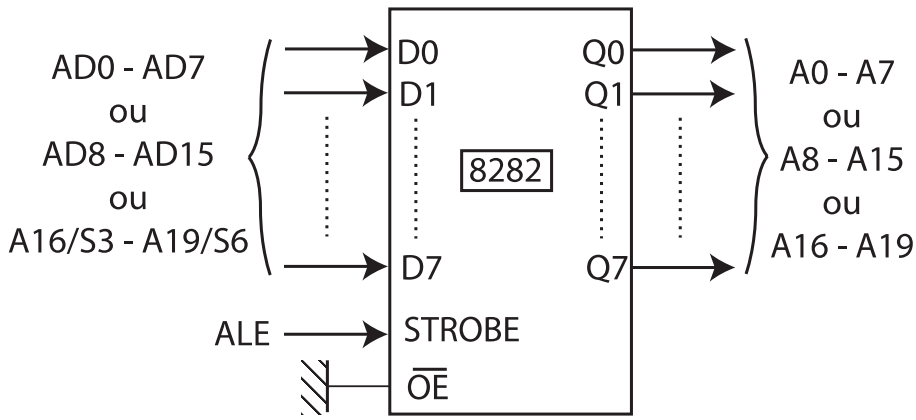
Le démultiplexage des signaux AD0 à AD15 (ou A16/S3 à A19/S6) se fait en mémorisant l'adresse lorsque celle-ci est présente sur le bus A/D, à l'aide d'un **verrou** (latch), ensemble de bascules D. La commande de mémorisation de l'adresse est générée par le microprocesseur : c'est le signal **ALE**, Address Latch Enable.
Circuit de démultiplexage A/D :



Fonctionnement :

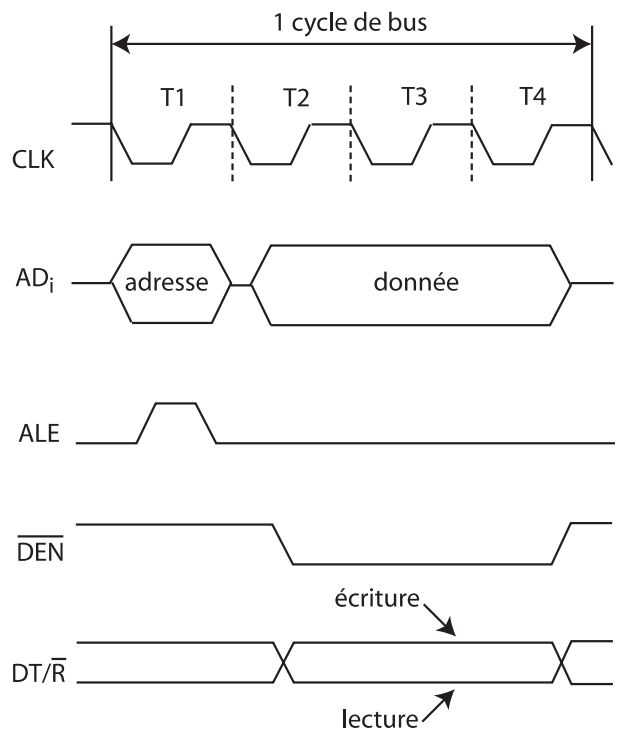
- si ALE = 1, le verrou est transparent (Q = D) ;
- si ALE = 0, mémorisation de la dernière valeur de D sur les sorties Q ;
- les signaux de lecture ($\overline{RD}$) ou d'écriture ($\overline{WR}$) ne sont générés par le microprocesseur que lorsque les données sont présentes sur le bus A/D.

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



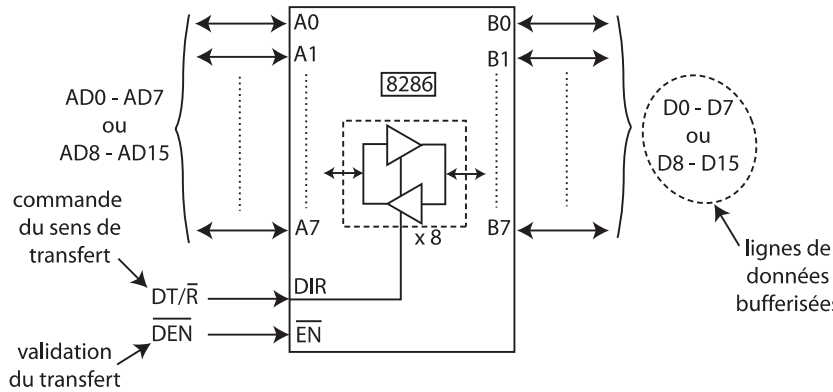
$\overline{\text{RD}}$: Read, signal de lecture d'une donnée.
$\overline{\text{WR}}$: Write, signal d'écriture d'une donnée.
M/$\overline{\text{IO}}$: Memory/Input-Output, indique si le 8086 adresse la mémoire (M/$\overline{\text{IO}} = 1$) ou les entrées/sorties (M/$\overline{\text{IO}} = 0$).
$\overline{\text{DEN}}$: Data Enable, indique que des données sont en train de circuler sur le bus A/D (équivalent de ALE pour les données).
DT/$\overline{\text{R}}$: Data Transmit/Receive, indique le sens de transfert des données :

- DT/$\overline{\text{R}} = 1$: données émises par le microprocesseur (écriture) ;
- DT/$\overline{\text{R}} = 0$: données reçues par le microprocesseur (lecture).



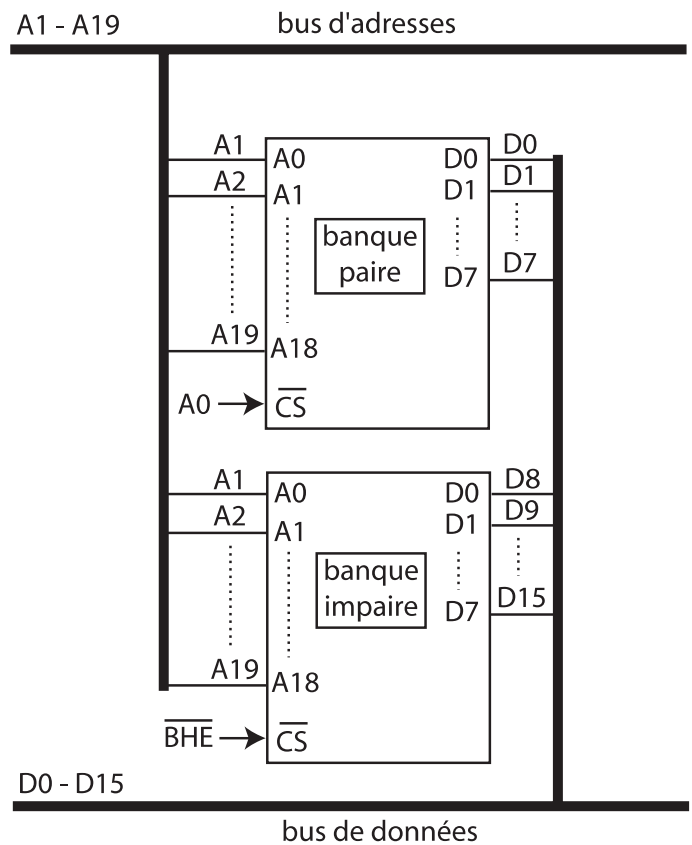
Les signaux $\overline{\text{DEN}}$ et DT/$\overline{\text{R}}$ sont utilisés pour la commande de **tampons de bus** (buffers) permettant d'amplifier le courant fourni par le microprocesseur sur le bus de données.

