Chapitre 3 : **La programmation du microprocesseur 8086**

III.1 Généralités

Le programme est une suite d'instructions. Une instruction est un mot binaire décrivant l'action que doit exécuter le microprocesseur,

Une instruction est définie par son code opératoire, qu'on écrit généralement en hexa, mais pour une plus grande facilité on les désignera par leurs mnémoniques. Le mnémonique est une abréviation en caractère latin.

La programmation en assembleur utilise un ensemble d'instructions appelé jeu d'instructions. Chaque microprocesseur à son propre jeu d'instructions.

Pour les microprocesseurs classiques, le nombre d'instructions reconnues varie entre 75 et 150 (microprocesseurs CISC : Complex Instruction Set Computer). Il existe aussi des microprocesseurs dont le nombre d'instructions est très réduit (microprocesseurs RISC : Reduced Instruction Set Computer) : entre 10 et 30 instructions, permettant d'améliorer le temps d'exécution des programmes.

Les instructions peuvent être classées en groupes :

- instructions de transfert de données ;
- instructions arithmétiques ;
- instructions logiques ;
- instructions de branchement ...

III.2 Les modes d'adressage

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

Les modes d'adressage définissent comment identifier l'opérande de chaque instruction.

Un mode d'adressage spécifie comment calculer l'adresse de la mémoire d'un opérande en utilisant les informations contenues dans les registres et / ou Les constantes contenues dans l'instruction.

Il y a plusieurs modes d'adressage dans le 8086:

III.2.1 Adressage registre

Ce mode utilise les registres internes de CPU, dans ce mode le CPU transfert d'une copie d'un octet (8 bits) ou un mot (16 bits) d'un registre (source) vers un autre registre (destination).

Exemples : MOV AX, BX : Copier le contenu de BX dans AX

Notez que le transfert d'un registre 8 bits avec un registre 16 bits est une erreur.

III.2.2 Adressage immédiat

L'adressage immédiat transfère la source, un octet immédiat ou le mot, dans le registre de destination ou l'emplacement mémoire.

Exemples : ADD AX, 177h : Additionner le registre AX avec le nombre hexadécimal 177.

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

III.2.3 Adressage direct

L'adressage direct déménage un octet ou un mot entre un emplacement mémoire et un registre. L'adresse de la case mémoire ou plus précisément son [Rseg : Offset] est précisé directement dans l'instruction.

Exemples : MOV AL, [1234H] : Copier le contenu de la mémoire d'adresse DS:1234 dans AL.

III.2.4 Adressage indirect par registre

L'adressage indirect de registre transfère un octet ou un mot entre un registre et un emplacement mémoire adressés par un intermédiaire (un des 4 registres d'offset BX, BP, SI ou DI).

Exemple : MOV AX, [BX] : transfert de contenu de la case mémoire 16 bits pointé par DS:BX.

MOV AX, [ES:BP] : Charger AX par le contenu de la mémoire d'adresse ES:BP.

Remarque :

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

III.2.5 Adressage basé

Il utilise les registres de base (BX, BP) comme pointeur. On peut ajouter un déplacement de registre offset pour déterminer l'offset,

Exemple : MOV AX, [BX] : Charger AX par le contenu de la mémoire d'adresse DS:BX

MOV AX, [BX+1000H] : Charger AX par le contenu de la mémoire d'adresse DS:BX+1000

III.2.6 Adressage indexé

Le même mode que le basé, mais il utilise les registres d'indexes (SI, DI) comme pointeur. On peut ajouter un déplacement de registre offset pour déterminer l'offset

Exemple : MOV AX, [SI] : Charger AX par le contenu de la mémoire d'adresse DS:SI

MOV AX, [SI+100] : Charger AX par la mémoire d'adresse DS:SI+100.

III.2.7 Adressage basé indexé

C'est un mode combine entre de modes d'adressage comme l'indique son nom.

