

La gamme de microprocesseurs 80x86 équipe les micro-ordinateurs de type PC et compatibles. Les premiers modèles de PC, commercialisés au début des années 1980, utilisaient le 8086, un microprocesseur 16 bits. Les modèles suivants ont utilisé successivement le 80286, 80386, 80486 et Pentium (ou 80586). Chacun de ces processeurs est plus puissant que les précédents : augmentation de la fréquence d'horloge, de la largeur de bus (32 bits d'adresse et de données), introduction de nouvelles instructions (par exemple calcul sur les réels) et ajout de registres. Chacun d'entre eux est *compatible* avec les modèles précédents ; un programme écrit dans le langage machine du 286 peut s'exécuter sans modification sur un 486. L'inverse n'est pas vrai, puisque chaque génération a ajouté des instructions nouvelles. On parle donc de *compatibilité ascendante*.

Le microprocesseur 80286 est organisé en quatre modules travaillant en parallèle :

1. L'unité de bus, qui gère le bus : les bus comprennent:
 - a. Le bus de données sur 16 bits ;
 - b. Le bus d'adresses sur 24 bits ;
 - c. Le bus de commandes.
2. L'unité des instructions, encadrée par deux files d'attente : elle reçoit les instructions et les décode.
3. L'unité d'exécution, chargée d'exécuter les instructions.
4. L'unité des adresses, qui gère la mémoire et les protections et calcule les adresses.

1. Registres internes du 80286

Le microprocesseur 80286 possède huit registres généraux sur huit (08) bits chacun notés :

- AL et AH
- BL et BH
- CL et CH
- DL et DH

Ces registres peuvent être utilisés tels ou associés par paire pour constituer des registres de 16 bits. Dans ce cas :

- La lettre **L** indique qu'il s'agit de l'octet de poids faible.
- La lettre **H** spécifie l'octet de poids fort.

La paire résultante devient AX pour AH+AL ou BX pour BH+BL, ainsi on obtient :

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

Ces registres sont à la disposition du programmeur, ils peuvent avoir des rôles précis avec certaines opérations.

Les registres de base, d'index et le pointeur de pile servent essentiellement à calculer des adresses en fournissant un décalage ou un déplacement (offset). Ils interviennent aussi dans certaines opérations.

- Le pointeur de pile est **SP** « *Stack Pointer* »
- Le registre de base s'appelle **BP** « *Base Pointer* »
- Les deux registres index sont **SI** « *Source Index* » et **DI** « *Destination Index* », ils servent essentiellement à gérer des déplacements de données.

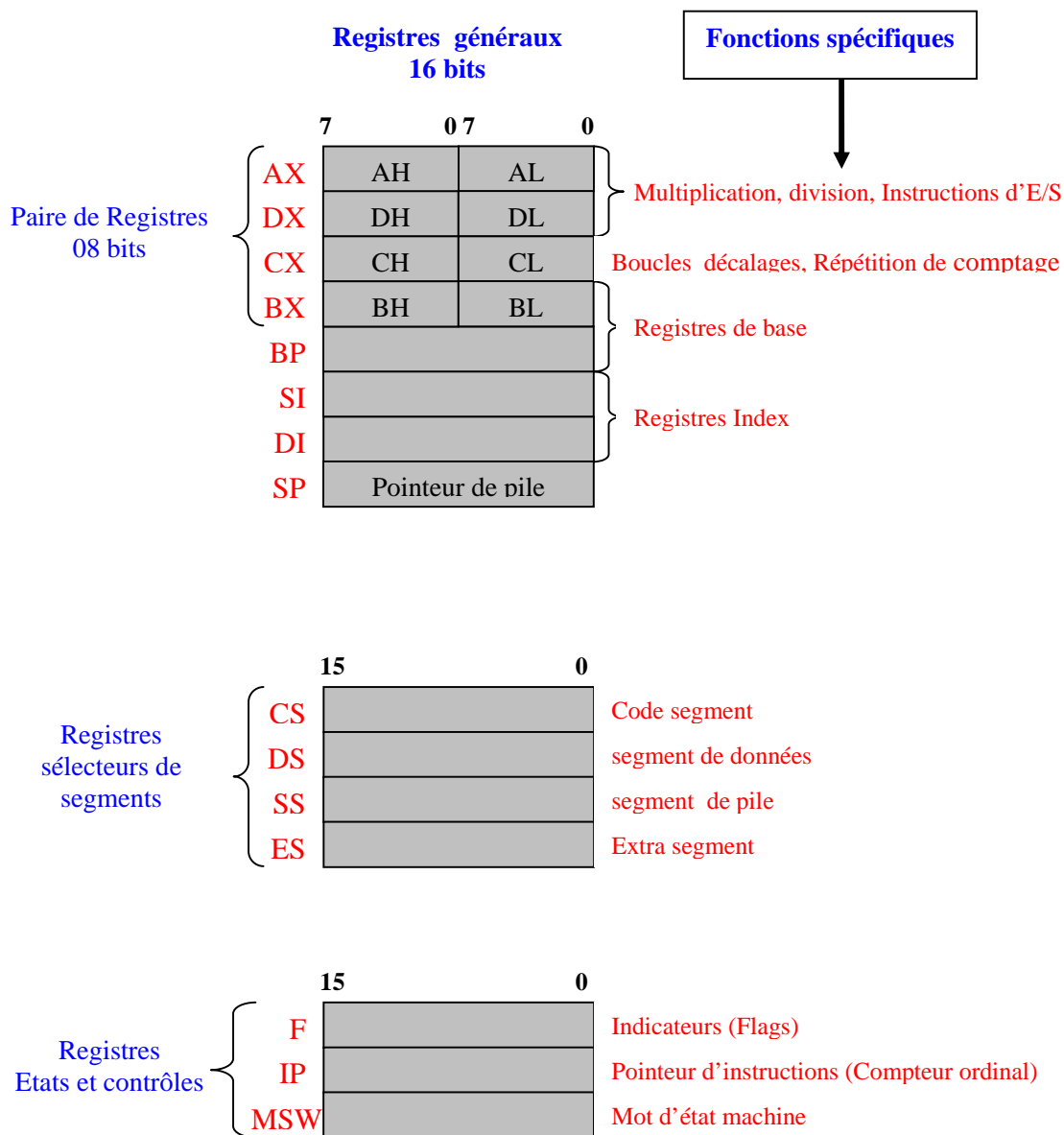


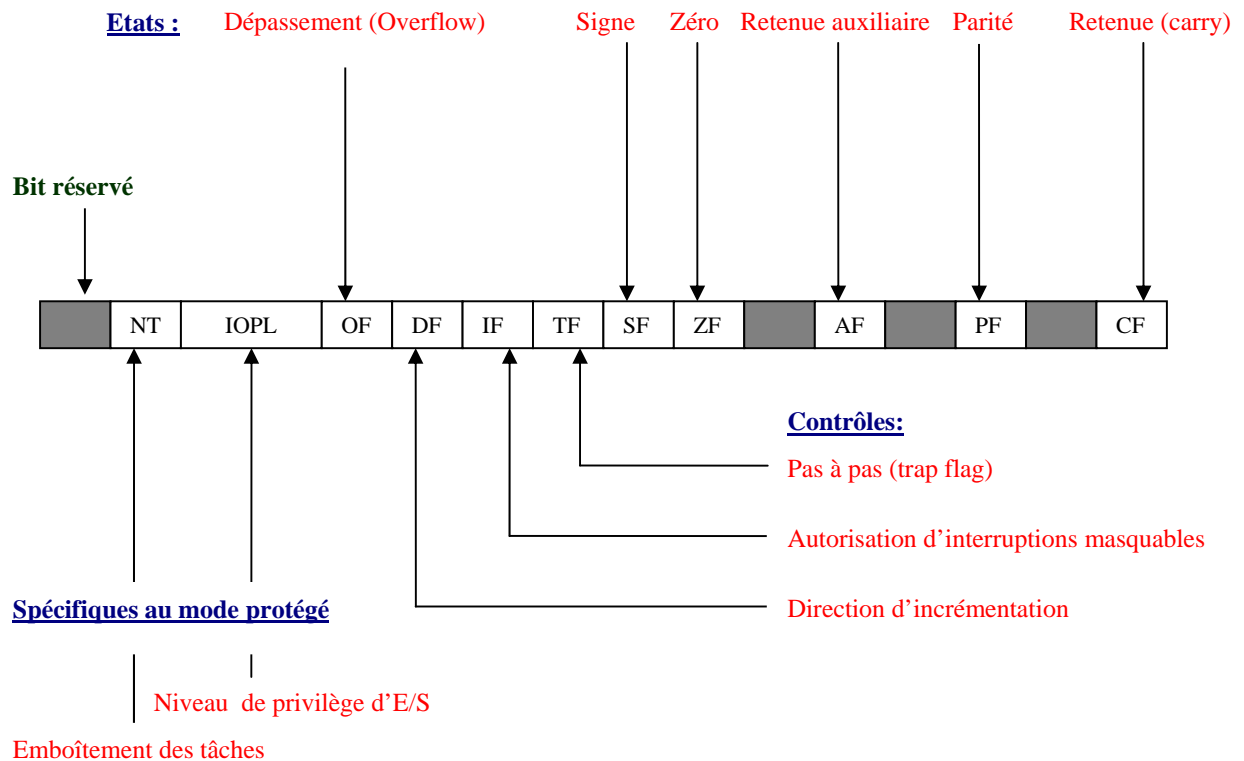
Fig.01. Les registres du microprocesseur 80286.

Le 80286 dispose aussi de quatre registres sélecteurs de segments, ces registres interviennent dans l'établissement des adresses en mémoire :

- **CS** est le sélecteur de segment de code,
- **DS** est le sélecteur de segment de données,
- **SS** est le sélecteur de segment de pile,
- **ES** est un sélecteur supplémentaire ou « extra-sélecteur ».

Les registres d'états et de contrôles du microprocesseur 80286 sont composés :

- Un registre pointeur **IP** « *Instruction Pointer* » dont le rôle est celui d'un compteur ordinal. Il spécifie l'adresse de la prochaine instruction à exécuter, en association avec le sélecteur de segment de code **CS**, toutefois.
- Deux registres d'indicateurs sur 16 bits. L'un regroupe les indicateurs habituels appelés « flags », ces indicateurs sont regroupés dans le registre **F** (Voire figure Fig.02.). L'autre registre témoigne de l'état de la machine et il est nommé **MSW** « *Machine Status Word* ».



bit	Nom	Fonction
0	CF	Mis à 1 sur retenue, à 0 autrement.
2	PF	Parité. Mis à 1 si l'octet de poids faible du résultat est pair.
4	AF	Retenue sur quartet de faible poids du registre AL.
6	ZF	Indicateur de zéro. Mis à 1 si le résultat est nul.
7	SF	Signe : identique au bit du plus fort poids.
8	TF	Pas à pas : à 1, déclenche une interruption après la prochaine instruction puis revient à 0.
9	IF	Autorisation d'interruptions masquables.

10	DF	Direction d'incrémentation. Mis à 1, décrémente l'index visé ; à 0, l'incrémente.
11	OF	Dépassement : sur résultat signé, passe alors à 1.
12 et 13	IOPL	Niveau de privilège d'entrée-sortie.
14	NT	Emboîtement de tâches.