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



$\overline{\text{BHE}}$: Bus High Enable, signal de lecture de l'octet de poids fort du bus de données. Le 8086 possède un bus d'adresses sur 20 bits, d'où la capacité d'adressage de 1 Mo ou 512 Kmots de 16 bits (bus de données sur 16 bits).
Le méga-octet adressable est divisé en deux **banques** de 512 Ko chacune : la banque **inférieure** (ou **paire**) et la banque **supérieure** (ou **impaire**). Ces deux banques sont sélectionnées par :

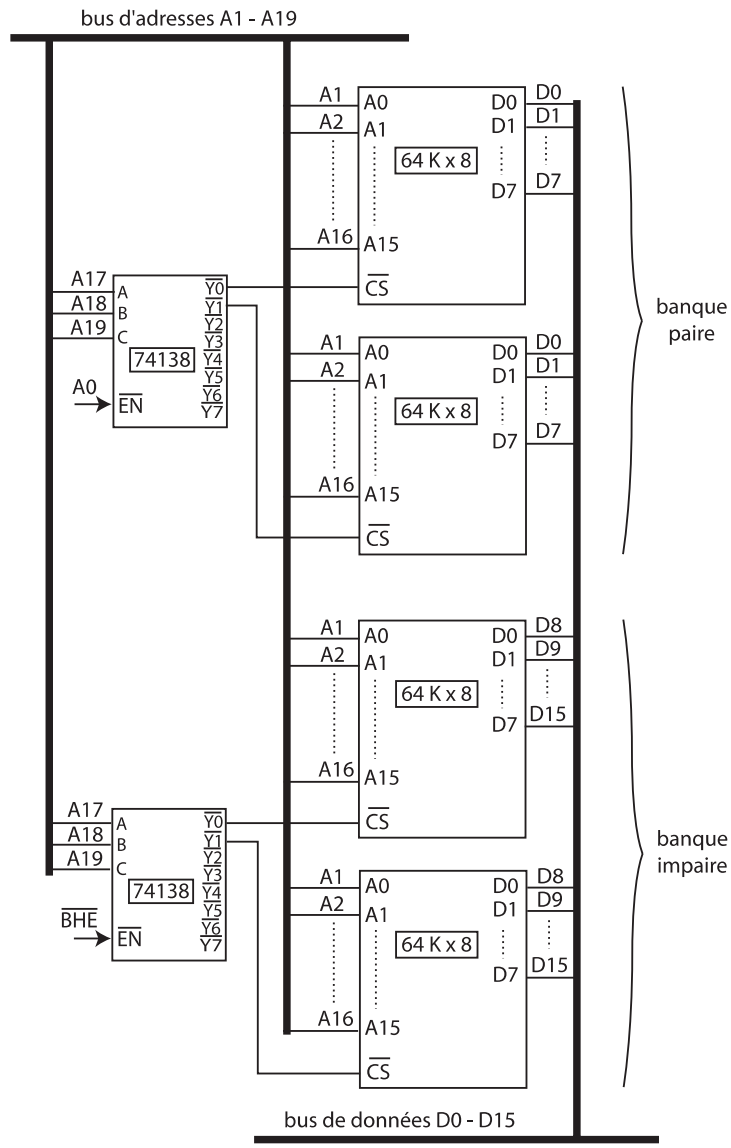
- A0 pour la banque paire qui contient les octets de poids faible ;
- $\overline{\text{BHE}}$ pour la banque impaire qui contient les octets de poids fort.



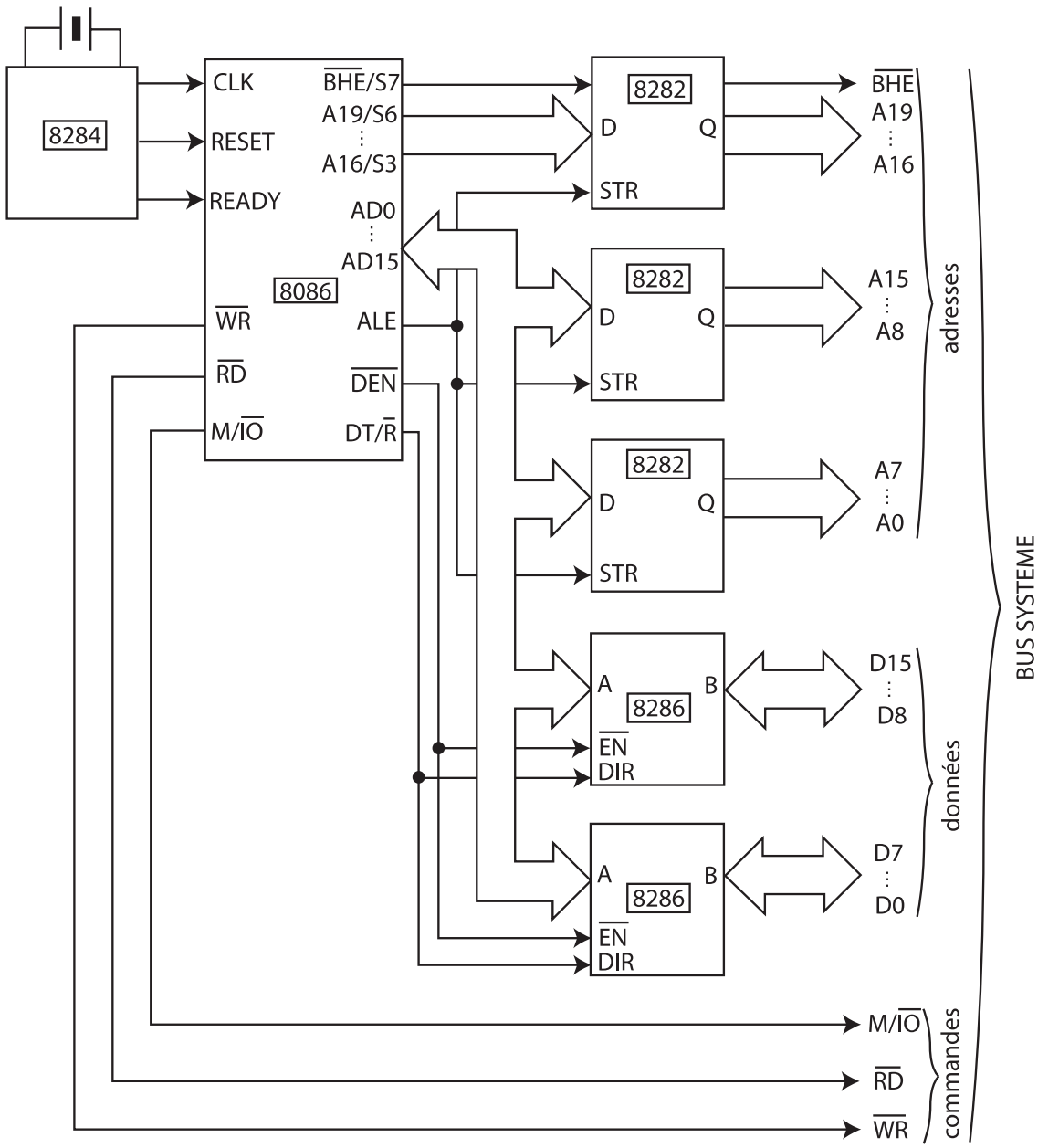
Seuls les bits A1 à A19 servent à désigner une case mémoire dans chaque banque de 512 Ko. Le microprocesseur peut ainsi lire et écrire des données sur 8 bits ou sur 16 bits :

$\overline{BHE}$	A0	octets transférés
0	0	les deux octets (mot complet)
0	1	octet fort (adresse impaire)
1	0	octet faible (adresse paire)
1	1	aucun octet

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.
Réalisation des deux banques avec plusieurs boîtiers mémoire :