Exemple : MOV AX, [BP+SI-8] : AX est chargé par la mémoire d'adresse SS:BP+SI-8

MOV AX, [BX+DI+4] : AX est chargé par la mémoire d'adresse DS:BX+DI+4

III.3 Les instructions de transfert

Elles permettent de déplacer des données d'une source vers une destination :

- registre vers mémoire ;
- registre vers registre ;
- mémoire vers registre.

Remarque : le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire (pour ce faire, il faut passer par un registre intermédiaire).

III.3.1 Instruction MOV

Le format général d'une instruction MOV est : **MOV Destination, Source**
(MOV Op1, Op2).
L'instruction MOV copie le contenu de l'opérande source dans l'opérande de destination.
La source peut être : Registre, mémoire, valeur immédiate (latérale).
La destination peut être : Registre, mémoire.
Toutes les combinaisons autorisées des deux opérandes sont énumérées ci-dessous:

MOV AX, BX	copier un registre dans un autre.
MOV M, AX	copier le contenu d'une case mémoire dans un registre.
MOV AX, M	copier un registre dans une case mémoire.
MOV AX, 4	copier une constante dans un registre.
MOV taille M, 4	copier une constante dans une case mémoire. (taille = BYTE ou WORD)

Les actions suivantes ne sont pas autorisées avec l'instruction MOV:

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

III.3.2 Instruction de pile

Les instructions de pile sont des instructions importantes qui stockent et récupérent des données à partir de la pile mémoire selon le protocole LIFO (dernier entré, premier sorti). Le processeur 8086 prend en charge 4 instructions de pile:

PUSH
POP
PUSF (Push Drapeau enregistrer)
POPF (Pop Drapeau enregistrer)

a/ PUSH

Le format général d'une instruction de PUSH est : **PUSH opérande**

- L'instruction PUSH stocke le contenu de l'opérande Op (16 bits) dans le sommet de Pile
- Décrémente SP de 2

Les opérandes admis de l'instruction PUSH sont:

PUSH AX registre
PUSH word [@] une case mémoire de 16 bits.

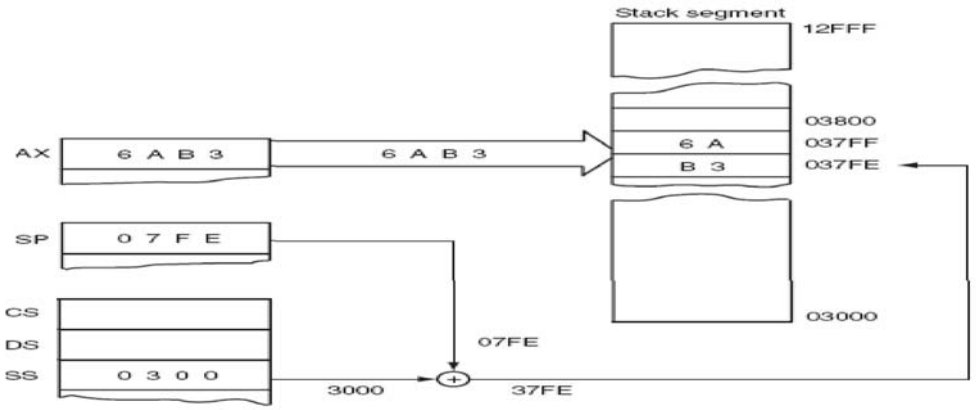

Figure III.1 : Fonctionnement de l'instruction PUSH

La figure III.1 montre le fonctionnement de l'instruction PUSH AX aprés exécution.
PUSHF : Empilement de registre FLAG vers le sommet de la PILE.

b/ POP

Le format général d'une instruction POP est : **POP opérande**.

- L'instruction POP Dépile une valeur de 16 bits depuis le sommet de la Pile (STACK SEGMENT) et la stocke dans les opérandes (16 bits).
- Incrémente SP de 2

- Les opérandes admis de l'instruction POP sont:

POP AX registre
POP word [@] une case mémoire de 16 bits.

Figure III.2 montre le fonctionnement de l'instruction POP BX aprés exécution.