Fig.02. Les registre mot des indicateurs d'états du microprocesseur 80286.

2. Segmentation de la mémoire

L'espace mémoire que le processeur 80286 peut adresser est de 1 Méga octets en mode réel (16 Méga octets en mode protégé). Dans les deux cas, le concept de segmentation est utilisé. La mémoire globale est divisée en segments. Un segment est défini par:

- Son adresse physique, absolue, de départ dans la mémoire.
- Sa longueur : la longueur maximale d'un segment est de 64 Ko.

La définition des segments « adresse de départ et longueur » sont à la charge du programmeur. Plusieurs segments peuvent être utilisés. Dans ce cas, les segments peuvent se suivre en mémoire, ou être totalement disjoints, ou alors se chevaucher partiellement ou totalement. En règle générale, un segment devra commencer sur une adresse divisible par 16 (une articulation).

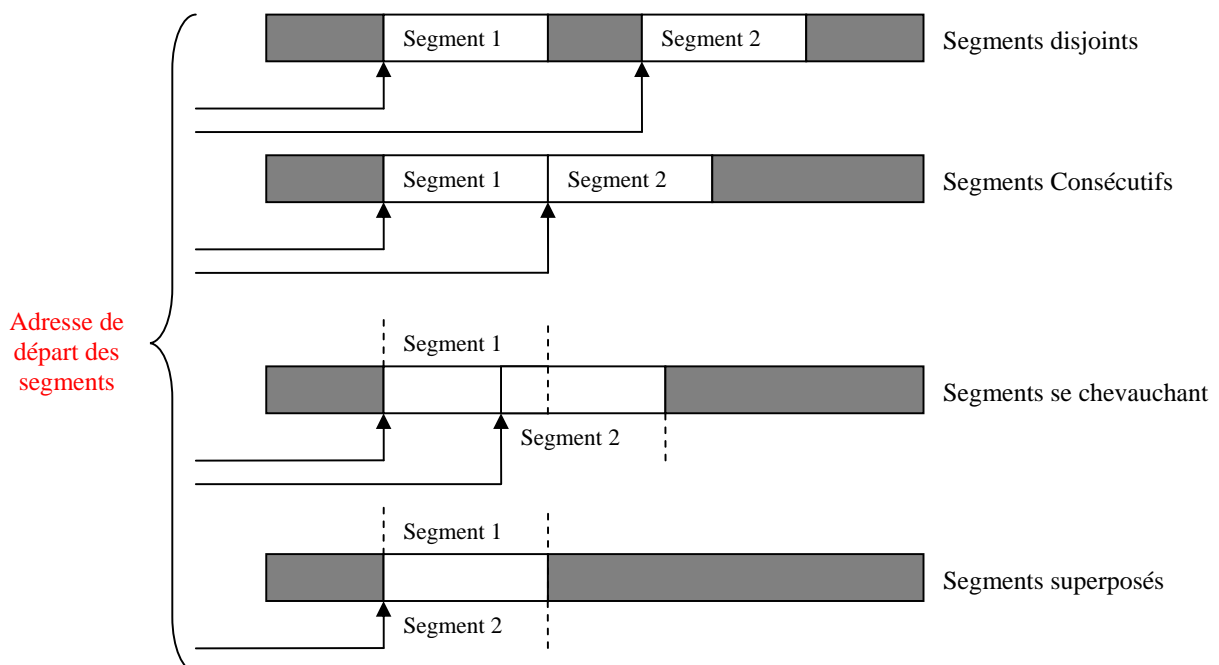
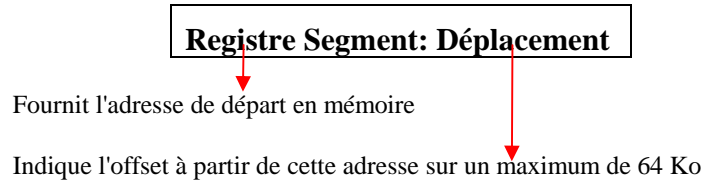


Fig.03. Distribution possible de segments dans la mémoire.

A un instant donné, l'adresse d'une information est définie à l'aide du contenu de deux registres:

- Le premier sélectionne l'adresse de départ du segment, il s'agit de l'un des registres sélecteurs de segments.
- Le second indique le déplacement "offset" dans ce segment.

Une telle adresse est notée comme suit:



Pour les instructions d'un programme, la paire de registres concernés est le registre sélecteur de segment de code CS et le registre pointeur d'instructions IP. Lorsque le processeur lit le code d'une instruction, l'adresse 32 bits est formée à l'aide du registre segment CS et du registre déplacement IP. La paire de ces deux registres est notée **CS: IP**

L'emploi très général des registres, lorsqu'il s'agit de désigner une adresse est résumé par le tableau suivant:

Références mémoires	Registre de segment		Offset (déplacement)
	Par défaut	Remplacement possible par	
Recherche d'instruction	CS	Aucun	IP
Opération sur pile	SS	Aucun	SP
Variable	DS	CS, ES, SS	Adresse effective
Source, pour une chaîne	DS	CS, ES, SS	SI
Destination, pour une chaîne	ES	Aucun	DI
Registre BP remplaçant le registre de base	SS	CS, DS, ES	Adresse effective
Registre BX remplaçant le registre de base	DS	CS, ES, SS	Adresse effective

L'adresse effective d'une cellule mémoire correspond au déplacement dans le segment. L'adresse physique, absolue, est l'adresse réelle dans la mémoire en tenant compte à la fois du segment et du déplacement.

2.1. Calcul de l'adresse absolue

Le processus de calcul de l'adresse absolue consiste à partir de l'adresse de départ du segment sur 16 bits (enregistré dans un registre sélecteur) à ajouter 4 bits à 0, en faible poids afin de former un mot de 20 bits, capable donc d'adresser 1 Mo. (L'adresse d'un segment est multiple de 16). Puis au résultat, ajouter le déplacement, fournit par le second registre.

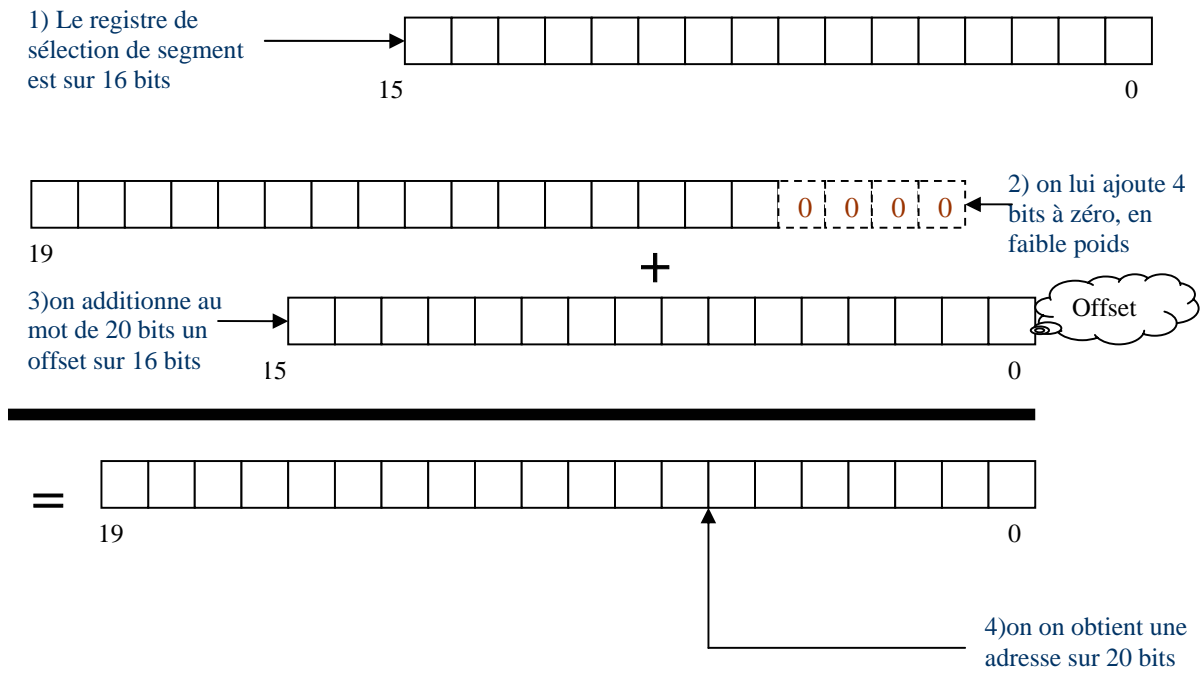


Fig.04. Principe de calcul de l'adresse physique sur le 80286.

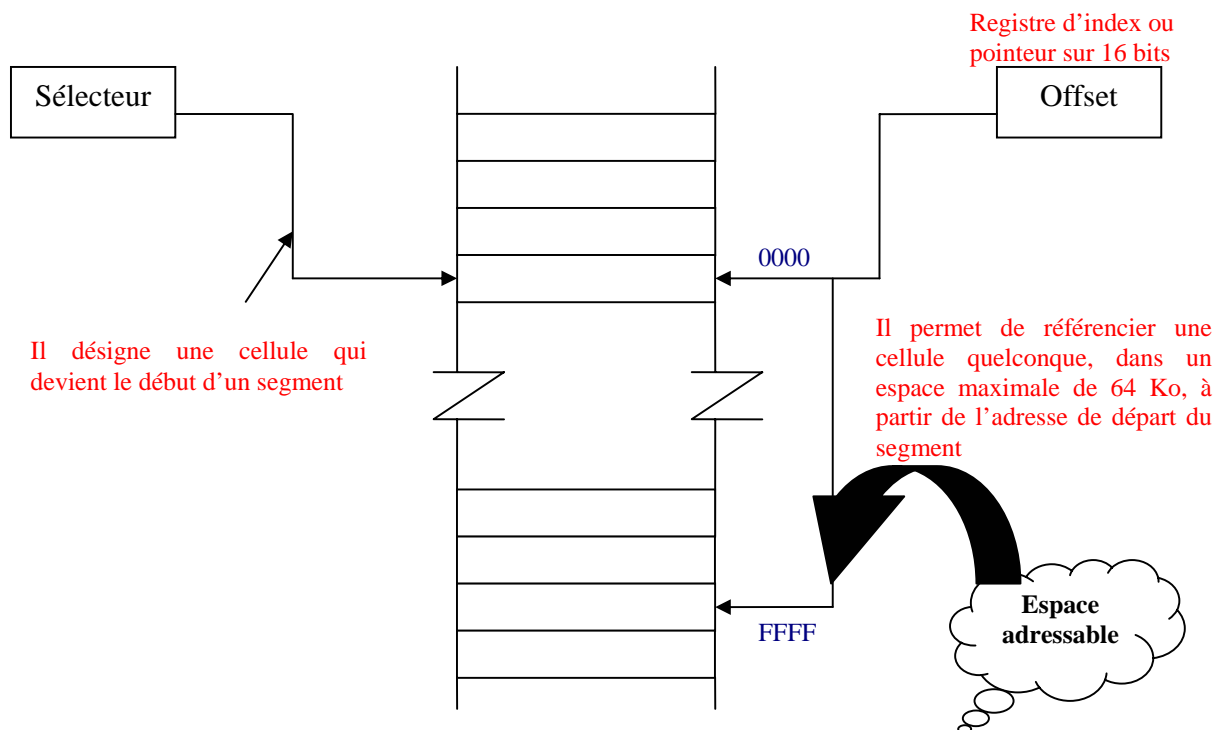


Fig.05. Principe de l'adressage sur le 80286.

