Chapitre II : La programmation en assembleur

Objectifs du chapitre

Avoir les outils et l'habilité de programmer en assembleur 8086

Contenu du chapitre

II.1 Introduction

L'écriture d'un programme pour un système à microprocesseur consiste à spécifier, directement ou indirectement, une séquence d'instructions machine. Les instructions machine à l'intérieur du système ont une forme binaire (0 et 1) qui est difficile, voire impossible, pour les gens de travailler avec et de comprendre. Il est préférable d'écrire ces programmes avec les symboles les plus connus du jeu de caractères alphanumériques. Chaque instruction symbolique peut être traduite en une instruction codée en binaire. Avant de voir les différentes instructions symboliques du microprocesseur 8086 on doit voir d'abord les modes d'adressage existants.

II.2 Modes d'adressage

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

INST *Od , Os*

L'opération *INST* est réalisée entre les 2 opérandes *Od et Os* et le résultat est toujours récupéré dans l'opérande de gauche (*Od*). Il y a des instructions qui agissent sur un seul opérande et d'autres qui n'ont pas besoin d'opérande. Les opérandes peuvent être des registres, le contenu de cases mémoires ou même des constantes. La façon avec laquelle le microprocesseur accède à ces opérandes s'appelle le mode d'adressage. Dans la suite on utilisera les abréviations suivantes :

INST : Opération quelconque

Op : Opérande

Os : Opérande source

Od : Opérande destination

R : Registre quelconque

Rseg : Registre Segment

Roff : Registre d'offset

Rb : Registre de base

Ri : Registre d'index

Im : Donnée (constante)

Adr : Adresse

[Adr] : Contenu mémoire

Off : Offset de l'adresse

Dep : Déplacement (constante).

II.2.1 Adressage Registre

Dans ce mode d'adressage, l'opération se fait sur un ou deux registres.

INST R

INST R , R

Exemples :

INC AX ; incrémenter le registre *AX*

MOV AX, BX ; Copier le contenu de *BX* dans *AX*

II.2.2 Adressage Immédiat

Dans ce mode, un des opérandes est une constante (valeur).

INST R , Im

INST taille [adr] , Im

Exemples :

ADD AX, 243h ; additionner le registre AX avec le nombre hexadécimal 243

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

II.2.3 Adressage Direct

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, [123] ; Copier le contenu de la case d'adresse *DS:123* dans *AX*

MOV [4567h], AX ; Copier le contenu de *AX* dans la case d'adresse *DS:4567h*

MOV AX, [SS:243] ; Copier le contenu de la case d'adresse *SS:243* dans *AX*

Remarque : l'instruction *INST [adr] , [adr]* n'est pas valide (Intel ne l'a pas implémentée).

II.2.4 Adressage Indirect

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 quatre (4) registres d'offset *BX*, *BP*, *SI* ou *DI*. L'adresse logique aura la forme *[Rseg:Roff]*. Si Rseg n'est pas spécifié, un registre

segment par défaut sera utilisé (voir Tableau II.1) :

INST R , [Rseg:Roff]

INST [Rseg:Roff] , R

INST taille [Rseg:Roff] , Im

Registre segment	Registres associés
CS	IP
SS	SP et/ou BP
DS	BX et/ou SI
ES (Opérations sur chaînes de caractères)	DI

Tableau II.1 : Registres segments associés aux registres d'Offset.

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

II.2.4.1 Adressage Basé

L'offset se trouve dans l'un des deux registres de base *BX* ou *BP*. On peut insérer un déplacement qui sera ajouté au contenu de *Rb* 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**:BP-200*

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

II.2.4.2 Adressage Indexé

L'offset se trouve dans l'un des deux registres de base *SI* ou *DI*. On peut insérer 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 le contenu de la mémoire d'adresse ***DS**:SI+500*

MOV AX, [DI-8] ; Charger *AX* par le contenu de la mémoire d'adresse ***DS**:DI-8*

MOV AX, [ES:DI+4] ; Charger *AX* par le contenu de la mémoire d'adresse ***ES**:DI+4*

II.2.4.3 Adressage Basé Indexé

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] ; Charger *AX* par le contenu de la mémoire d'adresse ***DS****:BX+SI*

MOV AX, [BX+DI+5] ; Charger *AX* par le contenu de la mémoire d'adresse ***DS****:BX+DI+5*

MOV AX, [BP+SI-8] ; Charger *AX* par le contenu de la mémoire d'adresse ***SS****:BP+SI-8*

MOV AX, [BP+DI] ; Charger *AX* par le contenu de la mémoire d'adresse ***SS****:BP+DI*