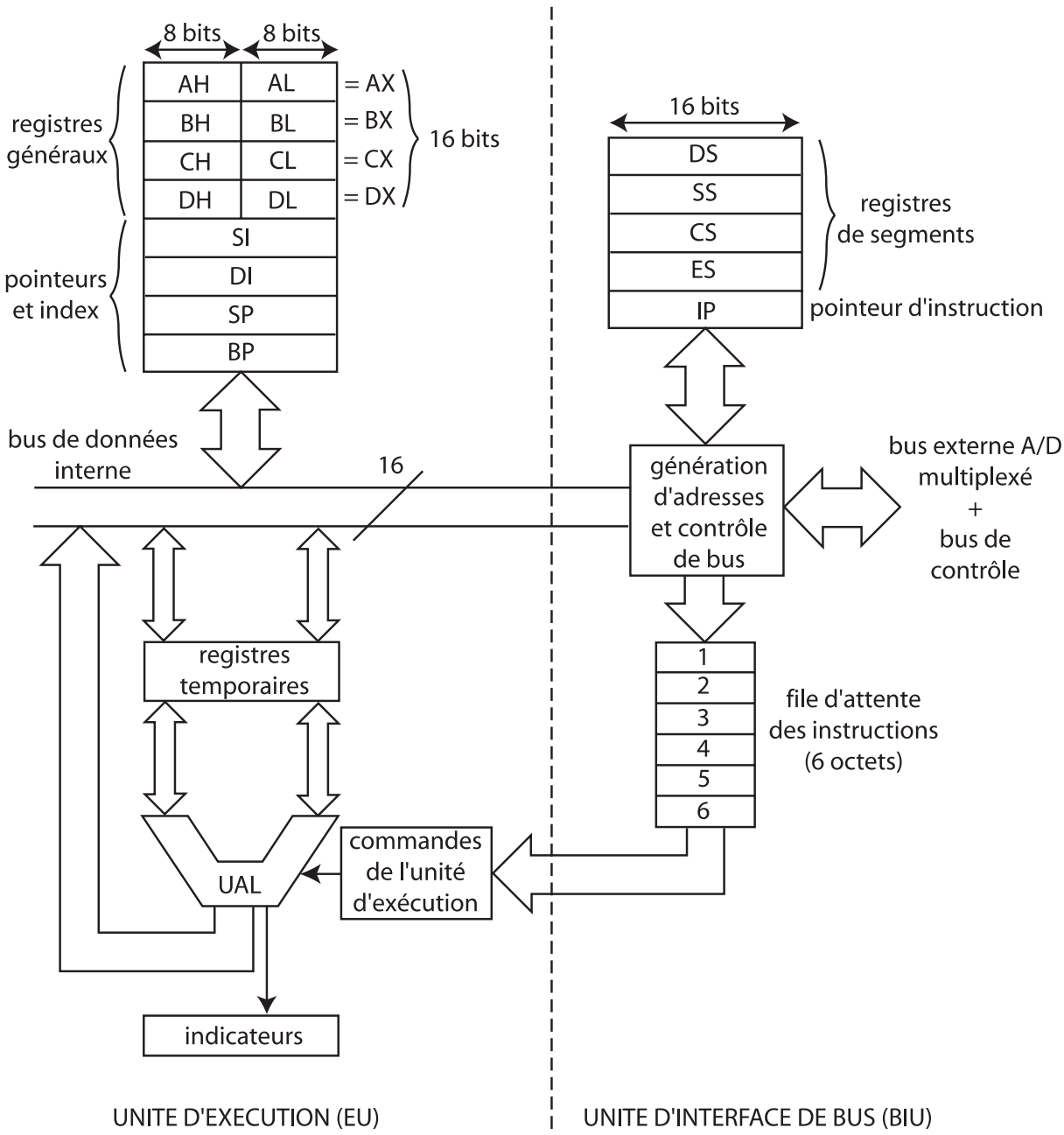
Création du bus système du 8086 :



4. Organisation interne du 8086

Le 8086 est constitué de deux unités fonctionnant en parallèle :

- l'**unité d'exécution** (EU : Execution Unit) ;
- l'**unité d'interface de bus** (BIU : Bus Interface Unit).



Rôle des deux unités :

- l'unité d'interface de bus (BIU) recherche les instructions en mémoire et les range dans une **file d'attente** ;
- l'unité d'exécution (EU) exécute les instructions contenues dans la file d'attente.

Les deux unités fonctionnent simultanément, d'où une accélération du processus d'exécution d'un programme (fonctionnement selon le principe du **pipe-line**).
Le microprocesseur 8086 contient 14 registres répartis en 4 groupes :

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

AX = (**AH**,**AL**) ;
BX = (**BH**,**BL**) ;
CX = (**CH**,**CL**) ;
DX = (**DH**,**DL**).

Ils peuvent être également considérés comme 8 registres sur 8 bits. Ils servent à contenir temporairement des données. Ce sont des registres généraux mais ils peuvent être utilisés pour des opérations particulières. Exemple : AX = accumulateur, CX = compteur.

- **Registres de pointeurs et d'index** : 4 registres sur 16 bits.

  Pointeurs :

  SP : Stack Pointer, pointeur de pile (la pile est une zone de sauvegarde de données en cours d'exécution d'un programme) ;

  BP : Base Pointer, pointeur de base, utilisé pour adresser des données sur la pile.

  Index :

  SI : Source Index ;

  DI : Destination Index.
  Ils sont utilisés pour les transferts de chaînes d'octets entre deux zones mémoire.

  Les pointeurs et les index contiennent des adresses de cases mémoire.

- **Pointeur d'instruction et indicateurs (flags)** : 2 registres sur 16 bits.

  Pointeur d'instruction : **IP**, contient l'adresse de la prochaine instruction à exécuter.

  Flags :

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

  CF : indicateur de retenue (carry) ;

  PF : indicateur de parité ;

  AF : indicateur de retenue auxiliaire ;

  ZF : indicateur de zéro ;

  SF : indicateur de signe ;

  TF : indicateur d'exécution pas à pas (trap) ;

  IF : indicateur d'autorisation d'interruption ;

  DF : indicateur de décrémentation ;

  OF : indicateur de dépassement (overflow).

- **Registres de segments** : 4 registres sur 16 bits.

  CS : Code Segment, registre de segment de code ;

DS : Data Segment, registre de segment de données ;

SS : Stack Segment, registre de segment de pile ;

ES : Extra Segment, registre de segment supplémentaire pour les données ;

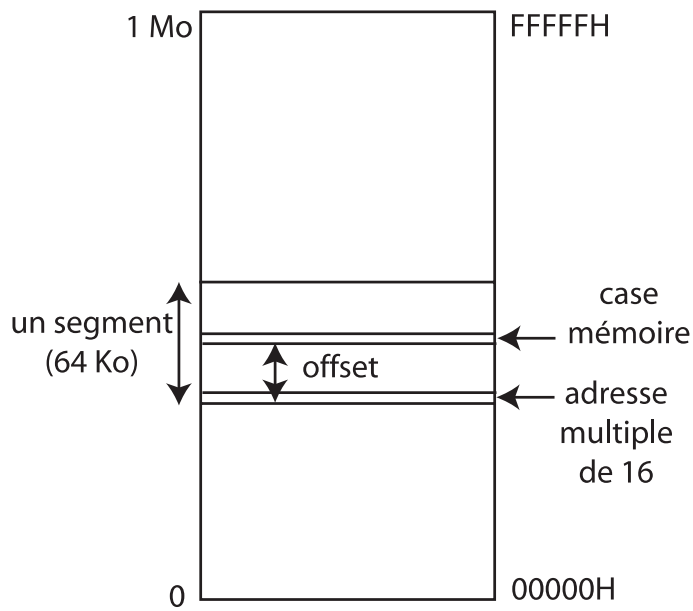
Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.

5. Gestion de la mémoire par le 8086

L'espace mémoire adressable par le 8086 est de 2^{20} = 1 048 576 octets = 1 Mo (20 bits d'adresses). Cet espace est divisé en **segments**. Un segment est une zone mémoire de 64 Ko (65 536 octets) définie par son adresse de départ qui doit être un multiple de 16. Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bits de poids fort, les 4 bits de poids faible étant implicitement à 0.
Pour désigner une case mémoire parmi les 2^{16} = 65 536 contenues dans un segment, il suffit d'une valeur sur 16 bits.
Ainsi, une case mémoire est repérée par le 8086 au moyen de deux quantités sur 16 bits :

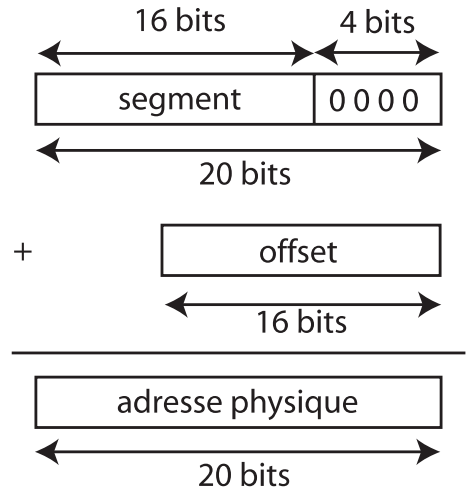
- l'adresse d'un segment ;
- un déplacement ou **offset** (appelé aussi **adresse effective**) dans ce segment.

Cette méthode de gestion de la mémoire est appelée **segmentation de la mémoire**.



La donnée d'un couple (segment,offset) définit une **adresse logique**, notée sous la forme **segment : offset**.
L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée **adresse physique** car elle correspond à la valeur envoyée réellement sur le bus d'adresses A0 - A19.