Lorsqu'on lance l'exécution d'un programme, le DOS lit tout d'abord le PSP (Préfixe de segment programme : un bloc de contrôle élaboré par le DOS, qui contient des informations relatives au programme.) et initialise les registres selon des règles rigoureuses :

- Le registre sélecteur de segment de données **DS** et le sélecteur de l'extra-segment **ES** reçoivent l'adresse de début du PSP.
- Le registre sélecteur de segment de code **CS** reçoit l'adresse de début du programme.
- Le registre sélecteur de segment de pile **SS** reçoit l'adresse de début de la pile de sauvegarde.
- Le registre sélecteur de pointeur de pile **SP** reçoit l'adresse de début de la fin de la pile.
- Le registre pointeur d'instructions **IP** reçoit une adresse se trouvant dans le segment de code.

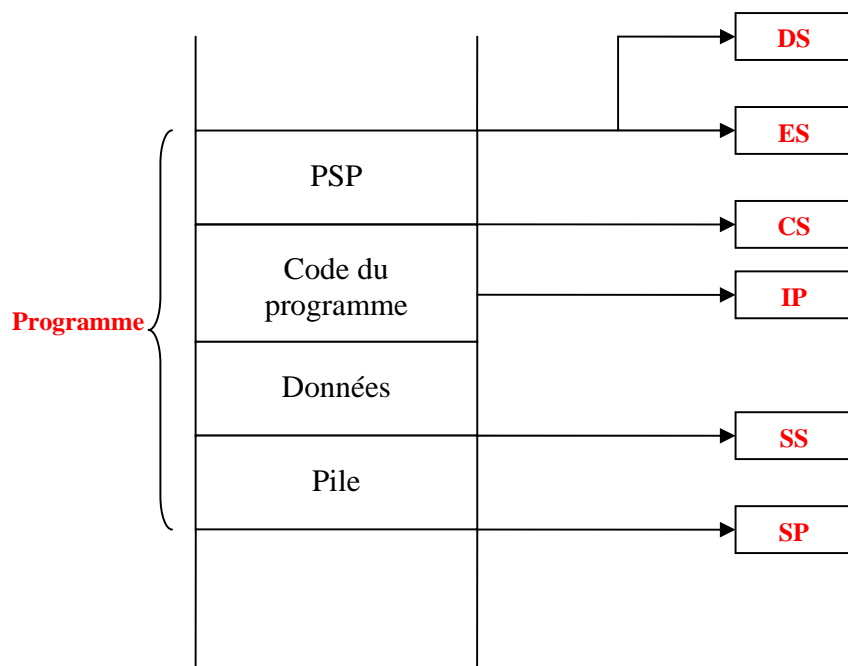


Fig.06. Initialisation des registres du microprocesseur par le DOS pour un programme.

3. Structure d'un programme

Un programme assembleur 80286 est composé d'instructions appelées du code, l'adresse de départ du segment de code doit être logée dans le registre sélecteur de segment de code **CS**. Donc, il faut indiquer à l'assembleur :

- où commence le segment,
- où il se termine, avec la ligne **CODE ENDS** (fin du segment code).


Un programme peut être divisé en séquences complètes appelées des procédures; chaque séquence (procédure) porte un nom et comporte une information la situant par rapport à la séquence précédente: proche (*NEAR*) ou éloignée (*FAR*). On spécifie ainsi à l'assembleur que cette procédure qu'elle se situe dans le même segment de code que la précédente (*NEAR*), ou alors qu'il faut changer de segment (*FAR*). Systématiquement une procédure se termine par un *ENDP*.


```

CODE SEGMENT                ; début du segment code
ADDITION PROC FAR          ; début de la procedure ADDITION

    MOV DL, 5
    ADD DL, 3

    ADD DL, 48      ; conversion du résultat binaire en ASCII pour l'affichage

    MOV AH, 2
    INT 21H      

    MOV AH, 4CH
    INT 21H      

ADDITION ENDP
CODE ENDS

END ADDITION

```

3.1. La déclaration des segments

Trois ou quatre zones mémoires, de préférence consécutives vont être occupées. L'assembleur se charge d'établir leurs adresses exactes "physiques" à la condition, toutefois que le programmeur définit la longueur maximale de ces zones ou leur occupation. Ces zones constituent les segments où chacune est pointée par un registre sélecteur de segment avec:

Registre	Nom du registre	Segment pointé
CS	Code segment	Code: programme constitué d'une liste d'instructions à exécuter
DS	Data segment	Données utilisées par les instructions du programme
SS	Stack segment	Pile de sauvegarde ou une zone de rangement de sécurité
ES	Extra segment	Un segment supplémentaire

L'assembleur offre aussi des paramètres facultatifs afin de compléter la définition de ces segments:

Label du segment Pile	<ul style="list-style-type: none"> - Ce segment doit démarrer sur une articulation (adresse divisible par 16) ce qu'on appelle un "paragraphe" standard. - un nom qui permet de référencier cette pile par exemple "Pile" - Il appartient à un ensemble de segments, une classe, dont le nom sera par exemple "Pile". La classe est utile pour des programmes importants.
Segment Pile	<ul style="list-style-type: none"> - Une certaine capacité est réservée pour la pile. Ces cellules peuvent être initialisées par des 0. - fin du segment.
Label du segment données	<ul style="list-style-type: none"> - Il doit démarrer aussi sur une articulation. - spécifier s'il s'agit de données publiques, non spécifiques à ce programme. - Donner à un nom au segment par exemple "Données". Ce segment appartient aussi à une classe.
Segment données	<ul style="list-style-type: none"> - Liste des données par octets ou mots, ou . . ., ou encore réservation d'espaces. - fin du segment.
Label du segment code	<ul style="list-style-type: none"> - Il doit démarrer aussi sur une articulation. - spécifier s'il s'agit d'instructions publiques. - Donner à un nom au segment par exemple "Code".
Segment code	<ul style="list-style-type: none"> - La liste des instructions qui constituent le programme. - fin du segment.

La directive **ASSUME** permet de déclarer un segment en assembleur, cette directive signifie « assumer » ou encore « affecter » par exemple **ASSUME CS: CODE** signifie que le registre CS assume la charge de contenir l'adresse de départ du segment code.

Exemple:

```

PILE          SEGMENT PARA STACK 'PILE'
              DB 256                DUP(0)
PILE          ENDS

DONNEES      SEGMENT PARA PUBLIC 'DONNES'
Nombre      DB    5
PLUS        DB    3
SOM         DB    ?
DONNEES     ENDS

CODE         SEGMENT PARA PUBLIC 'CODE'
ADDITION    PROC FAR
            ASSUME    CS: CODE

```

```

ASSUME    DS: DONNEES
ASSUME    SS: PILE

MOV  AX, DONNEES
MOV  DS, AX

MOV  AL, Nombre
ADD  AL, Plus
MOV  Som, AL

MOV  DL, Som
ADD  DL, 48

MOV  AH, 2
INT  21H

MOV  AH, 4CH
INT  21H

ADDITION  ENDP
CODE      ENDS
          END ADDITION

```

Remarque: Lors de l'initialisation des registres sélecteurs de segments, le DS ne reçoit pas l'adresse de départ du segment de données, il faut la lui fournir.

3.2. Segment de données

Les variables sont déclarées à l'aide des directives (pseudo instructions). L'assembleur attribuera à chaque variable une adresse. Dans le programme, on repère les variables grâce à leurs noms (identificateurs). Les noms des variables (comme les étiquettes) sont composés d'une suite de 31 caractères au maximum commençant obligatoirement par une lettre. Le nom peut comporter des majuscules, des minuscules, des chiffres, plus des caractères @ , ? et _. Lors de la déclaration d'une variable, on peut lui affecter une valeur initiale.

La directive **DB** (*Define Byte*) permet de déclarer des variables de un octet en réservant un octet pour la variable déclarée.

```

Syntaxe :  <Identificateur>  DB  <Valeur>
Exemple :  DATA SEGMENT
                Nbre             DB  5
                Res              DB  3
                SOM              DB  ?
                DATA ENDS

```

La directive **DW** (*Define Word*) permet de déclarer des variables de deux octets en réservant deux octets pour la variable déclarée.

Syntaxe : <Identificateur> DW <Valeur>
Exemple :

```
DATA SEGMENT
    entree DW 15 ; 2 octets initialises a 15
    sortie DW ? ; 2 octets non initialises
    cle DB ? ; 1 octet non initialise
    nega DB -1 ; 1 octet initialise a -1
DATA ENDS
```

Les valeurs initiales peuvent être données en hexadécimal (constante terminée par H) ou en binaire (terminée par b) :

Exemple :

```
DATA SEGMENT
    truc DW 0F0AH ; en hexa
    masque DB 01110000b ; en binaire
DATA ENDS
```

Les variables sont référenciés dans le programme en les désignant par leur nom. Après la déclaration précédente, on peut écrire par exemple :

Exemple :

```
MOV AL, Nb
ADD AL, Res
MOV Som, AL
```

Exemple :

```
MOV AX, truc
AND AL, masque
MOV truc, AX
```

L'assembleur se charge de remplacer les noms de variable par les adresses correspondantes.

Il est aussi possible de déclarer des tableaux, c'est à dire des suites d'octets ou de mots consécutifs. Pour cela, utiliser plusieurs valeurs initiales :

Exemple :

```
DATA SEGMENT
    Machine db 10, 0FH ; 2 fois 1 octet
    Max db -2, 'ALORS'
DATA ENDS
```

Remarquez la déclaration de la variable `Max` : un octet à -2 (=FEH), suivi d'une suite de caractères. L'assembleur n'impose aucune convention pour la représentation des chaînes de caractères : c'est à l'utilisateur d'ajouter si nécessaire un octet nul pour marquer la fin de la chaîne.

Après chargement de ce programme, la mémoire aura le contenu suivant :

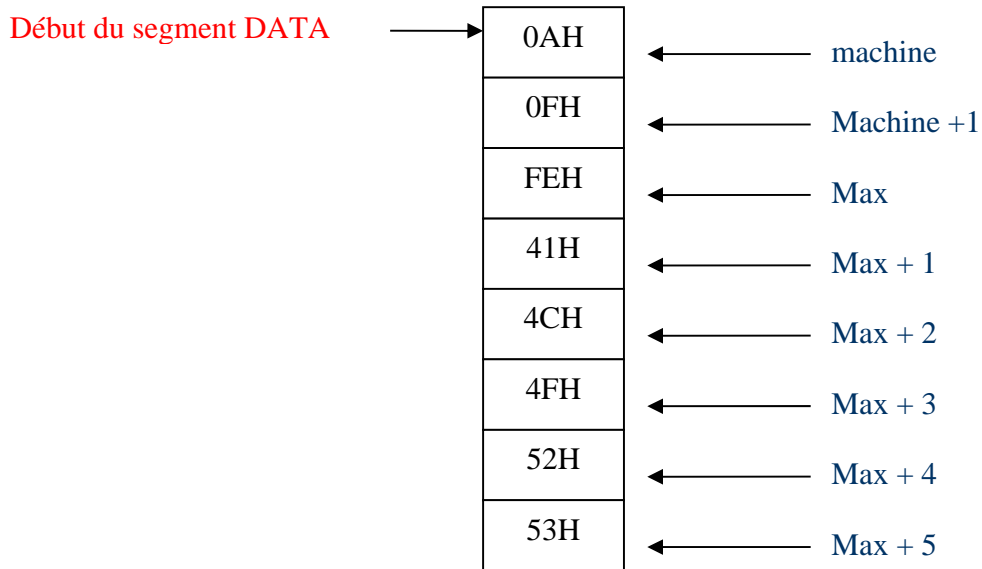


Fig.07. Contenu de la mémoire.