II.3 Etude du jeu d'instructions

L'ensemble des instructions machines qu'un microprocesseur peut exécuter s'appelle jeu d'instructions. Selon le travail effectué, on distingue plusieurs types d'instructions :

II.3.1 Les instructions de transfert

Dans ce groupe d'instructions, on s'intéresse aux instructions suivantes :

• **MOV Od , Os ;** Copie l'opérande *Source* dans l'opérande *Destination*. Les opérandes peuvent être des registres, des cases mémoire ou des valeurs immédiates :

MOV R1 , R2

MOV R , M

MOV M , R

MOV R , im

MOV taille M , im ; pour lever toute ambiguïté on précise la taille de la case mémoire (BYTE ou WORD)

• **PUSH Op** ; Empiler l'opérande *Op* dans la zone réservée pour la pile (*Op* doit être un opérande 16 bits).

PUSH R16

PUSH word M

• **POP Op** ; Dépiler de la zone réservée pour la pile vers l'opérande *Op* (*Op* doit être un opérande 16 bits).

POP R16

POP word M

• **XCHG OD , OS** ; Echange l'opérande *Source* avec l'opérande *Destination*. Impossible sur segment.

XCHG R1 , R2

XCHG M , R

XCHG R , M

II.3.2 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 entiers 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 permette de faire des calculs en décimal (BCD). Ces dernières instructions ne seront pas vues en détail ici.

• **ADD Od , Os ;** Additionne l'opérande source et l'opérande destination, le résultat est sauvegardé dans l'opérande de destination, *Od + Os → Od*

Exemples : ADD AX, 1234 ; ADD AX, BX ; ADD [5678], AX ; ADD BX, [SI]

• **ADC Od , Os ;** Additionne l'opérande source, l'opérande destination et le drapeau Carry. Le résultat est dans l'opérande destination : *Od + Os + C → Od*

• **INC Op ;** Incrémente l'opérande *Op* : *Op + 1 → Op*

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

INC byte M ou INC word M

• **SUB Od , Os ;** Soustrait l'opérande source de l'opérande destination avec résultat dans l'opérande destination, *Od - Os → Od*

• **SBB Od , Os ;** Soustrait l'opérande source et le carry 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

• **MUL Op ;** 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*). C'est instruction où un seul opérande est spécifié, l'autre est pris par défaut.

MUL Op8 ; *AL x Op8 → AX*

MUL Op16 ; *AX x Op16 → 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*

***** 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.

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

• **DIV Op ;** Effectue la division *AX/Op8* ou (*DX|AX)/Op16* 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].

DIV Op8 ; *AX* / *Op8* , Quotient → *AL* , Reste → *AH*

DIV Op16 ; *DX:AX* / *Op16* , Quotient → *AX* , Reste → *DX*

- *Op* ne peut pas être une donnée (immédiat)
- 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 respecter le signe.

Contenu de *AL* Contenu de *AX* après exécution de l'instruction *CBW*

(0000 0101)2 = (+5)10 (0000 0000 0000 0101)2

(1111 1011)2 = (-5)10 (1111 1111 1111 1011)2

• **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*.

II.3.3 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.

II.3.4 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 ou d'un double octet (″word″**).**

II.3.4.1 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.

II.3.4.2 Forcer un bit à 1

Pour forcer un bit à ″1″ sans modifier les autres bits, on utilise l'opérateur logique 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.

II.3.4.3 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.

II.3.5 Les instructions de décalage et rotation

Dans les instructions de décalage et de rotation, le nombre de bits à décaler est spécifié par l'opérande *k*, qui peut être soit une constante soit le registre *CL* (si le nombre est variable) :

INST R/M, k

Exemples :

INST AX, 1 ; décalage ou rotation de AX de 1 bit

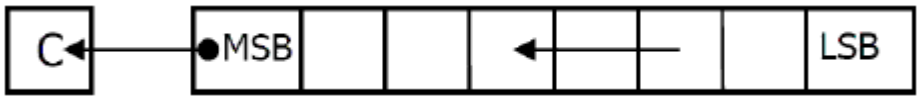
INST BX, CL ; décalage ou rotation de *BX* de *CL* bits

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

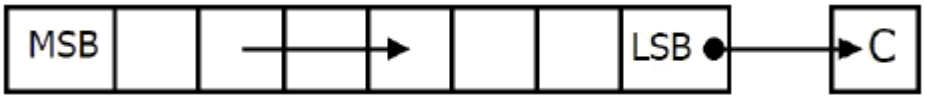
INST byte [BX], 1 ; décalage ou rotation du contenu de la case mémoire d'adresse *BX* de *1* bit.