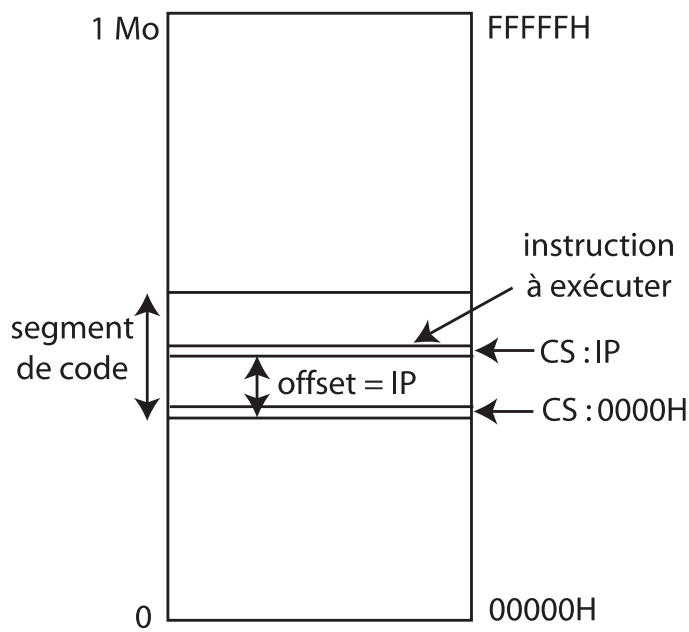
Correspondance entre adresse logique et adresse physique :



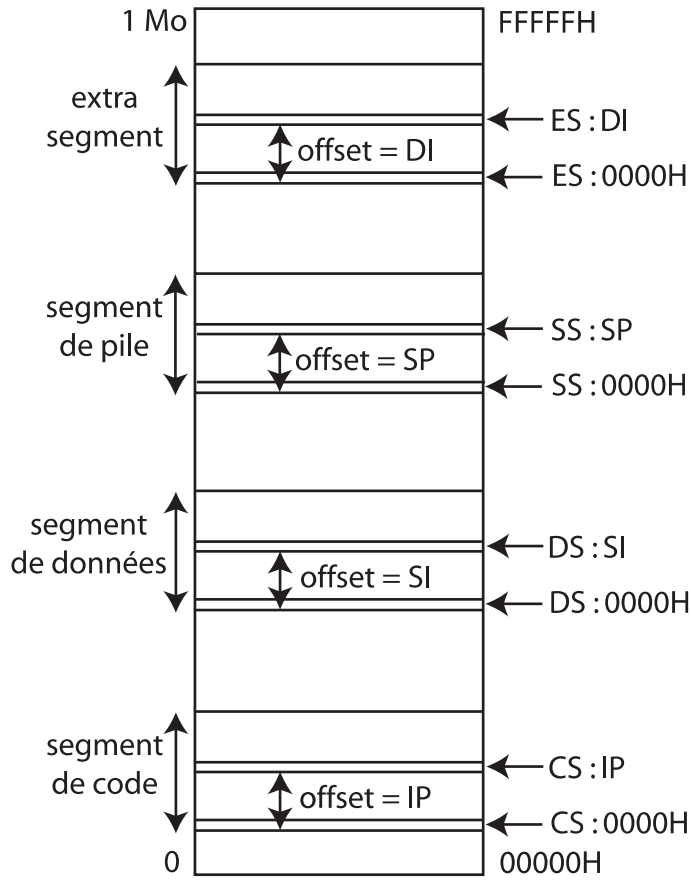
Ainsi, l'adresse physique se calcule par l'expression :

$$\textbf{adresse physique} = \textbf{16} \times \textbf{segment} + \textbf{offset}$$

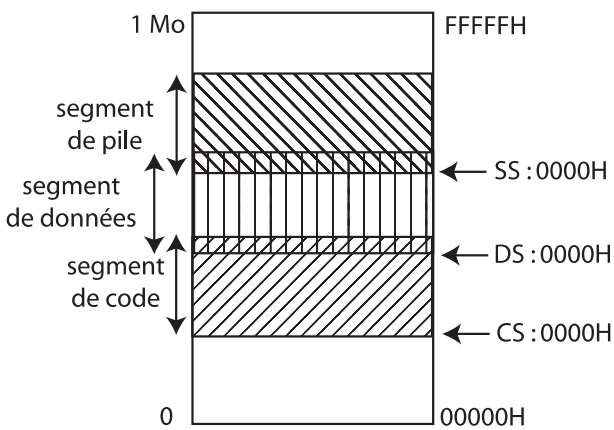
car le fait d'injecter 4 zéros en poids faible du segment revient à effectuer un décalage de 4 positions vers la gauche, c'est à dire une multiplication par $2^4 = 16$.
A un instant donné, le 8086 a accès à 4 segments dont les adresses se trouvent dans les registres de segment CS, DS, SS et ES. Le segment de code contient les instructions du programme, le segment de données contient les données manipulées par le programme, le segment de pile contient la pile de sauvegarde et le segment supplémentaire peut aussi contenir des données.
Le registre CS est associé au pointeur d'instruction IP, ainsi la prochaine instruction à exécuter se trouve à l'adresse logique CS : IP.

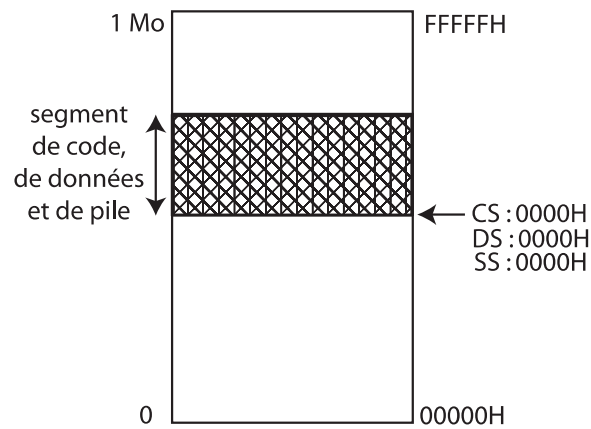


De même, les registres de segments DS et ES peuvent être associés à un registre d'index. Exemple : DS : SI, ES : DI. Le registre de segment de pile peut être associé aux registres de pointeurs : SS : SP ou SS : BP.
Mémoire accessible par le 8086 à un instant donné :



Remarque : les segments ne sont pas nécessairement distincts les uns des autres, ils peuvent se chevaucher ou se recouvrir complètement.





Le nombre de segments utilisé définit le **modèle mémoire** du programme.

Contenu des registres après un RESET du microprocesseur :

IP = 0000H

CS = FFFFH

DS = 0000H

ES = 0000H

SS = 0000H

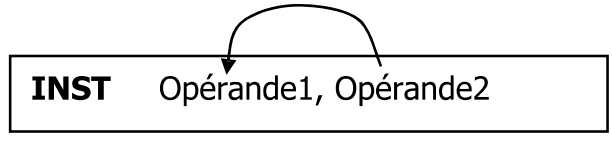
Puisque CS contient la valeur FFFFH et IP la valeur 0000H, la première instruction exécutée par le 8086 se trouve donc à l'adresse logique FFFFH : 0000H, correspondant à l'adresse physique FFFF0H (bootstrap). Cette instruction est généralement un saut vers le programme principal qui initialise ensuite les autres registres de segment.

6. MODES D'ADRESSAGE

Dans la suite on utilisera les abréviations suivantes :

INST : instruction,
R : Registre quelconque,
Rseg : Registre Segment
Roff : Registre d'offset
Adr : Adresse
[adr]: contenu Mémoire
Off : Offset de l'adresse
Im : donnée (constante)
Dep : déplacement (constante)
Op : Opérande
Os : Opérande source
Od : opérande destination

La structure la plus générale d'une instruction est la suivante :



L'opération est réalisée entre les 2 opérandes et le résultat est toujours récupéré dans l'opérande de gauche.