Si l'on veut écrire un caractère `X` à la place du `O` de `ALORS`, on pourra écrire :

```
MOV AL, 'X'
MOV Max+3, AL
```

Notons que `Max+3` est une constante (valeur connue au moment de l'assemblage): l'instruction générée par l'assembleur pour

```
MOV Max+3, AL      est      MOV [adr], AL .
```

Lorsque l'on veut déclarer un tableau de `n` cases, toutes initialisées à la même valeur, on utilise la directive **DUP** :

```
Tab  DB  100  dup (15)  ; 100 octets valant 15
Mat  DW   10  dup (?)   ; 10 mots de 16 bits non initialisés
```

3.2. Segment de Pile

Les piles offrent un moyen d'accéder à des données en mémoire principale, elles sont très utilisées pour stocker des valeurs temporaires. Une pile est une zone mémoire contiguë dont le registre SP contient l'adresse du sommet de pile. Deux opérations permettent de manipuler les données d'une pile :

- **PUSH registre** empile le contenu du registre dans la pile (sommet de pile).
- **POP registre** retire (dépile) la valeur en haut de la pile (sommet de pile) et la place dans le registre spécifié. La valeur dépilée par POP est la *dernière* valeur empilée ; c'est pourquoi on parle ici de pile LIFO.

Exemple: Transfert de AX vers BX en passant par la pile.

```
PUSH AX      ; Pile <- AX
POP  BX      ; BX <- Pile
```

La pile est souvent utilisée pour sauvegarder temporairement le contenu des registres, la pile peut conserver plusieurs valeurs:

Exemple:

```
                ; AX et BX contiennent des donnees a conserver
PUSH AX
PUSH BX
MOV  BX, Nb     ; on utilise AX et BX pour réaliser une addition
ADD  AX, BX
MOV  Nb, AX

POP  BX         ; recupère l'ancien BX
POP  AX         ; et l'ancien AX
```

La pile est stockée dans un segment séparé de la mémoire principale. Le processeur possède deux registres dédiés à la gestion de la pile, **SS et SP**. Le registre **SS** (*Stack Segment*) est un registre segment qui contient l'adresse du segment de pile courant (16 bits de poids fort de l'adresse). Il est normalement initialisé au début du programme et reste fixé par la suite. Le registre **SP** (*Stack Pointer*) contient le déplacement du sommet de la pile (16 bits de poids faible de son adresse).

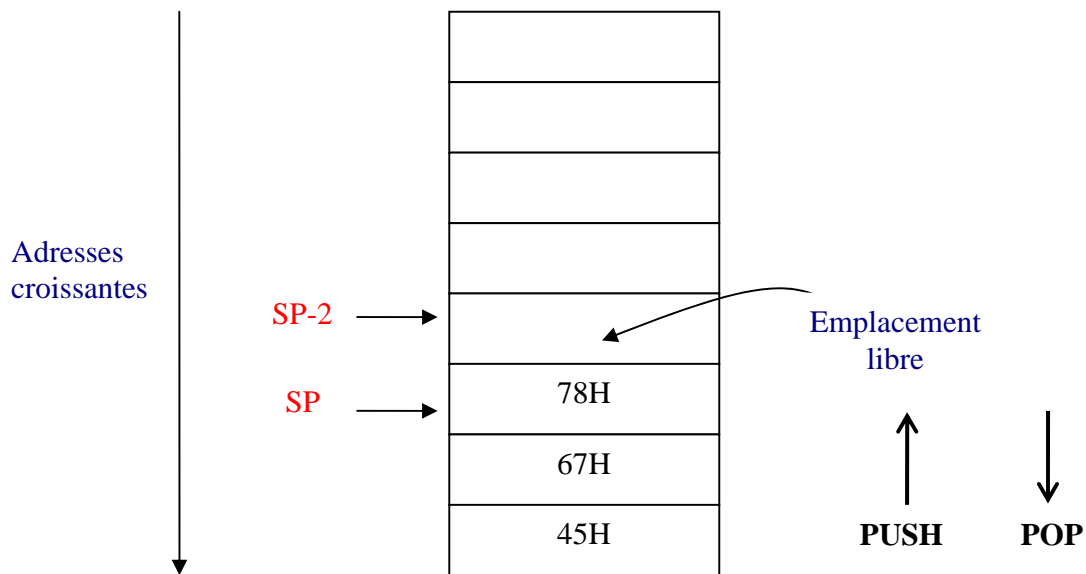


Fig.08. Fonctionnement d'une Pile.

L'instruction **PUSH** effectue les opérations suivantes (au début (pile vide), SP pointe "sous" la pile):

- $SP \leftarrow SP - 2$
- $[SP] \leftarrow$ valeur du registre 16 bits.

L'instruction **POP** effectue le travail inverse :

- registre destination $\leftarrow [SP]$
- $SP \leftarrow SP + 2$

Remarque : Si la pile est vide, POP va lire une valeur en dehors de l'espace pile, donc imprévisible.

Pour utiliser une pile en assembleur, il faut déclarer un segment de pile, et y réserver un espace suffisant. Ensuite, il est nécessaire d'initialiser les registres SS et SP pour pointer sous le sommet de la pile.

Exemple: déclaration d'une pile de 200 octets :

```

seg_pile    SEGMENT stack           ; mot clef stack car pile
DW 100     dup ( ?)                ; réserve l'espace
seg_pile    ENDS

```

4. Format des instructions

Une instruction en langage assembleur est devisée en quatre champs et doit être écrite sur une ligne de 132 caractères au maximum. On distingue :

3.1. Champ étiquette: Ce champ attribue une étiquette ou un nom à l'instruction ce qui permet de la distinguer. Elle pourrait être référencier par la suite en utilisant ce nom. Une étiquette comporte au maximum 31 caractères et se termine par un double point (:). Dans le cas d'une instruction et n'est pas nécessaire dans le cas d'une pseudo instruction. Les caractères autorisés pour représenter une étiquette sont :

- Les lettres de A..Z
- Les chiffres de 0..9
- Les caractères spéciaux : ?, ., @, _, \$.

Une étiquette commence toujours par une lettre alphabétique et ne doit pas utiliser un nom réservé tel que le nom d'un registre.

3.2. Champ code opération: Il contient un mnémonique qui définit l'opération à exécuter.

3.3. Champ opérande ou adresse: ce champ indique sur quoi porte l'opération mentionnée dans le champ code opération, ou à quel registre ou à quelle cellule mémoire l'opération fait référence.

3.4. Champ commentaire: ce champ est facultatif, il est utilisé pour apporter des informations sur l'instruction qui le précède.

Etiquette (Label)	opération	Opérandes	Commentaire
Facultatif (31 caractères maximum)	Obligatoire	Selon opération	Facultatif

Fig.08. Format d'une instruction en assembleur.

Remarques :

- Les champs sont séparés par un ou plusieurs espaces.
- Lorsque le champ opérande comprend plusieurs opérandes, elles sont séparées par une virgule, avec ou sans espaces.
- Un commentaire est obligatoirement précédé par un espace et par un point virgule le séparant du champ le précédent, la marque de sa fin est un retour à la ligne.

4. Les modes d'adressage du 80286

Le 80286 peut adresser 1 Mo en MC en mode réel, pointés par les registres sélecteurs de segments: registres de déplacement. Le nombre à 4 digits placé dans le registre sélecteur de segment

reçoit un digit supplémentaire, de plus faible poids, auquel on ajoute un déplacement sur 4 digits en hexadécimal. On appelle:

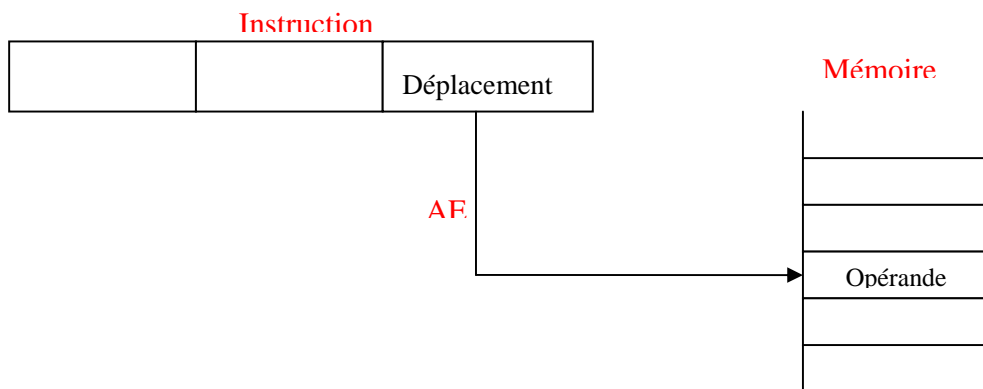
- Adresse absolue: l'adresse physique, réelle et complète, ainsi calculée.
- Offset: le déplacement dans le segment.
- Adresse effective (AE): c'est l'offset ci-dessus. Il s'agit d'un nombre sur 16 bits non signé.

4.1. Adressage immédiat: ne fait pas usage d'une adresse, l'opérande est directement inscrite dans l'instruction.

Exemple:

```
MOV DL, 5
ADD DL, 3
```

4.2. Adressage direct: l'instruction comporte l'adresse effective de l'opérande.

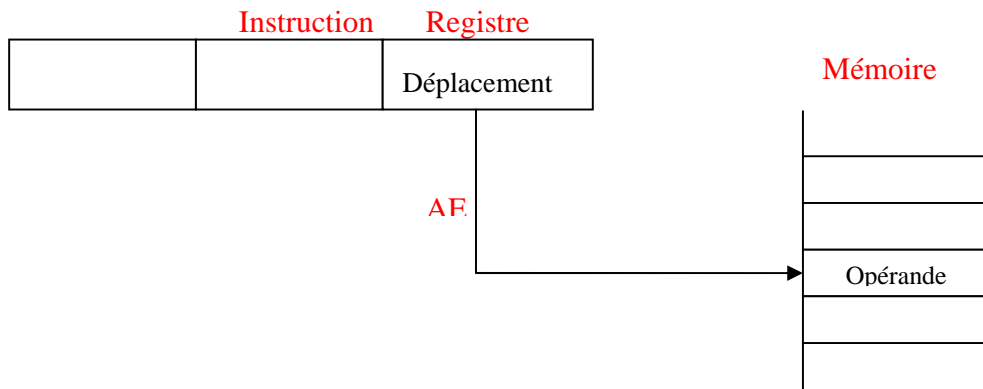


Exemple: MOV [adresse], AL ; les crochets indiquent qu'il s'agit d'une adresse et non pas de l'opérande.