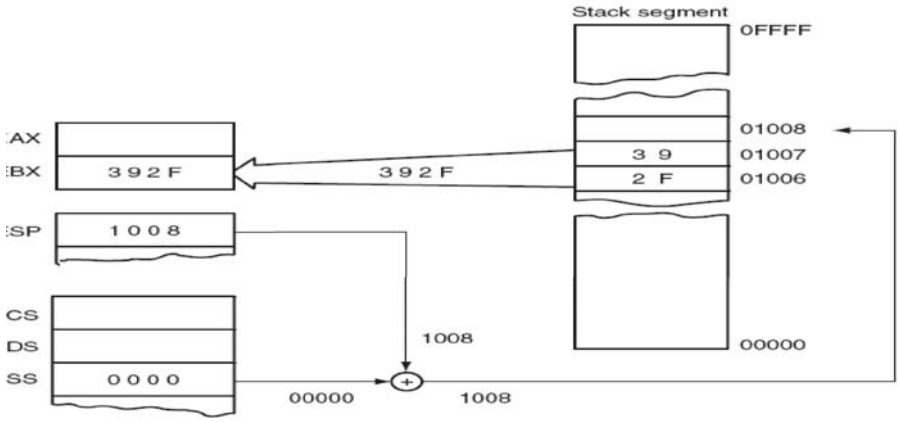

Figure III.2 : Fonctionnement de l'instruction POP

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

III.3.3 Instruction XCHG

Le format général de l'instruction XCHG est : **XCHG Op1, Op2.**
Echange l'opérande Source Op2 avec l'opérande Destination Op1. Impossible sur segment.
XCHG AL, AH
XCHG [@], AH
XCHG AH, [@]

III.4 Les instructions arithmétiques

Avec le 8086 on a les instructions arithmétiques de base : l'addition, la soustraction, la multiplication et la division. Qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles. Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signés ou non signés.

III.4.1 Instruction ADD

La forme générale de l'instruction ADD est : **ADD destination, source.**
L'instruction ADD Additionne l'opérande source et l'opérande destination avec résultat dans l'opérande destination, Destination ⟸ destination + source.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, valeur immédiate.
Toutes les combinaisons autorisées des deux opérandes sont énumérées ci-dessous:
Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.
Exemple : ADD AX, 4 ADD AX, BX ADD [@], BX ADD AX, [SI]

III.4.2 Instruction ADC

L'instruction ADC Additionne l'opérande source, l'opérande destination et le curry avec résultat dans l'opérande destination :
ADC Destination, Source ; Destination ⟸ destination + source + CF.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, valeur immédiate.
Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

III.4.3 Instruction INC

Incrémentation d'un opérande.
INC Destination ; Destination ⟸ destination+1.
La destination peut être : registre, mémoire.
Les FLAGs affectés : OF ; AF ; SF ; ZF ; PF
CF reste inchangé
Pour incrémenter une case mémoire, il faut préciser la taille :
INC byte [ ], INC word [ ]

III.4.4 Instruction SUB

La forme générale de l'instruction SUB (Soustraction) est : **SUB destination, source.**
L'instruction SUB Soustrait l'opérande source et l'opérande destination et place le résultat dans l'opérande de destination. Destination ⟸ destination – source.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, valeur immédiate.
Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

III.4.5 Instruction SBB

L'instruction SBB fait la soustraction entre deux opérandes avec carry.
SBB Destination, Source ; Destination ⟸ destination – source – CF.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, valeur immédiate.
Les FLAGs affectés : CF ; OF; AF ; SF ; ZF ; PF.

III.4.6 Instruction DEC

Décrémentation d'un opérande : **DEC Destination** ; Destination ⟸ destination – 1.
La destination peut être : registre, mémoire.
Les FLAGs affectés : OF ; AF ; SF ; ZF ; PF.
CF reste inchangé.

III.4.7 Instruction NEG

L'instruction NEG donne le complément à 2 de l'opérande Op : remplace Op par son négatif : NEG Destination ; Destination ⟸ Cà2 (destination).
La destination peut être : registre, mémoire.
Les FLAGs affectés : CF ; OF; AF; SF ; ZF; PF.