Il y a aussi des instructions qui agissent sur un seul opérande

Les opérandes peuvent être des registres, des constantes ou le contenu de cases mémoire, on appelle ça le mode d'adressage

➔ **Adressage registre (R)**

L'opération se fait sur un ou 2 registres
INST R , R
INST R

Exemples :
INC AX : incrémenter le registre AX
MOV AX, BX : Copier le contenu de BX dans AX

➔ **Adressage Immédiat (IM)**

Un des opérande est une constante (valeur) :
INST R , im
INST taille [adr] , im

Exemples :
MOV AX, 243 : charger le registre AX par le nombre décimal 243
ADD AX, 243h : additionner le registre AX avec le nombre hexadécimal 243
MOV AX, 0xA243 : Quand le chiffre de gauche du nombre hexadécimal est une lettre, il est préférable d'utiliser le préfix 0x pour l'hexadécimal

MOV AL, 'a' : Charger le registre AL par le code ASCII du caractère 'a'

MOV AX, 'a' : Charger le registre AH par 00 et le registre AL par le code ASCII du caractère 'a'
MOV AX,'ab' : Charger AH par 'a' et AL par 'b'

➔ **Adressage direct (DA)**

Un des deux opérandes se trouve en **mémoire**. L'adresse de la case mémoire ou plus précisément son Offset est précisé directement dans l'instruction. L'adresse Rseg:Off doit être placée entre [ ], si le segment n'est pas précisé, DS est pris par défaut,

INST R , [adr]
INST [adr] , R
INST taille [adr] , im

Exemples :
MOV AX,[243] : Copier le contenu de la mémoire d'adresse DS:243 dans AX
MOV [123],AX : Copier le contenu de AX dan la mémoire d'adresse DS:123
MOV AX, [SS:243] : Copier le contenu de la mémoire SS:243 dans AX

➔ **Adressage indirect (IR)**

Un des deux opérandes se trouve en **mémoire**. L'offset de l'adresse n'est pas précisé directement dans l'instruction, il se trouve dans l'un des 4 registres d'offset BX, BP, SI ou DI et c'est le registre qui sera précisé dans l'instruction : [Rseg : Roff]. Si Rseg n'est pas spécifié, le segment par défaut sera utilisé (voir Tableau 2.1)

INST R , [Rseg : Roff]
INST [Rseg : Roff] , R
INST taille [Rseg : Roff] , im

Exemples :
MOV AX, [BX] ; Charger AX par le contenu de la mémoire d'adresse DS:BX
MOV AX, [BP] ; Charger AX par le contenu de la mémoire d'adresse SS:BP
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [DI] ; Charger AX par le contenu de la mémoire d'adresse DS:DI
MOV AX, [ES:BP] ; Charger AX par le contenu de la mémoire d'adresse ES:BP

L'adressage indirect est divisé en 3 catégories selon le registre d'offset utilisé. On distingue ainsi, l'adressage Basé, l'adressage indexé et l'adressage basé indexé,

▶ **Adressage Basé (BA)**

L'offset se trouve dans l'un des deux registres de base BX ou BP. On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset,

INST R , [Rseg : Rb+dep]
INST [Rseg : Rb+dep] , R
INST taille [Rseg : Rb+dep] , im

Exemples :
MOV AX, [BX] : Charger AX par le contenu de la mémoire d'adresse **DS**:BX
MOV AX, [BX+5] : Charger AX par le contenu de la mémoire d'adresse **DS**:BX+5
MOV AX, [BP-200] : Charger AX par le contenu de la mémoire d'adresse **SS**:BX-200
MOV AX, [ES:BP] : Charger AX par le contenu de la mémoire d'adresse **ES**:BP

➔ **Adressage Indexé (X)**

L'offset se trouve dans l'un des deux registres d'index SI ou DI. On peut préciser un déplacement qui sera ajouté au contenu de Ri pour déterminer l'offset,

INST R , [Rseg : Ri+dep]
INST [Rseg : Ri+dep] , R
INST taille [Rseg : Ri+dep] , im

Exemples :
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse **DS**:SI
MOV AX, [SI+500] ; Charger AX par la mémoire d'adresse **DS**:SI+500
MOV AX, [DI-8] ; Charger AX par la mémoire d'adresse **DS**:DI-8
MOV AX, [ES:SI+4] ; Charger AX par la mémoire d'adresse **ES**:SI+4

➔ **Adressage Basé Indexé (BXI)**

L'offset de l'adresse de l'opérande est la somme d'un registre de base, d'un registre d'index et d'un déplacement optionnel.
Si Rseg n'est pas spécifié, le segment par défaut du registre **de base** est utilisé :

INST R , [Rseg : Rb+Ri+dep]
INST [Rseg : Rb+Ri+dep] , R
INST taille [Rseg : Rb+Ri+dep] , im

Exemples :
MOV AX,[BX+SI] ; AX est chargé par la mémoire d'adresse **DS**:BX+SI
MOV AX,[BX+DI+5] ; AX est chargé par la mémoire d'adresse **DS**:BX+DI+5
MOV AX,[BP+SI-8] ; AX est chargé par la mémoire d'adresse **SS**:BP+SI-8
MOV AX,[BP+DI] ; AX est chargé par la mémoire d'adresse **SS**:BP+DI

TAILLE DES ECHANGES AVEC LA MEMOIRE

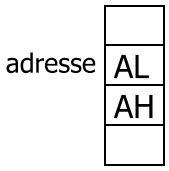
La mémoire est organisée en octets.

Quand on fait une instruction entre un registre et une donnée qui se trouve en mémoire, c'est le registre qui détermine la taille de l'opération.
Si le registre est un registre simple (8 bits), l'opération se fera avec une seule case mémoire.

MOV [adresse], AL donne ➔

Si le registre est un registre double (2 octets), l'opération se fera avec deux cases mémoires

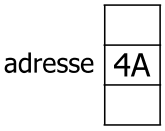
MOV [adresse], AX donne ➔



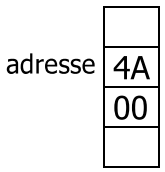
Remarquer que c'est la partie basse du registre qui est traitée en premier, et ceci dans les deux sens

Quand on fait une opération entre une **constante** et une **case mémoire**, il y a ambiguïté, le processeur ne sait pas s'il faut considérer la constante sur 8 bits ou sur 16 bits. Il faut utiliser les préfixes BYTE et WORD pour préciser le nombre d'octets á écrire :

MOV BYTE [adresse],4Ah ; On écrit 4A dans la position adresse



MOV WORD [adresse],4Ah ; On écrit 004A, 4A ➔ adresse, et 00 ➔ adresse+1



Exercice 2) (ram.asm) : tracer le programme ci-dessous

```
mov bx,4000h        ; adresse
mov ax,2233h
mov [bx],ax
mov word [bx+2],4455h
mov byte [bx+4],66
mov byte [bx+5],77
```

Afficher la ram en hexadécimal à partir de la position 4000H

Afficher la ram en décimal à partir de la position 4000H

2. INSTRUCTIONS DU 8086

2.1. Instructions de transfert

MOV Od , Os

Copie l'opérande Source dans l'opérande Destination

MOV R1 , R2	copier un registre dans un autre
MOV R , M	copier le contenu d'une case mémoire dans un registre
MOV M , R	copier un register dans une case mémoire
MOV R , im	copier une constante dans un registre
MOV taille M , im	copier une constante dans une case mémoire (taille = BYTE ou WORD)

~~MOV M , M~~

XCHG OD , OS

Echange l'opérande Source avec l'opérande Destination. Impossible sur segment.

XCHG R1 , R2
XCHG [adr] , R
XCHG R , [adr]

~~XCHG [ ] , [ ]~~

2.2. Les instructions Arithmétiques

Le 8086 permet d'effectuer les Quatre opérations arithmétiques de base, l'addition, la soustraction, la multiplication et la division. Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signés ou non signés. Les nombres signés sont représentés en complément à 2. Des instructions d'ajustement décimal permettent de faire des calculs en décimal (BCD).

▶ Addition :

ADD Od , Os Additionne l'opérande source et l'opérande destination avec résultat dans l'opérande destination,

Od + Os → Od

ADD AX,123 ADD AX,BX ADD [123],AX ADD BX,[SI]

ADC Od , Os Additionne l'opérande source, l'opérande destination et le curry avec résultat dans l'opérande destination,

Od + Os + C → Od

INC Op Incrémente l'opérande Op

Op + 1 → Op

Attention, l'indicateur C n'est pas positionné quand il y a débordement, C'est l'indicateur Z qui permet de détecter le débordement

Pour incrémenter une case mémoire, il faut préciser la taille :

INC byte [ ]

INC word [ ]

Exercice 4) :

En partant à chaque fois de la situation illustrée à droite. Quelle est la situation après chacune des instructions suivantes

INC byte [4000h]

INC word [4000h]

4000h→	25h
	33h

En partant à chaque fois de la situation illustrée à droite. Quelle est la situation après chacune des instructions suivantes

INC byte [4000h]

INC word [4000h]

4000h→	FFh
	33h

▶ Soustraction

SUB Od , Os Soustrait l'opérande source et l'opérande destination avec résultat dans l'opérande destination.