• **SHL R/M, k** (SHift Left) décalage logique à gauche d'un registre ou du contenue d'une case mémoire de k bits.

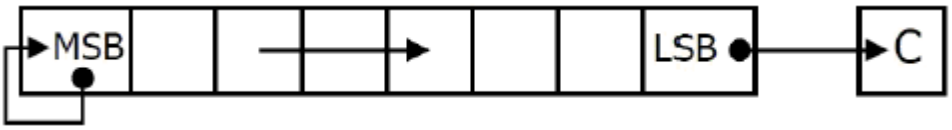
• **SAL R/M, k** (Shift Arithmetic Left) décalage arithmétique à gauche de k bits.



• **SHR R/M, k** (SHift Right) décalage logique à droite de k bits.

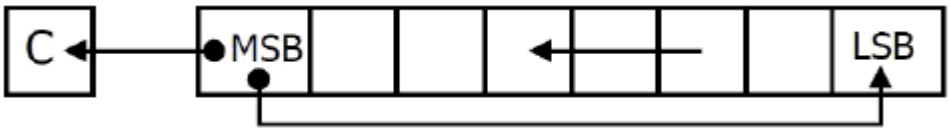


• **SAR R/M, k** (Shift Arithmetic Right) décalage arithmétique à droite de k bits.

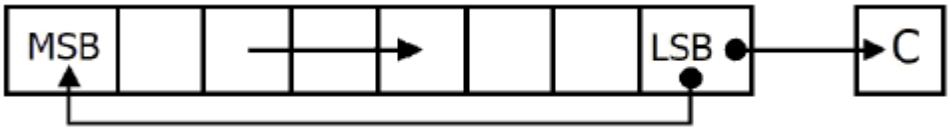


Les décalages arithmétiques permettent de conserver le signe. Ils sont utilisés pour effectuer des opérations arithmétiques comme des multiplications ou des divisions par 2.

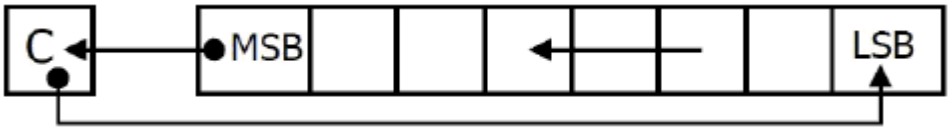
• **ROL R/M, k** (ROtate Left) Rotation à gauche de k bits.



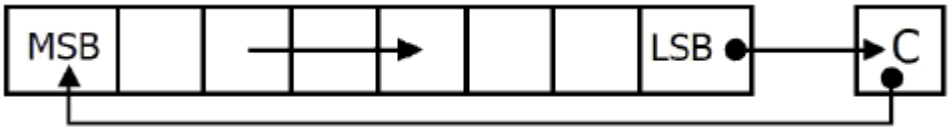
• **ROR R/M, k** (ROtate Right) Rotation à droite de k bits.



• **RCL R/M, k** (Rotate through Cf Left) Rotation à gauche de k bits à travers le bit Carry.

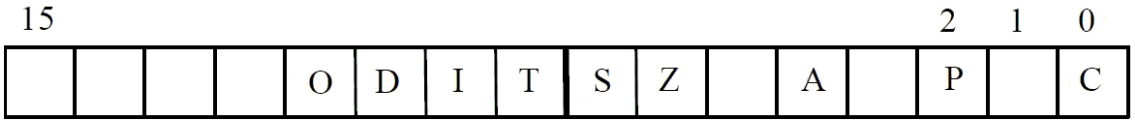


• **RCR R/M, k** (Rotate through Cf Right) Rotation à droite de k bits à travers le bit Carry.



II.3.6 Les instructions agissant sur les indicateurs

Les bits (indicateurs) du registre d'état sont positionnés automatiquement par l'unité arithmétique et logique après chaque opération.



Intel nous permet de modifier ces indicateurs (drapeaux) directement ou indirectement par les instructions suivantes :

• **CLC** (**CL**ear **C**arry) Positionne le drapeau *C* à *0*.