III.4.8 Instruction CMP

L'instruction CMP fait la comparaison entre deux opérandes et positionne les drapeaux en fonction du résultat. L'opérande Op1 n'est pas modifié.
CMP Op1, Op2 ; Résultat ⟸ Op1 – Op2.
Op1 peut être : registre, mémoire.
Op2 peut être : registre, mémoire, valeur immédiate.
Les FLAGs affectés : CF ; OF ; AF ; SF ; ZF ; PF.

III.4.9 Instruction MUL

Cette instruction exécute la multiplication entre l'accumulateur et un opérande (non signé) : **MUL Opérande**.
Si l'opérande est une valeur sur 8 bits : AX ⟸ AL * Opérande.
Si l'opérande est une valeur sur 16 bits : DX : AX ⟸ AX * Opérande.
L'opérande 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)
Les FLAGs affectés : CF et OF 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
Exemple :

```
MUL  BL   ;     AL x  BL    →    AX
MUL  CX   ;     AX x CX     →    DX:AX
```

III.4.10 Instruction DIV

Cette instruction exécute la Division entre l'accumulateur et un opérande (non signé):
DIV Opérande.
Si l'opérande sur 8 bits : AL ⟸ AX / Opérande ; AH ⟸ Reste.
Si l'opérande sur 16 bits : AX ⟸ (DX : AX) / Opérande ; DX ⟸ Reste.
L'opérande 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].
Les FLAGs : CF ; OF ; ZF ; SF ; PF ; et AF sont inconnus (X).

III.4.11 Instructions IMUL et IDIV

La multiplication et la division de l'accumulateur avec un opérande (signé).
Les mêmes syntaxes que MUL et DIV.

III.5 Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont : ET ; OU ; OU exclusif ; décalages et rotations.

III.5.1 Instruction AND

ET logique entre deux opérandes bit à bit : **AND Destination, Source ;**
Destination ⟸ destination (ET) source.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, immédiate.
Les FLAGs affectés : ZF ; SF ; PF

III.5.2 Instruction OR

OU logique entre deux opérandes bit à bit : **OR Destination, Source ;**
Destination ⟸ destination (OU) source.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, immédiate.
Les FLAGs affectés : ZF ; SF ; PF

III.5.3 Instruction XOR

OU Exclusif entre deux opérandes bit à bit : **XOR Destination, Source ;**
Destination ⟸ destination (ou exclusif) source.
La destination peut être : registre, mémoire.
La source peut être : registre, mémoire, immédiate.
Les FLAGs affectés : ZF ; SF ; PF
Remarque : Astuces sur les instructions logiques (OR, XOR, AND) :
1. Forçage à 1 : c'est le OR avec un « 1 ».
2. Masquage à 0 : c'est le AND avec un « 0 ».
3. Inverser un bit : c'est le XOR avec un « 1 ».

III.5.4 Instruction TEST

L'instruction TEST effectue l'opération ET.
La différence est que l'instruction AND change l'opérande de destination, alors que l'instruction TEST positionne uniquement les FLAG.
TEST Op1, Op2 ; Résultat ⟸ Op1 (ET) Op2.
Op1 peut être : registre, mémoire.
Op2 peut être : registre, mémoire, immédiate.
Les FLAGs affectés : ZF ; SF ; PF

III.5.5 Instruction NOT

Complément à 1 de l'opérande : NOT **Destination.**
La destination peut être : registre, mémoire.