Od - Os → Od

SBB Od , Os Soustrait l'opérande source et le curry de l'opérande destination avec résultat dans l'opérande destination.

Od - Os - C → Od

DEC Op Décrémente l'opérande Op
Op - 1 → Op

NEG Op Donne le complément à 2 de l'opérande Op : remplace Op par son négatif
Cà2(Op) → Op

CMP Od , Os Compare (soustrait) les opérandes Os et Od et positionne les drapeaux en fonction du résultat. L'opérande Od n'est pas modifié

▶ Multiplication

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

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

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

```
MUL  BL          ;   AL x  BL      →   AX
MUL  CX          ;   AX x CX       →   DX:AX
MUL  byte  [BX] ;   AL x (octet pointé par BX) →   AX
MUL  word [BX] ;   AX x (word pointé par BX) →   DX :AX
```

~~MUL im~~ ~~MUL AX,R~~ ~~MUL AX, im~~

- Les drapeaux C et O sont positionnés si la partie haute du résultat est non nulle. La partie haute est AH pour la multiplication 8 bits et DX pour la multiplication 16 bits

Exercice 5) : (mul.asm) Tracer le programme ci-dessous en indiquant à chaque fois la valeur des indicateurs C et O

```
mov al,64h
mov bl,2
mul bl
mov al,64h
mov cl,3
mul cl
```

IMUL Op (Integer Multiply) Identique à MUL excepté qu'une multiplication signée est effectuée.

Exercice 6) (imul.asm) : différence entre MUL et IMUL
Tracer le programme ci-dessous

```
MOV     al,11111111b
mov     bl,00000010b
mul     bl
MOV     al,11111111b
mov     bl,00000010b
imul    bl
```

▶ Division

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

AX	Op_8
AH	AL

DIV Op_8 ;AX / Op_8 , Quotient → AL , Reste → AH

DIV S_{16} ;DX:AX / S_{16} , Quotient → AX , Reste → DX

DX:AX	Op_{16}
DX	AX

- S ne peut pas être une donnée (immédiat) ~~DIV 125h~~
- Après la division L'état des indicateurs est indéfini
- La division par 0 déclenche une erreur

IDIV Op Identique à DIV mais effectue une division signée

CBW (Convert Byte to Word) Effectue une extension de AL dans AH. On écrit le contenu de AL dans AX en respectant le signe

Si AL contient un nombre positif, On complète par des 0 pour obtenir la représentation sur 16 bits.

Si AL contient un nombre négatif, On complète par des 1 pour obtenir la représentation sur 16 bits.

+5 = 0000 0101 ⇒ 0000 0000 0000 0101

5 = 1111 1011 ⇒ 1111 1111 1111 1011

CWD (Convert Word to Double Word) effectue une extension de AX dans DX en respectant le signe. On écrit AX dans le registre 32 bits obtenu en collant DX et AX DX | AX

2.3. Les instructions logiques

NOT Op Complément à 1 de l'opérande Op

AND Od , Os ET logique
Od ET Os → Od

OR Od , Os OU logique
Od OU Os → Od

XOR Od , Os OU exclusif logique
Od OUX Os → Od

TEST Od , Os Similaire à AND mais ne retourne pas de résultat dans Od, seuls les indicateurs sont positionnés

Exercice 7) : (logic.asm) Tracer en binaire le programme ci-dessous

```
MOV     AX,125h
AND     AL,AH
```

Exercice 8) : (xor.asm) Programme qui fait AX ⊕ BX sans utiliser l'instruction XOR

2.4. Les masques logiques :

Le 8086 ne possède pas d'instructions permettant d'agir sur un seul bit. Les masques logiques sont des astuces qui permettent d'utiliser les instructions logiques vues ci-dessus pour agir sur un bit spécifique d'un octet out d'un word

Forcer un bit à 0 :

Pour forcer un bit à 0 sans modifier les autres bits, on utilise l'opérateur logique AND et ces propriétés :

- x AND 0 = 0 (0 = élément absorbant de AND)
- x AND 1 = x (1 = élément neutre de AND)

On fait un AND avec une valeur contenant des 0 en face des bits qu'il faut forcer à 0 et des 1 en face des bits qu'il ne faut pas changer

```
      xxxx xxxx
AND   1110 1101
-------------------
      xxx0 xx0x
```

Forcer un bit à 1 :

Pour forcer un bit à 1 sans modifier les autres bits, on utilise l'opérateur logique OR et ces propriétés :

- x OR 1 = 1 (1 = élément absorbant de OR)
- x OR 0 = x (0 = élément neutre de OR)

On fait un OR avec une valeur contenant des 1 en face des bits qu'il faut forcer à 1 et des 0 en face des bits qu'il ne faut pas changer

```
      xxxx xxxx
OR    0010 0000
-------------------
      xx1x xxxx
```

Inverser un bit :

Pour inverser la valeur d'un bit sans modifier les autres bits, on utilise l'opérateur logique XOR et ces propriétés :

- X XOR 1 = X
- X XOR 0 = X (0 = élément neutre de XOR)

Donc, on fait un XOR avec une valeur contenant des 1 en face des bits qu'il faut inverser et des 0 en face des bits qu'il ne faut pas changer

```
      xxxx xxxx
XOR   0010 0000
-------------------
      xxXx xxxx
```

Exercice 9) : (masque.asm) Tracer le programme ci-dessous

```
MOV     AX,1A25h
AND     AX,F0FFh
```

2.5. Les instructions de décalage

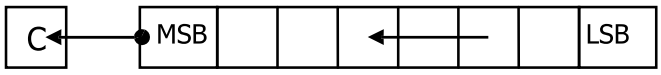
Dans les instructions de décalage, l'opérande k peut être soit une constante (immédiat) soit le registre CL :

```
INST  AX,1  ; décaler AX de 1 bit
INST  BL,4  ; décaler BL de 4 bits
INST  BX,CL ; décaler BX de CL bits
```

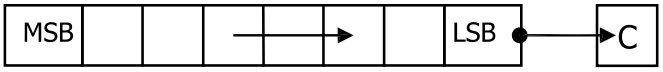
On peut aussi décaler le contenu d'une case mémoire mais il faut préciser la taille

```
INST  byte [BX],1   ; décaler une fois le contenu de la case
                    mémoire d'adresse BX
```

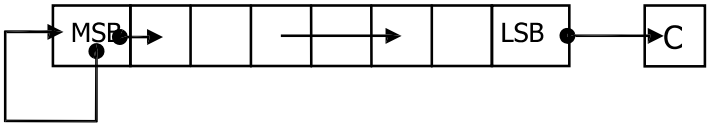
SHL R/M,k (SHift Logical Left) décalage logique à gauche de k bits
SAL R/M,k (SHift Arithmé Left) décalage arithmétique à gauche



SHR R/M,k (SHift Logical right) décalage logique à droite

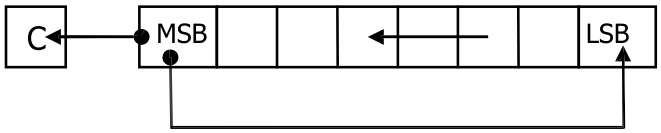


SAR R/M,k (SHift Arithmetic right) décalage arithmétique à droite

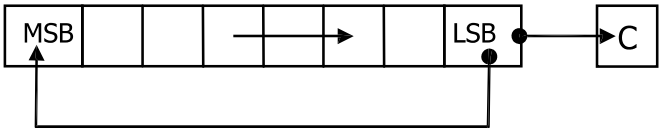


Les décalages arithmétiques permettent de conserver le signe. Ils sont utilisés pour effectuer des opérations arithmétiques comme des multiplications et des divisions par 2.