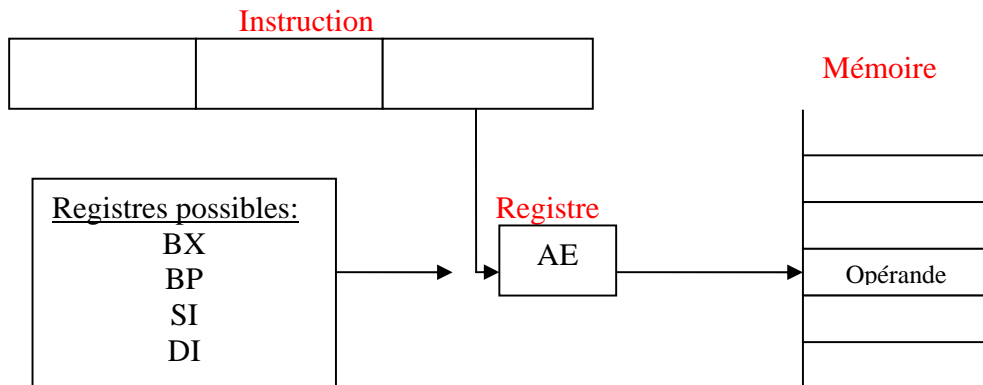
4.3. Adressage implicite: aucune adresse n'est indiquée dans le corps de l'instruction car l'instruction implique un registre implicitement.

Exemple: AAA ; ajustage ASCII pour l'addition, elle porte sur le registre AX.

4.4. Adressage par registre: l'instruction spécifie un registre qui contient l'opérande.



4.5. Adressage indirect par registre: l'instruction spécifie un registre qui contient l'adresse effective de l'opérande.



Exemple: MOV AX, [SI] ; SI contient l'AE et non pas l'opérande.

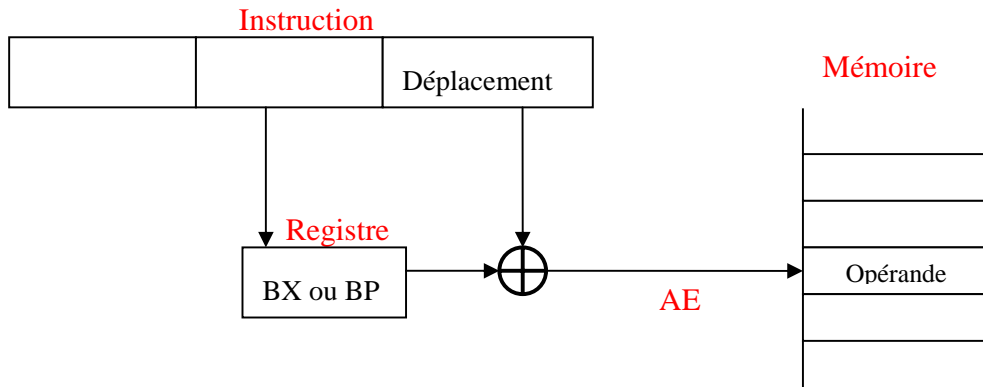
L'adressage indirect est très utile par exemple pour traiter des tableaux. L'adressage indirect utilise un des registres BX, BP, SI ou DI pour stocker l'adresse d'une donnée.

Exemple:

```
MOV AX, [130]     ; adressage direct
MOV AX, [BX]     ; adressage indirect
```

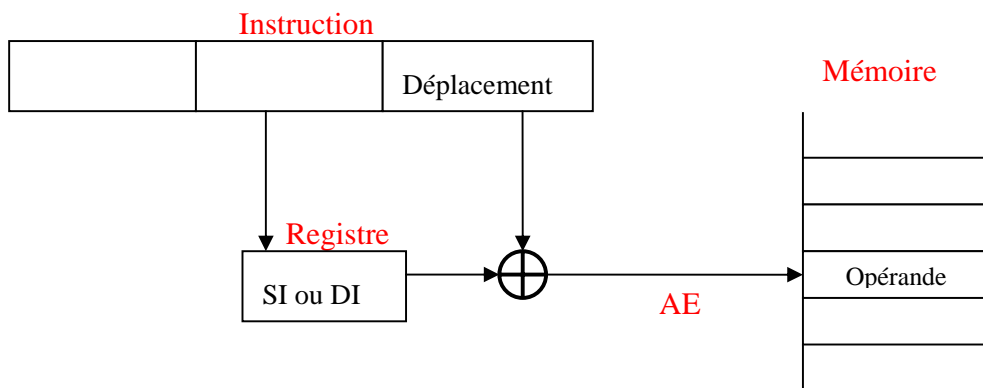
Ici, BX contient l'adresse de la donnée. L'avantage de cette technique est que l'on peut modifier l'adresse en BX, par exemple pour accéder à la case suivante d'un tableau. Avant d'utiliser un adressage indirect, il faut charger BX avec l'adresse d'une donnée.

4.6. Adressage par registre de base (relatif): c'est un mode d'adressage indirect. Avec l'adressage par registre de base BX ou BP, l'adresse effective AE résulte de la somme du contenu du registre BX ou BP et d'un déplacement inclus dans l'instruction.



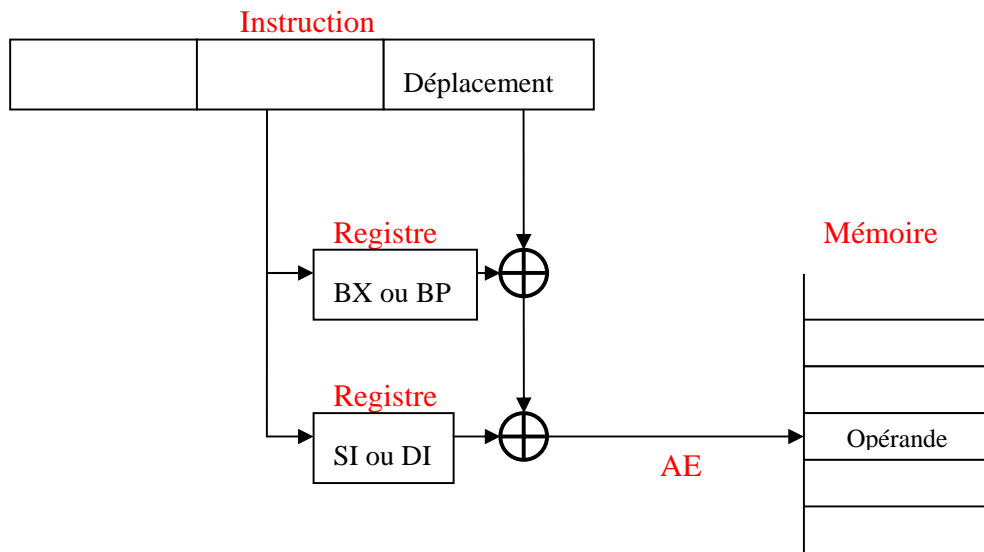
Exemple: `MOV AX, [BP+37H]` ; l'AE = contenu de BP + 37H.

4.7. Adressage indexé: ce type d'adressage est assuré via des registres d'index SI ou DI.



Exemple: `MOV AX, [DI+37H]` ; l'AE = contenu de DI + 37H.

4.8. adressage par index et base: dans cet adressage, on additionne le contenu d'un registre de base, d'un registre d'index et d'un déplacement.



4.9. Adressage de chaînes: ce type d'adressage fait implicitement appel aux index, avec SI pour la source et DI pour la destination.

Le tableau ci-dessous illustre une synthèse, définissant les registres de segments impliqués par les différents modes d'adressage.

Mode	Format de l'opérande	Registre de segment
Immédiat à registre A registre Implicite	Donnée Un registre Rien	Aucun
Immédiat à mémoire	Donnée	DS
Direct	Constante Label	DS DS
Indirect par registre	[BX] [BP] [DI] [SI]	DS SS DS DS
Par base	[BP + déplacement] [BX + déplacement]	SS DS
Indexé	[DI + déplacement] [SI + déplacement]	SS DS
Indexé par base	[BX + DI+ déplacement] [BX + SI+ déplacement] [BP + DI+ déplacement] [BP + SI+ déplacement]	DS DS SS SS
Chaînes -Source -Destination	[SI] [DI]	DS ES

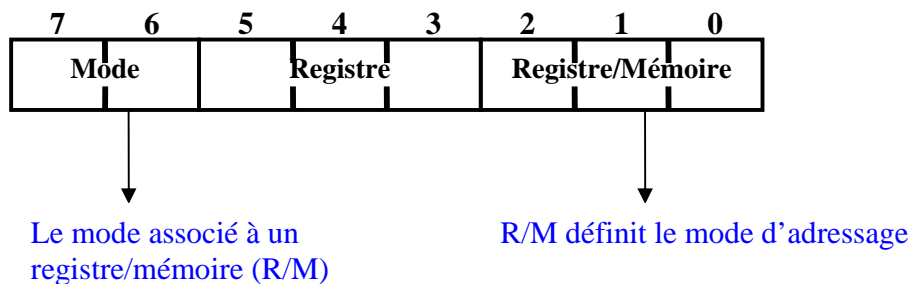
Registres de segments impliqués selon les modes d'adressage

Remarque: On peut spécifier dans une instruction et nommément le registre sélecteur de segment référencié. Par exemple, lors d'une adresse mémoire n'impliquant pas les chaînes, on peut désigner une cellule mémoire avec *ES* et *DI*. Dans ce cas et pour un adressage indirect via *DI*, on écrira : *ES:[DI]*.

5. Structure des instructions

Une instruction peut comporter 1 à 7 octets dont 1 à 2 octets sont utilisés pour le code opération :

- Le premier octet représente le code opération ;
- Le second octet du code opération, s'il existe, indique le mode d'adressage, le ou les registres concernés, ou encore le type d'instruction dans une famille, pour laquelle le premier octet est le même. Sa structure est la suivante :



- Les octets suivants définissent :
 - Les préfixes : changement de segment, répétition.
 - Les déplacements, notés DEP, sur 8 ou 16 bits, qui servent à calculer l'adresse effective AE.
 - La donnée, sur 8 ou 16 bits, en cas d'opérations immédiates.
 - Le contenu de CS ou de IP, lorsqu'on change de segment.

5.1. Les registres

Les trois bits utilisés pour représenter le registre sont éventuellement complétés par un quatrième bit noté *W* qui est représenté au niveau du premier octet de l'instruction. Avec *W* =1, on travail sur 16 bits ; avec *W* =0, on est sur 08 bits.

Registres				Segment	
16 bits (w=1)		8 bits (w=0)			
0 0 0	AX	0 0 0	AL	0 0	ES
0 0 1	CX	0 0 1	CL	0 1	CS
0 1 0	DX	0 1 0	DL	1 0	SS
0 1 1	BX	0 1 1	BL	1 1	DS
1 0 0	SP	1 0 0	AH		
1 0 1	BP	1 0 1	CH		
1 1 0	SI	1 1 0	DH		
1 1 1	DI	1 1 1	BH		

5.2. Les modes d'adressage

Le Mode associé à R/M est codé sur 2 bits, les 4 combinaisons possibles sont :

Mode	Déplacement
0 0	- DEP = 0, pas de déplacement. -Si R/M = 110 alors AE= DEP-fort :DEP-faible (déplacement sur 16 bits)
0 1	DEP = DEP-faible (DEP sur 8 bits) avec signe étendu sur l'octet de poids fort
1 0	AE= DEP-fort :DEP-faible
1 1	R/M est traité comme un champ registre

* DEP-fort : Octet de poids fort du déplacement.