III.5.6 Instructions de Décalages

Ces instructions déplacent d'un certain nombre de positions les bits d'un mot vers la gauche ou vers la droite.
Dans les décalages, les bits qui sont déplacés sont remplacés par des zéros. Il y a les décalages logiques (opérations non signées) et les décalages arithmétiques (opérations signées).
Le 8086 prend en charge quatre instructions de décalage différentes :

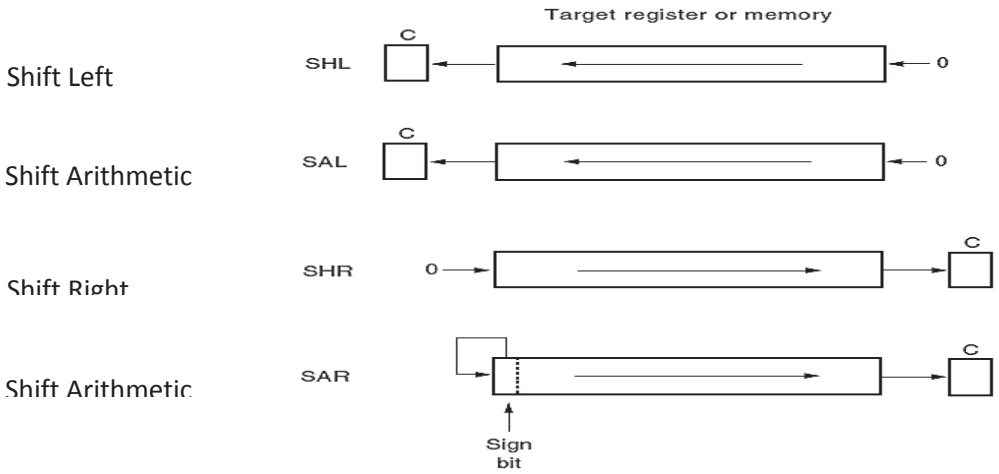

Figure III.3 Les instructions de décalage

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

La forme générale des instructions de décalage est : **XXX destination, N**
Où: XXX est le mnémonique de décalage (i.e., SHL, SHR, SAL, SAR),
La destination peut être : registre, mémoire et l'opérande *N* peut être soit une constante (immédiat) soit le registre CL :

Exemples: SHL AX, 1
SHL AX, CL

III.5.7 Instructions de Rotations

Dans les rotations, les bits déplacés dans un sens sont réinjectés de l'autre côté du mot. Les quatre opérations de rotation disponibles apparaissent dans la figure suivante :

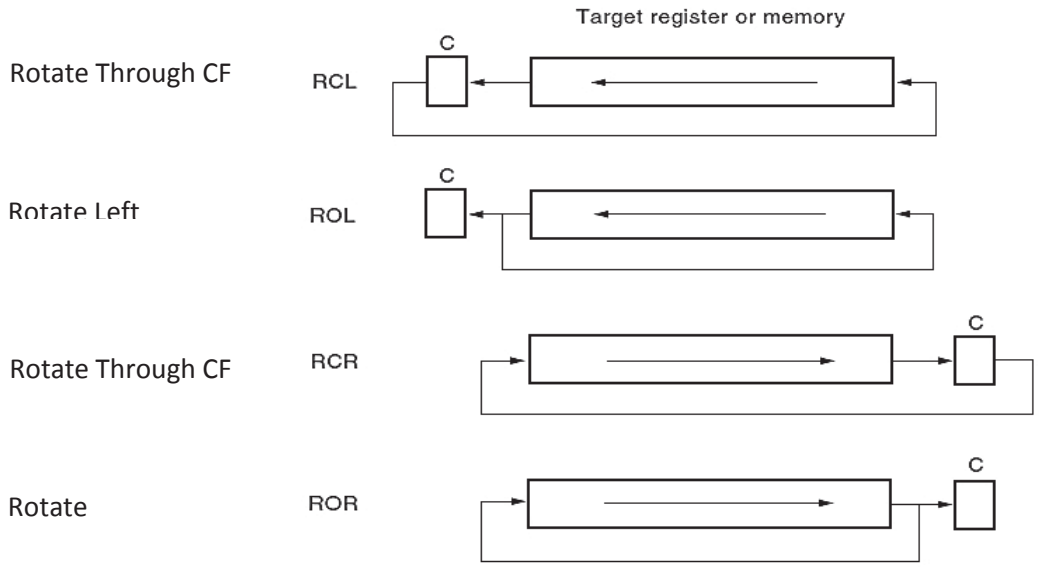

Figure III.4: instructions de rotation du registre ou de la mémoire indiquant la direction et le fonctionnement de chaque rotation.