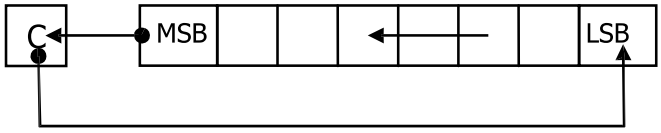
ROL R/M,k (Rotate Left) Rotation à gauche



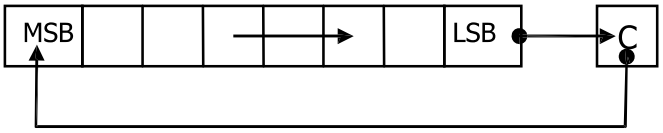
ROR R/M,k (Rotate Right) Rotation à droite



RCL R/M,k (Rotate Through CF Left) Rotation à gauche à travers le Carry



RCR R/M,k (Rotate Through CF Right) Rotation à droite à travers le Carry



2.6. Instructions agissant sur les indicateurs

15													2	1	0
				O	D	I	T	S	Z		A		P		C

CLC (CLear Carry) positionne le drapeau C à 0

STC (Set Carry) positionne le drapeau C à 1

CMC Complémente le drapeau C

CLD Positionne le Drapeau D à 0

STD Positionne le Drapeau D à

CLI Positionne le Drapeau I à 0

STI Positionne le Drapeau I à 1

LAHF

Copier l'octet bas du registre d'état dans AH

Après l'opération, on a :

AH :	S	Z		A		P		C

SAHF

Opération inverse de LAHF : Transfert AH dans l'octet bas du registre d'état

PUSHF

Empile le registre d'état,

POPF

Dépile le registre d'état,

Exercice 10) :

1. programme qui positionne l'indicateur P à 0 sans modifier les autres indicateurs
2. programme qui positionne l'indicateur O à 1 sans modifier les autres indicateurs

2.7. Les instructions de contrôle de boucle

LOOP xyz L'instruction **loop** fonctionne automatiquement avec le registre CX (compteur). Quant le processeur rencontre une instruction loop, il décrémente le registre CX. Si le résultat n'est pas encore nul, il reboucle à la ligne portant l'étiquette xyz, sinon il continue le programme à la ligne suivante

```
        MOV  CX,une valeur
ici:    XXX  xx,yy
        XXX  xx,yy
        ……….
        XXX  xx,yy
        XXX  xx,yy
        loop ici
        XXX  xx,yy
```

Remarque sur l'étiquette :

L'étiquette est une chaîne quelconque qui permet de repérer une ligne. Le caractère ':' à la fin de l'étiquette n'est obligatoire que si l'étiquette est seule sur la ligne

LOOPZ xyz (Loop While Zero) Décrémente le registre CX (aucun flag n'est positionné) on reboucle vers la ligne xyz tant que CX est différent de zéro et le flag Z est égal à 1. La condition supplémentaire sur Z, donne la possibilité de quitter une boucle avant que CX ne soit égal à zéro.

LOOPNZ xyz Décrémente le registre CX et reboucle vers la ligne xyz tant que CX est différent de zéro et le flag Z est égal à 0. Fonctionne de la même façon que loopz,

JCXZ xyz branchement à la ligne xyz si CX = 0 indépendamment de l'indicateur Z

Exercice 11) : (boucle.asm) Programme qui ajoute la valeur 3 au contenu des 100 cases mémoire du segment DATA dont l'adresse (offset) commence à 4000h

Exercice 12) : (boucle2.asm) Programme qui multiplie par 3 les 100 words contenu dans le segment DATA à partir de l'offset 4000h. On suppose que les résultats tiennent dans 16 bits (< 65536)

28. Les instructions de branchement

3 types de branchement sont possibles :

- Branchements inconditionnels
- Branchements conditionnels
- Appel de fonction ou d'interruptions

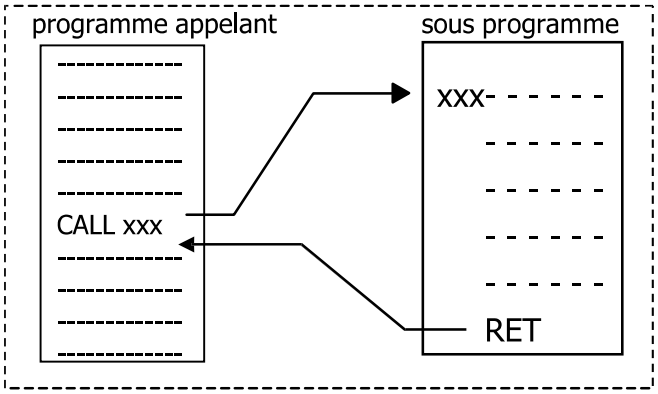
Tous ces transferts provoquent la poursuite de l'exécution du programme à partir d'une nouvelle position du code. Les transferts conditionnels se font dans une marge de -128 à +127 octets à partir de la position de transfert.

▶ Branchements inconditionnels

JMP xyz Provoque un saut sans condition à la ligne portant l'étiquette xyz.

CALL xyz Appel d'une procédure (sous programme) qui commence à la ligne xyz. La position de l'instruction suivant le CALL est empilée pour assurer une poursuite correcte après l'exécution du sous programme.

RET Retour de sous programme. L'exécution du programme continue à la position récupérée dans la pile.



INT n appel à l'interruption logicielle n° n

▶ Branchements conditionnels

Les branchements conditionnels sont conditionnés par l'état des indicateurs (drapeaux) qui sont eux même positionnés par les instructions précédentes.

Dans la suite nous allons utiliser la terminologie :

- **supérieur** ou **inférieur** pour les nombres non signés
- **plus petit** ou **plus grand** pour les nombres signés
- \+ pour l'opérateur logique OU

JE/JZ xyz (Jump if Equal or Zero) Aller à la ligne xyz si résultat nul ou si égalité. C'est-à-dire si Z=1

JNE/JNZ xyz (Jump if Not Equal or Not Zero) Aller à la ligne xyz si résultat non nul ou si différent. C'est-à-dire si Z=0

JA xyz (Jump if Above) aller à la ligne xyz si supérieur (non signé). C'est-à-dire si C + Z = 0

JAE xyz (Jump if Above or Equal) aller à la ligne xyz si supérieur ou égal (non signé). C'est-à-dire si C = 0

JB xyz (Jump if Bellow) Branche si inférieur (non signé). C'est-à-dire si C = 1

JBE xyz (Jump if Bellow or Equal) aller à la ligne xyz si inférieur ou égal (non signé). C'est-à-dire si C + Z = 1

JC xyz (Jump if CArry) aller à la ligne xyz s'il y a retenu. C'est-à-dire si C = 1

JNC xyz (Jump if No CArry) aller à la ligne xyz s'il n'y a pas de retenu. C'est-à-dire si C = 0

JG xyz (Jump if Grater) aller à la ligne xyz si plus grand (signé). C'est-à-dire si (S ⊕ O) + Z = 1

JGE xyz (Jump if Grater or Equal) aller à la ligne xyz si plus grand ou égal (signé). C'est-à-dire si S ⊕ O = 0

JL xyz (Jump if Less) aller à la ligne xyz si plus petit (signé). C'est-à-dire si S ⊕ O = 1

JLE xyz (Jump if Less or Equal) aller à la ligne xyz si plus petit ou égal (signé). C'est-à-dire si (S ⊕ O) + Z = 1

JO xyz (Jump if Overflow) aller à la ligne xyz si dépassement. C'est-à-dire si O = 1

JNO xyz (Jump if No Overflow) aller à la ligne xyz s'il n'y a pas de dépassement O = 0

JP/JPE xyz (Jump if Parity or Parity Even) aller à la ligne xyz si parité paire. C'est-à-dire si P = 1

JNP/JPO xyz (Jump if No Parity or if Parity Odd) aller à la ligne xyz si parité impaire. C'est-à-dire si P = 0

JS xyz (Jump if Sign) aller à la ligne xyz si signe négatif. C'est-à-dire si S = 1

JNS xyz (Jump if No Sign) aller à la ligne xyz si signe positif. C'est-à-dire si S = 0

Exercice 13) :
Ecrire la suite d'instructions pour réaliser les étapes suivantes :

1. Mettre 1 dan AX
2. incrémenter AX
3. si AX < 200 recommencer au point 2
4. sinon copier AX dans BX

Exercice 14) :
Ecrire la suite d'instructions pour réaliser les étapes suivantes :

1. copier le contenu de la case mémoire [1230h] dan CX
2. Comparer CX à 200
    a. si < incrémenter CX et recommencer au point 2
    b. si > décrémenter CX et recommencer au point 2
    c. si = copier CX dans AX et continuer le programme

Exercice 15) : (cherche.asm)
Programme qui cherche la valeur 65 dans le RAM à partir de la position 4000h. Une

fois trouvée, placer son adresse dans le registre AX

Exercice 16) : (boucle3.asm) Programme qui divise par 3 les 100 octets contenus dans les 100 cases mémoires commençant à l'adresse 4000h. Ne pas utiliser l'instruction **loop**