DEP-faible : Octet de poids faible du déplacement.

Le calcul de l'adresse effective AE est défini par le champ R/M et obéit aux règles suivantes :

R/M	Adresse de l'opérande
0 0 0	(BX) + (SI) + DEPL
0 0 1	(BX) + (DI) + DEPL
0 1 0	(BP) + (SI) + DEPL
0 1 1	(BP) + (DI) + DEPL
1 0 0	(SI) + DEPL
1 0 1	(DI) + DEPL
1 1 0	(BP) + DEPL
1 1 1	(BX) + DEPL

Il résulte que l'adresse effective dépend du mode et du R/M et est calculée comme suit :

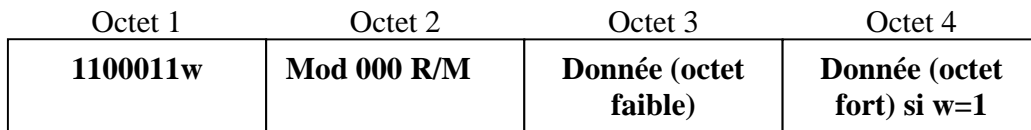
R/M	Adresse effective en « Mode »		
	Mode = 00	+, pour mode = 01	+, pour mode = 10
000	(BX) + (SI)	Un déplacement sur 8 bits	Un déplacement sur 16 bits
001	(BX) + (DI)		
010	(BP) + (SI)		
011	(BP) + (DI)		
100	(SI)		
101	(DI)		
110	DEP (16 bits)		
111	(BX)		

5.3. Codes opérations

Le code opération est décrit par :

- Un bit noté **W** qui indique la longueur de l'opérande, **W=0** pour un opérande octet et **W=1** pour mot.
- La lettre **d** est utilisée pour indiquer le destinataire, **d=1** pour un registre éventuellement le premier (si deux registres interviennent). Son code est fourni par **REG**.

Ainsi une instruction sur 4 octets peut avoir cette allure :



↑
Code opération
MOV avec opérande
immédiat

Exemples :

Instruction	1 ^{er} Octet	2 ^{ème} Octet	3 ^{ème} Octet	4 ^{ème} Octet
ADD - Opérande en registre ou mémoire avec opérande en registre - Opérande immédiat avec opérande en registre ou mémoire - opérande immédiat à accumulateur AX ou AL	000000dw 100000sw 0000010w	Mode Reg R/M Mode 000 R/M Donnée	Donnée Donnée si w=1	Donnée si w=1
AND - Opérande en registre ou mémoire avec opérande en registre - Opérande immédiat avec opérande en registre ou mémoire - opérande immédiat à accumulateur AX ou AL	001000dw 1000000w 0010010w	Mode Reg R/M Mode 100 R/M Donnée	Donnée Donnée si w=1	Donnée si w=1
MOV - mémoire ou registre de /vers registre - immédiat à mémoire ou registre - immédiat à registre - mémoire à AL ou AX - AL ou AX à mémoire - mémoire ou registre à registre de segment - registre de segment à mémoire ou registre	100010dw 1100011w 1011w Reg 1010000w 1010001w 10001110 10001100	Mode Reg R/M Mode 000 R/M Donnée Adresse basse Adresse basse Mode Reg R/M Mode Reg R/M	Donnée Donnée si w=1 Adresse haute Adresse haute	Donnée si w=1

6. Instructions

6.1. Instructions de mouvements

Usage	Nom	Fonction	Syntaxe
Général	MOV	Transfert de données d'octets ou de mots d'une source à une destination.	MOV Destination, Source <i>Modes et exemples :</i> - Accumulateur à mémoire : MOV mém, AX - Mémoire à accumulateur : MOV AX, mém - Mémoire ou registre à registre de segment : MOV DS, AX - Registre de segment à mémoire ou registre : MOV AX, DS - Registre à registre : MOV CX, DX - Mémoire ou registre à registre : MOV CX, mém - Registre à Mémoire ou registre : MOV [BX][SI], DX - Donnée immédiate à registre : MOV AX, 37 - Donnée immédiate à mémoire ou registre:MOV [BX][SI],37
	MOVS MOVSB MOVSW	Déplacement d'un octet (Byte) ou d'un mot de 16 bits (Word) d'une chaîne pointé par DS:DI. Après le transfert, SI et DI sont incrémentés ou décrémentés selon la situation selon l'indicateur de direction DF, du nombre d'octets transférés. Le préfixe REP sert à répéter l'opération.	MOVS Destination, Source MOVSB MOVSW <i>Exemple :</i> - MOVS Recu, Emis
	PUSH	Sauvegarde dans la pile : Décrémente le pointeur de pile SP de 2 puis transfère le mot source dans la pile.	PUSH Source <i>Modes et exemples :</i> - De registre 16 bits : PUSH AX - De registre de segment : PUSH ES - D'une mémoire ou d'un registre : PUSH mém
	PUSH	Sauvegarde dans la pile de l'octet ou le mot immédiat contenu dans l'instruction.	PUSH Mot ou Octet <i>Exemple :</i> PUSH 1234
	PUSHA	Sauvegarde de tous les registres généraux dans l'ordre : AX, CX, DX, BX, SP d'origine, BP, SI et DI.	PUSHA
	POP	Dépille le mot au sommet de la pile dans le registre ou la cellule mémoire indiquée par destination. Après le POP, le pointeur SP est incrémenté de 2.	POP Destination <i>Modes et exemples :</i> - En registre 16 bits : POP AX - En registre de segment : POP DS - En mémoire ou registre : POP mém
	POPA	Extraction de tous les registres de la pile. Recharge successivement DI, SI, BP, SP, BX, CX, DX et AX.	POPA

Usage	Nom	Fonction	Syntaxe
Général	XCHG	Echange le contenu des deux opérandes (en sont exclus les registres de segments).	XCHG Destination, Source <u>Modes et exemples :</u> - Accumulateur et registre : XCHG AX, BX - Registre et mémoire ou registre : XCHG AX, mém
	IN	Entrée d'un octet ou d'un mot : Transfère l'opérande arrivant par un port d'entrée dans AL ou AX. Le port est spécifié dans l'instruction, sur 8 bits (le port est désigné de 0 à 256) ou en indirect via le registre DX (64 K ports maximum).	IN Accumulateur AL ou AX, Port IN Accumulateur AL ou AX, DX <u>Exemples :</u> IN AL, 53H IN AX, DX
Entrées/ Sorties	INS INSB INSW	Permet le chargement d'un mot (INSW) ou d'un octet (INSB) dans une chaîne de caractères. DX contient l'adresse du port et ES:DI pointe la mémoire.	INS Chaîne, Registre du port INSB INSW <u>Exemples :</u> INS Hello, DX INSW
	OUT	Transfère l'octet dans AL, ou le mot dans AX, vers le port de sortie désigné en mode immédiat sur un octet ou en mode indirect par le registre DX.	OUT Port, Accumulateur AL ou AX OUT DX, Accumulateur AL ou AX <u>Exemples :</u> OUT 37, AX OUT DX, AL
	OUTS OUTSB OUTSW	Sortie d'une chaîne pointée par DS:SI vers un port d'adresse spécifiée par DX (ou d'un octet, ou d'un mot).	OUTS Registre du port , Chaîne OUTSB OUTSW <u>Exemples :</u> OUTS DX, Hello OUTSW
Adresses	LEA	Transfère de l'adresse effective de la "source" dans le registre destination. La source doit être en mémoire.	LEA Destination, Source <u>Exemple:</u> LEA BX, Table[SI]
	LDS	Transfère un pointeur de 32 bits contenant l'adresse d'un segment et son décalage de la source dans : DS pour le segment, et un registre de 16 bits indiqué par destination pour le décalage.	LDS Destination, Source <u>Exemple:</u> LEA SI, [BX]
	LES	Transfère un pointeur de 32 bits contenant l'adresse d'un segment et son décalage de la source dans : ES pour le segment, et un registre de 16 bits indiqué par destination pour le décalage.	LDS Destination, Source <u>Exemple:</u> LEA DI, [BX]

Usage	Nom	Fonction	Syntaxe								
Indicateurs	LAHF	Transfère les indicateurs SF, ZF, AF, PF et CF dans le registre AH selon le schéma suivant où X marque une position indéfinie.	LAHF <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>SF</td><td>ZF</td><td>X</td><td>AF</td><td>X</td><td>PF</td><td>X</td><td>CF</td> </tr> </table>	SF	ZF	X	AF	X	PF	X	CF
	SF	ZF	X	AF	X	PF	X	CF			
	SAHF	Transfère les cinq indicateurs SF, ZF, AF, PF et CF, contenus dans AH vers le registre des indicateurs.	SAHF								
	PUSHF	Sauvegarde le registre des indicateurs dans la pile.	PUSHF								
POPF	Restauration des indicateurs extraits de la pile. Transfère le mot au sommet de pile dans le registre des indicateurs. Puis SP est incrémenté de 2.	POPF									

Instructions arithmétiques

Usage	Nom	Fonction	Syntaxe
Addition	ADD	Réalise une addition, le résultat est dans la destination.	ADD Destination, Source <u>Modes et exemples :</u> - Registre ou mémoire avec registre: ADD CX, Nombre ADD BL, CL - Accumulateur avec opérande immédiat : ADD BL, 17
	ADC	Addition avec retenue : elle exécute une addition en prenant en compte une retenue précédente.	ADC Destination, Source <u>Modes et exemples :</u> - Registre ou mémoire avec registre: ADC CX, Nombre ADC BL, CL - Accumulateur avec opérande immédiat : ADC AX, 23
	INC	Incrémente l'opérande de 1.	INC Destination <u>Exemples :</u> INC DI INC AX INC mém
	AAA	Ajustage ASCII pour l'addition : suit une addition DCB. On suppose qu'on vient d'effectuer une addition portant sur des nombres décimaux en codes ASCII (codes de 30 à 39). Dans ce cas, l'ajustage porte sur le contenu du résultat dans AL avec la séquence : 1- Si (AL) contient un nombre de 0 à 9 et si	AAA <u>Exemples :</u> ADD AL, 32H AAA