La forme générale des instructions de rotation est : **XXX destination, N**
Où: XXX est la mnémonique de rotation (i.e., RCL, RCR, ROL, ROR),

Exemples: ROL AX, 1
ROL AX, CL

III.6 Les instructions de branchement

Les instructions de branchement (ou saut) permettent de faire un saut dans le programme, et exécuter une instruction beaucoup plus éloignée, ou au contraire revenir en arrière. Il existe 3 types de saut :

- saut inconditionnel ;
- sauts conditionnels ;
- appel de sous-programmes.

III.6.1 Instruction de saut inconditionnel

JMP label ; Provoque un saut sans condition à la ligne portant l'étiquette label.
Où label est un identificateur d'adresse de programme.

III.6.2 Instructions de sauts conditionnels

Instructions de sauts conditionnels sont conditionnées par l'état des indicateurs (drapeaux) qui sont eux même positionnés par les instructions précédentes.
Jxxx label où: xxx décrit la condition du saut.

Tableau 1: Liste des instructions de sauts conditionnels du 8086.

	Instruction	Description	Tested Condition
1	JA	Jump if above	CF = 0 and ZF = 0
2	JNBE	Jump if neither bellow nor equal	CF = 0 and ZF = 0
3	JBE	Jump if bellow or equal	CF = 1 or ZF = 1
4	JNA	Jump if not above	CF = 1 or ZF = 1
5	JAE	Jump if above or equal	CF = 0
6	JNB	Jump if not bellow	CF = 0
7	JB	Jump if bellow	CF = 1
8	JNAE	Jump if neither above nor equal	CF = 1
9	JG	Jump if greater than	ZF = 0 and SF = OF
10	JNLE	Jump if not less nor equal	ZF = 0 and SF = OF
11	JLE	Jump if less or equal	ZF = 1 or SF ≠ OF
12	JNG	Jump if not greater	ZF = 1 or SF ≠ OF
13	JGE	Jump if greater than or equal	SF = OF
14	JNL	Jump if not less than	SF = OF
15	JL	Jump if less than	SF ≠ OF
16	JNGE	Jump if neither greater nor equal	SF ≠ OF
17	JO	Jump if overflow	OF = 1
18	JNO	Jump if no overflow	OF = 0
19	JZ	Jump if zero	ZF = 1
20	JE	Jump if equal	ZF = 1
21	JNZ	Jump if not zero	ZF = 0
22	JNE	Jump if not equal	ZF = 0
23	JC	Jump on carry	CF = 1
24	JNC	Jump on not carry	CF = 0
25	JS	Jump if sign (negative)	SF = 1
26	JNS	Jump if no sign (not negative)	SF = 0
27	JP	Jump if parity	PF = 1
28	JPE	Jump if even parity	PF = 1
29	JNP	Jump if no parity	PF = 0
30	JPO	Jump if parity odd	PF = 0
31	JCXZ	Jump if CX is zero	CX = 0

Remarque :

- Les termes supérieur et inférieur (above and below) pour les nombres non signés.
- Les termes plus grand et plus petit (greater than and less than) pour les nombres signés.

III.6.3 Instruction CALL

Appel d'une procédure (sous-programme) : CALL **nom_ sous-programme**
Le CPU fait plusieurs choses quand une instruction CALL est exécutée: (1) Il ajuste le contenu du registre IP comme s'il va exécuter l'instruction suivante.

(2) Il stocke le contenu du registre IP dans la pile de programme.
(3) On ajuste le contenu du registre d'IP de nouveau pour pointer sur la première instruction de la procédure appelée.
L'unité centrale continue à exécuter les instructions jusqu'à ce qu'elle rencontre une instruction RET.

III.6.4 Instruction RET

Un RET (Return) instruction de retour de sous-programme : RET
Quand le CPU rencontre une instruction RET, il effectue les actions suivantes :
(1) Il récupère le contenu du registre IP stocké dans la pile de programme par l'instruction CALL.
(2) Il met la valeur restaurée dans le registre IP et continue l'exécution.
Le CPU revient à l'état où il était lorsque l'instruction CALL a été rencontrée.