• **STC** (**S**e**T** **C**arry) Positionne le drapeau *C* à *1*

• **CMC** (**C**o**M**plement **C**arry) Complémente le drapeau *C*

• **CLD** (**CL**ear **D**irection) Positionne le Drapeau *D* à *0*

• **STD** (**S**e**T** **D**irection) Positionne le Drapeau *D* à *1*

• **CLI** (**CL**ear **I**nterrupt) Positionne le Drapeau *I* à *0*

• **STI** (**S**e**T** **I**nterrupt) Positionne le Drapeau *I* à *1*

• **LAHF** (**L**oad **AH** from **F**lags) Copie l'octet bas du registre d'état dans le registre *AH*

• **SAHF** (**S**tore **AH** in **F**lags) Opération inverse de *LAHF* : Transfert *AH* dans l'octet bas du registre d'état

• **PUSHF** (**PUSH F**lags onto stack) Empile le registre d'état,

• **POPF** (**POP F**lags off stack) Dépile le registre d'état.

II.3.7 Les instructions de transfert de programme

Dans un programme 8086/8088 l'exécution séquentielle des instructions est déterminée par le contenu du registre "Code Segment" (*CS*) et du pointeur d'instruction (*IP*). La séquence normale se fait instruction après l'autre. Lors du transfert de programme, l'unité d'interfaçage des bus obtient l'adresse de la prochaine instruction en utilisant les nouvelles valeurs de *IP* et *CS*. Quatre groupes de transferts de programme sont disponibles pour le 8086/8088 : les instructions de contrôle de boucles, les branchements inconditionnels, les branchements conditionnels et les instructions relatives aux interruptions.

II.3.7.1 Les instructions de contrôle de boucles

Les instructions de contrôle d'itération peuvent être utilisées pour gérer la répétition des boucles logicielles. Ces instructions utilisent le registre *CX* comme compteur.

• **LOOP etiq ;** Quand 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 *etiq*, sinon il continue le programme à la ligne suivante. (Voir l'extrait de programme ci-dessous)

```
        MOV   CX, xx
ici:    INST  Op1, Op2
        INST  Op1, Op2
          :     :    :
        INST  Op1, Op2
        LOOP  ici
        INST  Op1, Op2
```

• **LOOPZ etiq** (**LOOP** while **Z**ero) Décrémente le registre *CX* (aucun flag n'est positionné), on reboucle vers la ligne *etiq* 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 etiq** (**LOOP** while **N**on **Z**ero) Décrémente le registre *CX* et reboucle vers la ligne *etiq* tant que *CX* est différent de zéro et le flag *Z* est égal à 0.

• **JCXZ etiq** (**J**ump if **CX Z**ero) branchement à la ligne *etiq* si *CX = 0* indépendamment de l'indicateur *Z*

IV.3.7.2 Branchements inconditionnels

Dans ce groupe d'instructions, le transfert est effectué sans condition chaque fois que l'instruction est exécutée.

• **JMP etiq** Provoque un saut sans condition à la ligne portant l'étiquette etiq.

• **CALL etiq** Appel d'une procédure (sous-programme) qui commence à la ligne etiq. La position de l'instruction suivant le *CALL* est empilée pour assurer un retour correct après l'exécution du sous-programme.

• **RET** Retour de sous-programme. L'exécution du programme continue à l'instruction juste après le *CALL* qui a activé la procédure. (Adresse récupérée depuis la pile).

II.3.7.3 Branchements conditionnels

Les instructions de branchement conditionnel sont des sauts qui peuvent transférer le contrôle ou non en fonction de l'état des drapeaux du microprocesseur au moment de l'exécution de l'instruction. Si la condition est vraie, le contrôle est transféré vers la cible spécifiée dans l'instruction. Si la condition est fausse, le contrôle passe à l'instruction suivante (celle qui suit le saut conditionnel). Les instructions de branchement conditionnel s'utilisent en général immédiatement après une instruction de comparaison *CMP* ou d'autres instructions ayant modifié certains drapeaux du registre d'états (Flags).

Dans la suite nous allons utiliser la terminologie *supérieur* ou *inférieur* pour les relations impliquant des valeurs non-signées et *plus grand* ou *plus petit* pour les valeurs signées.

• **JA etiq** (Jump if Above) Sauter à la ligne *etiq* si supérieur (non signé). (si *C* ou *Z = 0*)

• **JAE etiq** (Jump if Above or Equal) Sauter à la ligne *etiq* si supérieur ou égal (non signé). (si *C = 0*)

• **JB etiq** (Jump if Bellow) Sauter à la ligne *etiq* si inférieur (non signé). (si *C = 1*).