2. .9 Instructions d'accès aux ports d'E/S

IN AL/AX,port lire un port d'entrée sortie.
IN AL , port ; charge le contenu du port d'adresse port dans AL
IN AX , port ; charge le contenu du port d'adresse port dans AL et le contenu du port d'adresse port+1 dans AH

Si l'adresse port tient sur 8 bits, elle peut être précisée immédiatement, sinon il faut passer par le registre DX

```
IN  AL , 4Dh          MOV  DX , 378h
                      IN   AL , DX
```

OUT port , AL/AX Ecriture dans un port d'E/S
L'adressage du port se fait de la même façon que pou IN

Out port , AX ; écrit AL dans port et AH dans port+1

2.6 CE QU'IL NE FAUT PAS FAIRE

Voici une liste non exhaustive de quelques erreurs à éviter

- Une opération ne peut pas se faire entre deux cases mémoire, il faut que ça passe par un registre. On ne peut pas avoir des instructions du style :

~~MOV [245],[200]~~
~~ADD [BX],[BP]~~

~~INST [ ], [ ]~~

- Bien que le registre SP soit un registre d'adressage, il ne peut être utilisé directement pour accéder à la pile, on peut toutefois le copier dans un registre valide pour l'adressage (BX, BP,SI, DI) et utiliser ensuite ce dernier :

~~MOV AX,[SP]~~

```
MOV   BP,SP
MOV   AX,[BP]
```

- On ne peut pas faire des opérations directement sur un registre segment, il faut passer par un autre registre. On ne peux pas non plus copier un registre segment dans un autre

~~MOV ES,02F7H~~

```
MOV   BX,02F7h
MOV   ES,BX
```

~~MOV ES,DS~~

```
MOV   AX,DS
MOV   ES,AX
```

~~INC ES~~

```
MOV   BX,ES
INC   BX
MOV   ES,BX
```

~~ADD ES,2~~

```
MOV   BX,ES
ADD   BX,2
MOV   ES,BX
```

- On ne peut pas utiliser directement une adresse segment dans une instruction, il faut passer par un registre segment.

~~MOV [2A84h : 55],AX~~

```
MOV   BX,2A84h
MOV   ES,BX
MOV   [ES : 55] , AX
```

- On ne peut pas faire une multiplication ou une division sur une donnée (immédiat)

~~MUL im~~
~~DIV im~~

- On ne peut pas empiler/dépiler un opérande 8 bits

~~PUSH R/M_8~~

- Le segment et l'offset sont séparé par le caractère ":" et non ","

~~[DS , BX]~~ [DS : BX]

- On ne peut pas utiliser les opérateurs + - x sur des registre ou des mémoires

MOV AX, ~~BX+2~~
MOV AX, ~~DX x 2~~

A signifie Accumulateur qui peut être AL ou AX selon la taille de l'opération 8 bits ou 16 bits
X est l'extension de l'accumulateur. (8 bits : X =AH→ X:A = AH:AL = AX) (16 bits : X = DX→ X:A = DX:AX)

AX	AH	AL	SP	CS
BX	BH	BL	BP	DS
CX	CH	CL	SI	SS
DX	DH	DL	DI	ES

					O	D	I	T	S	Z		A		P		C

* : *Positionné selon le résultat*
? : *Modifié, résultat non défini*

Transfert		O	S	Z	A	P	C
MOV D , S	D ← S						
PUSH R/M	empile R/M (16 bits)						
R/M	dépile R/M (16 bits)						
XCHG D , S	D ⇆ S						
XLAT	AL ← [BX+AL]						
LEA R_{16} , M	R_{16} ← offset de M						
LDS R_{16} , [a]	R_{16} ← [a], DS←[a+2]						
LES R_{16} , [a]	R_{16} ← [a], ES←[a+2]						
Arithmétique							
ADD D , S	D ← S + D	*	*	*	*	*	*
ADC D , S	D ← S + D + C	*	*	*	*	*	*
INC D	D ← D + 1	*	*	*	*	*	!
SUB D,S	D ← D - S	*	*	*	*	*	*
SBB D,S	D ← D – S - C	*	*	*	*	*	*
NEG D	D ← -D	*	*	*	*	*	*
CMP D,S	D - S	*	*	*	*	*	*
DEC D	D ← D - 1	*	*	*	*	*	*
MUL S	X:A ←A x S	*	?	?	?	?	*
IMUL S	X:A ←A x S (signée)	*	?	?	?	?	*
DIV S	A←(X:A) / S, X←Rst	?	?	?	?	?	?
IDIV S	division signée	?	?	?	?	?	?
CBW	Byte(AL) to word(AX) (signé)						
CWD	Word(AX) to Dword (DX|AX) (signé)						
DAA	Dec. adj. after ADD	?	*	*	*	*	*
AAA	ascii adj after ADD	?	?	?	*	?	*
DAS	dec adj after SUB	?	*	*	*	*	*
AAS	ascii adj after SUB	?	?	?	*	?	*
AAM	ascii adj after MUL	?	*	*	?	*	?
AAD	ascii adj before DIV	?	*	*	?	*	?
Logique							
NOT R/M	R/M ← R/M						
AND D , S	D ← D AND S	0	*	*	?	*	0
OR D , S	D ← D OR S	0	*	*	?	*	0
XOR D , S	D ← D XOR S	0	*	*	?	*	0
TEST D , S	D AND S	0	*	*	?	*	0
SHL/SHL R/M	C ◄ ◄ ◄ ◄ ◄ 0	*	*	*	?	*	*
SHR R/M	0 ► ► ► ► ► C	*	*	*	?	*	*
SAR R/M	► ► ► ► ► C	*	*	*	?	*	*
ROL R/M	C ◄ ◄ ◄ ◄ ◄						*
ROR R/M	► ► ► ► ► C						*
RCL R/M	C ◄ ◄ ◄ ◄ ◄						*
RCR R/M	► ► ► ► ► C						*
Chaînes							
MOVSb/w	[ES:DI] ←[DS:SI],						
SCASb/w	AL/AX – [ES:DI];	*	*	*	*	*	*
LODSb/w	AL/AX ← [DS:SI]						
CMPSb/w	[ES:DI] - [DS:SI],	*	*	*	*	*	*
STOSb/w	AL/AX → [ES:DI]	*	*	*	*	*	*

Branchement		O	S	Z	A	P	C
JMP a	branche à l'adresse a						
JE/JZ a	saut si = (Z levé) POP						
JNE/JNZ a	saut si ≠ (Z baissé)						
JA a	saut si > (non signé)						
JAE a	saut si ≥ (non signé)						
JB a	saut si < (non signé)						
JBE a	saut si ≤ (non signé)						
JG a	saut si > (signé)						
JGE a	saut si ≥ (signé)						
JL a	saut si < (signé)						
JLE a	saut si ≤ (signé)						
JC a	saut si C = 1						
JNC a	saut si C = 0						
JO a	saut si O = 1						
JNC a	saut si C = 0						
JP/JPE a	saut si P = 1						
JNP/JPO a	saut si P = 0						
JS a	saut si S = 1						
JNS a	saut si S = 0						
CALL a	saut à sous programme						
RET	retour de sous programme						
LOOP A	dec CX, saut si CX ≠ 0						
LOOPZ a	dec CX, saut si :CX ≠ 0 et Z = 1						
LOOPNZ a	dec CX, saut si :CX ≠ 0 et Z = 0						
JCXZ a	saut si CX = 0 (∀ Z)						
action sur Indicateurs							
LAHF	AH ← RE_L						
SAHF	RE_L ← AH		*	*	*	*	*
PUSHF	empile RE						
POPF	dépile RE	*	*	*	*	*	*
CLC	Clear Carry flag						0
STC	Set Carry flag						1
CMC	complémente Carry						*
CLD	Clear Direction flag						
STD	Set Direction flag						
CLI	Clear Interrupt flag						
STI	Set Interrupt flag						
Divers							
INT n	déclenche l'interrupt n						
INTO n	interrupt si Overflow						
IRET	Retour d'interruption	*	*	*	*	*	*
HALT	entre en mode vail						
WAIT	entre en attente						
NOP	ne fait rien						
IN al/ax,port	Lit un port d' E/S						
out port,al/ax	écrit dans un port						