	<p>l'indicateur AF=0, on passe au 3^{ème} pas.</p> <p>2- Si (AL) contient un nombre hexa de A à F ou si AF=1, AAA ajoute 6 à AL, ajoute 1 à AH et remet AF à 0.</p> <p>3- Mise à 0 du quartet de poids fort de AL.</p>	
	<p>DAA</p> <p>Ajustage décimal pour l'addition : utilisé pour adapter le résultat en DCB compacté dans AL :</p> <p>1- Si le quartet faible de AL contient A à F ou si AF=1, ajoute 6 à AL et met AF à 0.</p> <p>2- Si le quartet fort de AL contient A à F ou si CF=1, ajoute 6 à AL et met CF à 0.</p>	<p>DAA</p> <p><u>Exemples :</u> ADD AL, 37 DAA</p>
Soustraction	<p>SUB</p> <p>Exécute l'opération (destination = destination – source).</p>	<p>SUB Destination, Source</p> <p><u>Modes et exemples :</u> -Registre ou mémoire avec registre: SUB AX, Memoire [SI] -Accumulateur avec opérande immédiat : SUB AL, 17 -Mémoire ou registre avec opérande immédiat : SUB SI, 33</p>
	<p>SBB</p> <p>Soustraction avec retenue: exécute l'opération (destination = destination – source – retenue précédente). La retenue précédente est conservée dans l'indicateur CF.</p>	<p>SBB Destination, Source</p> <p><u>Modes et exemples :</u> -Registre ou mémoire avec registre: SBB DX, Mem[DI] -Accumulateur avec opérande immédiat : SUB AX, 37 -Mémoire ou registre avec opérande immédiat: SBB CL, 33</p>
	<p>DEC</p> <p>Décrémente la destination de 1.</p>	<p>DEC Destination</p> <p><u>Modes et exemples :</u> -Registre sur 8 ou 16 bits: DEC AX -Mémoire : DEC Nombre</p>
	<p>NEG</p> <p>Formation du complément à 2: exécute l'opération (destination= 0 – destination). L'opérande est dans un registre ou une cellule mémoire.</p>	<p>NEG Destination</p> <p><u>Exemples :</u> NEG AL</p>
	<p>CMP</p> <p>Comparaison de deux opérandes. Elle effectue l'opération (Destination – source) sans modifier le contenu des opérandes mais positionne les indicateurs.</p>	<p>CMP Destination, Source</p> <p><u>Modes et exemples :</u> -Registre ou mémoire avec registre: CMP BX, CX -Accumulateur avec opérande immédiat : CMP AX, 17 -Mémoire ou registre avec opérande immédiat: CMP Nb,34</p>

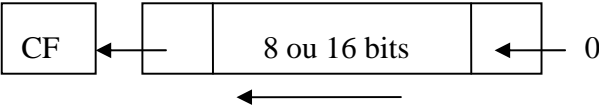
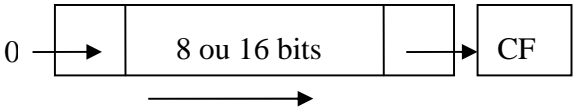
Usage	Nom	Fonction	Syntaxe
Soustraction	AAS	<p>Ajustage ASCII pour la soustraction : Exécute une correction ASCII sur le résultat d'une soustraction dont les termes sont en DCB étendu. Et porte sur AL selon la séquence :</p> <ol style="list-style-type: none"> 1- Si le quartet faible de AL contient 0 à 9 ou si AF=0, Passer à 3. 2- Si le quartet faible de AL contient A à F ou si AF=1, faire (AL)-6 puis (AH)-1, et AF est mis à 1. 3. Le quartet fort de AL est mis à zéro. 4. L'indicateur CF est positionné comme AF. 	<p>AAS</p> <p><u>Exemples :</u></p> <p>SUB AL,34H</p> <p>AAS</p>
	DAS	<p>Ajustage décimal pour la soustraction : Après une soustraction en DCB compacté, le résultat étant dans AL, corrige AL s'il y a lieu pour obtenir du DCB compacté :</p> <ol style="list-style-type: none"> 1- Si le quartet faible de AL contient A à F ou si AF=0, faire (AL)-6 et met AF à 1. 2- Si le quartet fort de AL contient A à F ou si CF=1, faire (AL)-6, et CF=1. 	<p>DAS</p> <p><u>Exemples :</u></p> <p>SUB AL,37</p> <p>DAS</p>
Multiplication	MUL	<p>Multiplication du contenu de l'accumulateur par le multiplicateur contenu dans un registre ou une cellule mémoire. Le multiplicande est l'octet dans AL ou deux octets dans AX, le multiplicateur peut être un octet ou un mot. Le produit va dans AX (opération sur un octet) ou dans DX:AX (opération sur 16 bits). Si la moitié de poids fort du produit est nulle, OF et CF sont mis à 0, sinon ils sont mis à 1.</p>	<p>MUL Multiplicateur</p> <p><u>Exemples :</u></p> <p>MUL CL</p> <p>MUL CX</p>
	IMUL	<p>Multiplication entière signé : Le multiplicande dans AL (8 bits) ou dans AX (16 bits) est multiplié par le multiplicateur.</p>	<p>IMUL Multiplicateur</p> <p><u>Exemples :</u></p>

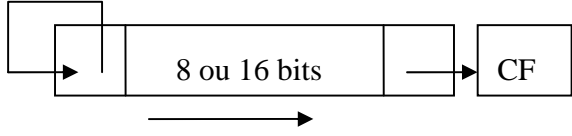
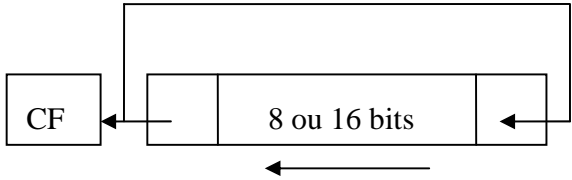
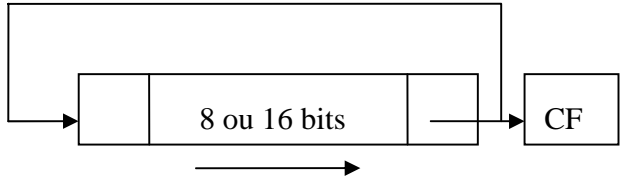
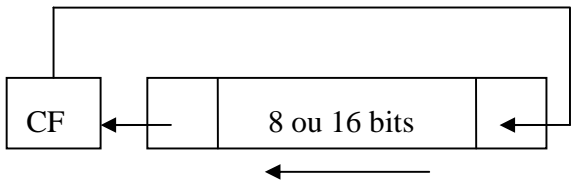
	<p>Le résultat va dans AX (pour un multiplicande de 8 bits) ou dans DX:AX (pour un multiplicande de 16 bits). Les indicateurs OF et CF sont mis à 1 si la moitié du résultat, partie forte n'est pas l'extension du signe de l'autre moitié à 16 (ou 32 bits).</p>	<p>IMUL BX IMUL [BX] IMUL Nombre</p>
AAM	<p>Ajustage ASCII pour la multiplication: Après une multiplication en DCB étendu. AX contenant le résultat en DCB, AAM procure un résultat en DCB dans AL. La séquence déclenchée est:</p> <ol style="list-style-type: none"> (AL) est divisé par 0AH. Le quotient va dans AH et le reste dans AL. Les indicateurs sont positionnés. 	<p>AAM <i>Exemples :</i> MUL AL,BL AAM</p>
DIV	<p>Exécute une division non signée d'un dividende de 16 bits contenu dans AX, ou d'un nombre de 32 bits contenu dans DX:AX par le diviseur de 8 ou 16 bits. Le quotient va dans AL (dividende de 16 bits) ou AX (dividende de 32 bits) et le reste dans AH ou DX.</p> <p>Si la valeur maximale du quotient, de FFH (dividende de 16 bits) ou FFFFH (dividende de 32 bits) est dépassée. Le microprocesseur déclenche une interruption de type 0, le quotient et le reste étant alors indéfinis ; il en va de même lors d'une division par 0. Le diviseur doit se trouver dans un registre ou dans une cellule mémoire.</p>	<p>DIV Diviseur <i>Exemples :</i> DIV CL DIV CX DIV Nombre</p>
IDIV	<p>Division entière signée : divise un dividende de 16 bits contenu dans AX, ou d'un nombre de 32 bits contenu dans DX:AX par un diviseur. Le quotient va dans AL (dividende de 16 bits) ou AX (dividende de 32 bits) et le reste dans AH ou DX.</p> <p>La valeur maximale absolue du quotient signé est de 7FH ou 7FFFH. Si elle est dépassée ou en cas de division par 0, Le</p>	<p>IDIV Diviseur <i>Exemples :</i> IDIV CL IDIV CX IDIV Nombre</p>

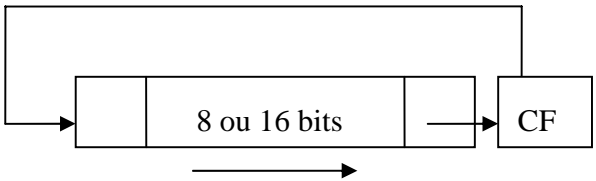
Division		microprocesseur déclenche une interruption de type 0. IDIV tronque les quotients non entiers et renvoie un reste avec le même signe que le dividende par convention. Le dividende et le diviseur sont considérés comme signés. Lorsque le dividende et le diviseur ont la même longueur, le dividende doit d'abord être converti en 16 ou 32 bits avec CBW ou CWD.	
	AAD	Ajustage ASCII pour la division: Précède une division; AX est supposé contenir DCB étendu. Dans ce cas, AAD convertit (AL) en un opérande binaire selon la séquence: 1. Multiplication de (AH) par 0AH. 2. (AH) est ajouté à (AL). 3. (AH) est remis à zéro. 4. Positionnement des indicateurs.	AAD <u>Exemples :</u> Si (AX) contient 0604 en hexadécimal, après AAD, il contiendra 0040H et les indicateurs AF, CF et OF seront indéfinis alors que PF=0, SF=0 et ZF=0.
	CBW	Conversion d'octet en mot de 16 bits : exécute une expansion du signe de (AL) dans (AH), ce signe étant dupliqué sur les 8 bits de AH. Donc si (AL)<80H alors (AH)=0 et si (AL)≥80H alors (AH)=FFH.	CBW <u>Exemples :</u> Si AL contient 1000 0000, après CBW, AX contiendra 1111 1111 1000 0000.
	CWD	Conversion de mot de 16 bits en double mot de 32 bits : le bit de signe du registre AX est étendu sur les 16 bits de DX. Donc si (AX)<8000H alors (DX)=0 et si (AX)≥8000H alors (DX)=FFFFH.	CWD <u>Exemples :</u> Si AX contient F000H, après CWD, DX contiendra FFFFH.

Traitement de bits : des instructions qui exécutent des fonctions logiques, des décalages et des rotations.