III.6.5 Instructions des interruptions

INT n: Appel à l'interruption logicielle n° n
IRET : Cette instruction termine un sous-programme de traitement d'une interruption.

III.6.6 Instruction de boucles LOOP

L'instruction LOOP décrémente le contenu de CX et provoque un saut relatif court si ce dernier n'est pas nul.
La forme générale d'une instruction de boucle est : **LOOP label**
Où label est un identificateur d'adresse de programme.
L'instruction LOOP est équivalente au deux instructions :
DEC CX
JNZ label
La diférence entre les deux instructions ci-dessus et de l'instruction LOOP est que cette dernière, n'affecte pas les états des flags.
Le 8086 à 4 variations de l'instruction LOOP:
LOOPZ (Loop While Zero) ou **LOOPE** (Loop While Equal) : si le flag ZF=1 et si CX ≠ 0.
LOOPNZ (Loop While Not Zero) ou **LOOPNE** (Loop While Not Equal) : si le flag ZF=0 et si CX ≠ 0.
Ces variations permettent le déroulement du programme pour deux conditions (le registre CX et le drapeau zéro (ZF) en même temps).

III.7 Les instructions de manipulation de chaines

Dans le Cpu 8086 on a 5 instructions de manipulation de chaînes :
MOVS, LODS, STOS sont des instructions de transfert, elles peuvent être répétées à l'aide du préfixe **REP**
CMPS et SCAS sont des instructions de comparaisons, elles peuvent être répétées à l'aide du préfixe **REPZ**

Remarque : Chacune des instructions existe en deux versions, entre deux octets, ou entre 2 words, le préfixe REP pour les instructions de transfert MOVS LODS STOS, Ce préfixe utilise le registre CX comme un compteur de répétition.

Les préfixes REPE et REPZ : Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=1 et CX≠0.

Les préfixes REPNE et REPNZ : Répétition des instructions de comparaisons SCAS et CMPS tant que ZF=0 et CX≠0.

Ces instructions sont utilisées sans opérandes. les registres utilisés sont : pour la source DS:SI, et pour la destination ES:DI. A chaque exécution d'une instruction de traitement de chaîne, les registres d'indexes sont incrémentés ou décrémentés selon le flag DF de registre d'état.

* DF = 0 : SI et DI sont incrémentés, (l'instruction CLD mit (DF = 0))
* DF = 1 : SI et DI sont décrémentés. (l' instruction STD mit (DF = 1))

SI et DI sont incrémentés de 1 ou de 2 selon que l'opération s'effectue sur un octet (byte) ou sur un mot de 16 bits (word).

MOVSB : Copie Le contenu d'un octet de mémoire DS: [SI] dans un octet de mémoire ES: [DI], puis auto inc/decrémente les registres SI et DI.

MOVSW : Copie Le contenu du mot de mémoire DS: [SI] dans le mot de mémoire ES: [DI], puis auto inc/decrémente de 2 les registres SI et DI

LODSB : Copie l'octet source DS: [SI] dans AL puis inc/décr le registre SI.

LODSW : Copie le mot source DS: [SI] dans AX puis inc/décr de 2 le registre SI.

STOSB : Copie AL dans l'octet destination ES: [DI] et inc/décr le registre DI.

STOSW : Copie AX dans le mot destination ES: [DI] et inc/décr de 2 le registre DI.

CMPSB : Compare l'octet source ES : [DI] avec l'octet destination DS: [SI], positionne les indicateurs puis inc/décrémente les registres SI et DI.

CMPSW : Compare le mot source ES : [DI] avec mot destination DS: [SI], positionne les indicateurs puis inc/décrémente de 2 les registres SI et DI

SCASB : Compare AL avec l'octet destination ES : [DI], positionne les indicateurs puis inc/décr le registre DI.

SCASW : Compare AX avec le mot destination ES : [DI], positionne les indicateurs puis inc/décr le registre DI de 2.