• **JBE etiq** (Jump if Bellow or Equal) Sauter à la ligne *etiq* si inférieur ou égal (non signé). (*C* ou *Z = 1*)

• **JE/JZ etiq** (Jump if Equal or Zero) Sauter à la ligne *etiq* si résultat nul ou si égalité. (si *Z = 1*)

• **JNE/JNZ etiq** (Jump if Not Equal or Not Zero) Sauter à la ligne *etiq* si résultat non nul ou si différent. (si *Z = 0*)

• **JC etiq** (Jump if Carry) Sauter à la ligne *etiq* s'il y a retenu. (si *C = 1*)

• **JNC etiq** (Jump if No Carry) Sauter à la ligne *etiq* s'il n'y a pas de retenu. (*C = 0*)

• **JO etiq** (Jump if Overflow) Sauter à la ligne *etiq* si dépassement. (si *O = 1*)

• **JNO etiq** (Jump if No Overflow) Sauter à la ligne *etiq* s'il n'y a pas de dépassement. (si *O = 0*)

• **JP/JPE etiq** (Jump if Parity or Parity Even) Sauter à la ligne *etiq* si parité paire. (si $P = 1$)

• **JNP/JPO etiq** (Jump if No Parity or if Parity Odd) Sauter à la ligne *etiq* si parité impaire. (si $P = 0$)

• **JS etiq** (Jump if Sign) Sauter à la ligne *etiq* si signe négatif. (si $S = 1$)

• **JNS etiq** (Jump if No Sign) Sauter à la ligne *etiq* si signe positif. (si $S = 0$)

• **JG etiq** (Jump if Greater) Sauter à la ligne *etiq* si plus grand (signé). (si $(S \oplus O) + Z = 0$)

• **JGE etiq** (Jump if Greater or Equal) Sauter à la ligne *etiq* si plus grand ou égal (signé).(si $S \oplus O = 0$)

• **JL etiq** (Jump if Less) Sauter à la ligne *etiq* si plus petit (signé). (si $S \oplus O = 1$)

• **JLE etiq** (Jump if Less or Equal) Sauter à la ligne *etiq* si plus petit ou égal (signé).

(si $(S \oplus O) + Z = 1$)

Exemple :

```
MOV AX, 1234h
MOV BX, 5678h
CMP AX, [BX]
JS ICI ; aller à ICI si négatif ; sinon faite la suite
XCHG AX, BX
JMP SUITE
ICI: SHL AX, 4
SUITE: SHR BX, 1
; etc …
```

II.3.7.4 Instructions relatives aux interruptions

Les instructions d'interruption permettent aux routines de service d'interruption d'être activées par des programmes ainsi que par des périphériques matériels externes.

• **INT n** (INTerrupt) Appel à l'interruption logicielle n° *n*. Le registre d'états, le registre *CS* ainsi que le registre *IP* sont stockés dans la pile. (Voir la section gestion de la pile pour plus de détail).

• **INTO n** (INTerrupt on Overflow) Appel à l'interruption logicielle n° *n* en cas de dépassement de capacité. Si l'indicateur de débordement est levé, le même travail que celui de l'instruction *INT* sera effectué.

• **IRET** (Interrupt RETurn) Retour d'interruption. En récupérant les valeurs des registres *IP*, *CS* et *FLAGS* depuis la pile, cette instruction permet de sortir de la routine de service d'interruption et de revenir au programme appelant juste au point où l'interruption s'est produite.

II.3.8 Les instructions d'accès aux ports d'entrées-sorties

Le microprocesseur 8086 communique avec l'environnement extérieur (périphériques) à travers un circuit d'interface comportant des ports d'entrées/sorties. Le 8086 fourni deux instructions permettant de lire ou d'écrire sur ces ports.

• **IN AL/AX, port** (INput) Cette instruction transfère un octet ou un mot du port d'entrée n° *port* vers le registre *AL* ou *AX*. Elle permet donc de lire des données à partir du monde extérieur. Si l'adresse *port* tient sur 8 bits, elle peut être spécifiée immédiatement, sinon il faut passer par le registre *DX*.
Exemples : *IN AL, 4Dh MOV DX, 376h*
IN AL, DX

• **OUT port, AL/AX** (OUTput) Cette instruction transfère un octet ou un mot du registre *AL* ou *AX* vers le port de sortie n° *port*. Ceci permet d'agir sur le monde extérieur. L'adressage du port se fait de la même façon que pour *IN*.
Exemples :
OUT 23h, AX ; écrit *AL* dans le port d'adresse 23h et AH dans 24h (23 + 1)
MOV DX, 5B8h ; Charge l'adresse 5B8h dans le registre *DX*
OUT DX, AL ; écrit *AL* dans le port d'adresse spécifiée par *DX*