Usage	Nom	Fonction	Syntaxe
Logique	NOT	Inverse les bits, formant le complément à 1 de l'opérande, en exécutant l'opération (0FF-opérande) pour un octet, ou (0FFFF-opérande) pour un mot de 2 octets. L'opérande est dans	NOT Opérande <u>Exemples :</u> NOT AL

		un registre ou en mémoire.	
	AND	Exécute la fonction logique ET entre deux opérandes. Le résultat est dans la destination.	<p>AND Destination, Source</p> <p><u>Modes et exemples :</u></p> <ul style="list-style-type: none"> -Registre ou mémoire avec registre: AND CL, BL AND AX, Nombre -Accumulateur avec opérande immédiat : AND AX, 37H
	OR	Exécute la fonction logique OU inclusive entre deux opérandes. Le résultat est rangé dans la destination.	<p>OU Destination, Source</p> <p><u>Modes et exemples :</u></p> <ul style="list-style-type: none"> -Registre ou mémoire avec registre: OR CL, BL OR AX, Nombre -Accumulateur avec opérande immédiat : OR AX, 1234 -Registre ou mémoire avec opérande immédiat : OR BX, 1234
	XOR	Exécute un OU exclusif entre deux opérandes. Le résultat est rangé dans la destination. Des bits identiques fournissent un 0, des bits différents fournissent un 1.	<p>XOR Destination, Source</p> <p><u>Modes et exemples :</u></p> <ul style="list-style-type: none"> -Registre ou mémoire avec registre: XOR AX, Nombre -Registre et mémoire ou registre: XOR Nombre, AX -Accumulateur avec opérande immédiat : XOR AX, 1234 -Registre ou mémoire avec opérande immédiat: XOR Nb, 12
	TEST	Compare deux opérandes bit par bit sans fournir de résultat mais positionne les indicateurs. La comparaison est assurée par un ET logique.	<p>TEST Destination, Source</p> <p><u>Modes et exemples :</u></p> <ul style="list-style-type: none"> -Registre ou mémoire avec registre: TEST BX, Nombre - Accumulateur avec opérande immédiat : TEST AX, 37 -Registre ou mémoire avec opérande immédiat : TEST Nombre, 34
Décalages	SHL SAL	Décalage arithmétique ou logique à gauche ; le nombre de décalages est spécifié par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur. Chaque décalage équivaut à une multiplication par 2.	 <p>SAL Destination, Compte SHL Destination, Compte</p> <p><u>Exemples :</u> SAL AX, 3 SAL AX, CL SAL Nombre, CL</p>
	SHR	Décalage arithmétique à droite; le nombre de décalages est spécifié par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.	 <p>SHR Destination, Compte</p> <p><u>Exemples :</u> SHR AX, 3</p>

		SHR AX, CL SHR Nombre, CL
	SAR	<p>Décalage arithmétique à droite; le bit du signe est conservé. le nombre de décalages est spécifié par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.</p>  <p>SAR Destination, Compte</p> <p><i>Exemples :</i> SAR AX, 3 SAR AX, CL SAR Nombre, CL</p>
Rotations	ROL	<p>Rotation à gauche, le nombre de rotations est spécifié par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.</p>  <p>ROL Destination, Compte</p> <p><i>Exemples :</i> ROL AX, 3 ROL AX, CL ROL Nombre, CL</p>
	ROR	<p>Rotation à droite, le nombre de rotations est spécifié par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.</p>  <p>ROR Destination, Compte</p> <p><i>Exemples :</i> ROR AX, 3 ROR AX, CL ROR Nombre, CL</p>
	RCL	<p>Rotation à gauche via l'indicateur de retenue CF : exécute le nombre par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.</p>  <p>RCL Destination, Compte</p>

		<p><u>Exemples :</u> RCL AX, 3 RCL AX, CL RCL Nombre, CL</p>
RCR	<p>Rotation à droite via l'indicateur de retenue CF : exécute le nombre par « compte », de 1 à 31, ou alors par CL, lequel agit en compteur.</p>	 <p>RCR Destination, Compte</p> <p><u>Exemples :</u> RCR AX, 3 RCR AX, CL RCR Nombre, CL</p>

Commandes de Transferts : Instructions de branchement, de boucles et d'interruptions.

Usage	Nom	Fonction	Syntaxe
Branchements inconditionnels	CALL	<p>Appel de procédure ; l'appel de procédure déclenche les :</p> <ol style="list-style-type: none"> 1- Rangement des registres CS et IP dans la pile. 2- Chargement des 2^{ème} et 3^{ème} Octets de l'instruction dans IP. 3- Chargement des 4^{ème} et 5^{ème} Octets de l'instruction (pour un appel inter-segments) dans CS. <p>L'exécution se produit à partir de cette nouvelle adresse (CS :IP).</p>	<p>CALL Cible</p> <p><u>Modes :</u></p> <ul style="list-style-type: none"> -Direct intra-segment ou intra-groupe (CS ne change pas) -Direct inter-segment (CS change). -Direct intra-segment (par registre ou mémoire, CS ne change pas) -Direct inter-segment (par registre ou mémoire, CS change). <p><u>Exemples :</u></p> <p>CALL Calcul</p> <p>CALL BX</p>
	RET	<p>Retour de procédure ; termine une procédure et renvoie à l'instruction qui suivait l'instruction CALL. Le retour peut-être (NEAR) court ou long (FAR) d'où l'importance des bonnes déclarations. Avec un RET NEAR, l'adresse du déplacement est</p>	<p>RET</p> <p>RET Nombre à ajouter à SP</p> <p>SP est incrémenté de 2 après un RET NEAR, et de 4 après un RET FAR. Il s'y ajoute éventuellement la constante mentionnée.</p> <p><u>Modes :</u></p> <ul style="list-style-type: none"> - Intra-segment : RET

		automatiquement déchargé du sommet de la pile et restaure IP ; avec un RET FAR ; c'est IP puis CS qui sont restaurés. En outre le RET peut se voir associer à un nombre immédiat qui sera ajouté au pointeur de la pile SP, afin d'abandonner des paramètres de la pile.	<ul style="list-style-type: none"> - Intra-segment et addition à SP : RET 4 - Inter-segment : RET - Inter-segment et addition à SP : RET 4
	JMP	Saut, Branchement inconditionnel à l'adresse du segment dans CS, de +127 à -128 ou de 32767 à -32768 en intra-segment, ou branchement en inter-segment.	<p>JMP Cible</p> <p><u>Modes :</u></p> <ul style="list-style-type: none"> -Direct intra-segment ou intra-groupe (-32768 à +32767) -Direct intra-segment court (-128 à +127) - inter-segment direct. -Inter-segment indirect. -Direct intra-segment (par registre ou mémoire, CS ne change pas) - inter-segment ou intra-segment indirect, par registre ou mémoire <p><u>Exemples :</u></p> <p>JMP Calcul_NEAR JMP SI JMP [BP][DI] Déplacement</p>
Branchements conditionnels (arithmétique non signée)	JA JNBE	Saut si supérieur/ si non inférieur ou égal ; branchement relatif dans un segment, de -128 à +127 à partir de l'instruction qui suit JA/ JNBE. Le branchement est exécutée si CF=ZF=0.	<p>JA Cible</p> <p>JNBE Cible</p>
	JAE JNB	Saut si supérieur ou égal/ si non inférieur; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JAE/ JNB. Le branchement est exécutée si CF=0.	<p>JAE Cible</p> <p>JNB Cible</p>
	JB JNAE	Saut si inférieur/ si non supérieur ni égal; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JB/ JNAE. Le branchement est exécutée si CF=1.	<p>JB Cible</p> <p>JNAE Cible</p>
	JBE JNA	Saut si inférieur ou égal/ si non supérieur; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JBE/	<p>JBE Cible</p> <p>JNA Cible</p>

		JNA. Le branchement est exécutée si CF ou ZF=1.	
Branchements conditionnels (arithmétique signée)	JG JNLE	Saut si plus grand/ si non inférieur ou égal ; branchement relatif dans un segment, de -128 à +127 à partir de l'instruction qui suit JG/ JNLE. Le branchement est exécutée si ZF=0 et si SF=OF.	JG Cible JNLE Cible
	JGE JNL	Saut si plus grand ou égal/ si non inférieur ; branchement relatif dans un segment, de -128 à +127 à partir de l'instruction qui suit JGE/ JNL. Le branchement est exécutée si SF=OF.	JGE Cible JNL Cible
	JL JNGE	Saut si moins que / si non supérieur ou égal ; branchement relatif dans un segment, de -128 à +127 à partir de l'instruction qui suit JL/ JNGE. Le branchement est exécutée si si SF≠OF.	JL Cible JNGE Cible
	JLE JNG	Saut si inférieur ou égal/ si non supérieur ; branchement relatif dans un segment, de -128 à +127 à partir de l'instruction qui suit JLE/ JNG. Le branchement est exécutée si ZF=1 ou si SF≠OF.	JLE Cible JNG Cible
Branchements conditionnels (Indicateurs)	JC	Saut si indicateur de retenue CF=1; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JB/ JNAE.	JC Cible
	JE JZ	Saut si égal/ si résultat est nul; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JE/ JZ. Le branchement est exécutée si ZF=1.	JE Cible JZ Cible
	JNC	Saut pas de retenue; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JNC. Le branchement a lieu si indicateur de retenue CF=0.	JNC Cible
	JNE	Saut si pas égal/ si résultat	JNE Cible

JNZ	est non nul; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JNE/JNZ. Le branchement est exécutée si ZF=0.	JNZ Cible	
JNO	Saut pas de dépassement; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JNO. Le branchement a lieu si indicateur de retenue OF=0.	JNC Cible	
JNP JPO	Saut si pas de parité/ si parité impaire; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JNP/JPO. Le branchement est exécutée si PF=0.	JNP Cible JPO Cible	
JNS	Saut si pas de signe/ si positif; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JNS. Le branchement est exécutée si SF=0.	JNS Cible	
JO	Saut sur dépassement; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JO. Le branchement a lieu si indicateur de retenue OF=1.	JO Cible	
JP JPE	Saut si de parité/ si parité paire; branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JP/JPE. Le branchement est exécutée si PF=1.	JP Cible JPE Cible	
JS	Saut si sur signe (résultat négatif); branchement relatif dans un segment, de (-128 à +127) à partir de l'instruction qui suit JS. Le branchement est exécutée si SF=1.	JS Cible	
Boucles	LOOP	Boucler jusqu'à exécution du compte ; décrémente le registre CX compteur de 1 puis transfère le contrôle à l'opérande cible si CX n'est pas égal à 0. en mode relatif	LOOP Cible

		de (-128 à +127).	
	LOOPE LOOPZ	Boucler si égal/ si zéro; boucle conditionnelle a lieu si ZF=1 et si le registre compteur CX n'est pas égal à 0. ces deux instructions agissent comme LOOP.	LOOPE Cible LOOPZ Cible
	LOOPNE LOOPNZ	Boucler si non égal/ si non zéro; c'est la condition inverse de LOOPE/LOOPZ : la boucle conditionnelle a lieu si ZF=0 et si le registre compteur CX n'est pas égal à 0. ces deux instructions agissent comme LOOP.	LOOPNE Cible LOOPNZ Cible
	JCXZ	Saut si CX=0 ; Branchement relatif de (-128 à +127) par rapport à l'instruction qui suit JCXZ.	JCXZ Cible
Interruptions	INT	<p>Appelle d'une interruption vectorisée en :</p> <ol style="list-style-type: none"> 1-sauvegardant les indicateurs dans la pile. 2- Mettant IF et TF à 0. 3- Sauvegardant CS dans la pile. 4- Chargeant le registre de segment avec le segment de l'interruption. 5- Sauvegardant IP dans la pile. 6- Chargeant le décalage de l'interruption dans la pile. <p>Pour la forme de INT sur 1 octet, l'interruption est de type 3.</p>	<p>INT Type de l'interruption</p> <p><u>Exemples :</u></p> <p>INT 21H</p>
	INTO	<p>Interruption si dépassement; si l'indicateur de dépassement OF=0 ; équivaut à un NOP, si OF=1 exécute une interruption de type 4 en :</p> <ol style="list-style-type: none"> 1-sauvegardant les indicateurs dans la pile. 2- Mettant IF et TF à 0. 3- Sauvegardant CS dans la pile. 4- Chargeant le registre de segment avec le segment lu en 00012H. 	INTO

	<p>5- Sauvegardant IP dans la pile.</p> <p>6- Chargeant le décalage lu en 00010H est placé dans IP.</p> <p>L'exécution se poursuit à partir de cette nouvelle adresse.</p>	
IRET	<p>Retour d'interruption ; transfère le contrôle à l'adresse de retour (CS et IP) qui avait été sauvegardée par le déclenchement de l'interruption. Restaure le registre des indicateurs.</p>	IRET

