Chapitre 2 : Introduction au jeu d'instructions et interruptions

II.1 Jeux d'instructions du μP

Chaque microprocesseur reconnait un ensemble d'instructions appelé jeu d'instructions (Instruction Set) fixé par le constructeur. 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.

Une instruction est définie par son code opératoire, valeur numérique binaire difficile à manipuler par l'être humain. On utilise donc une notation symbolique pour représenter les instructions : les mnémoniques. Un programme constitué de mnémoniques est appelé programme en assembleur.

Les différents types d'instructions peuvent être classés :

II.1.1 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

Syntaxe : MOV destination, source

Il existe différentes façons de spécifier l'adresse d'une case m´mémoire dans une instruction ce sont **les modes d'adressage.**

a) Mode d'adressage par registre

mov AX, BX : charge le contenu du registre BX dans le registre AX. Dans ce cas, le transfert se fait de registre à registre

b) Mode d'adressage immédiat

mov AL, 12H : charge le registre AL avec la valeur 12H. La donnée est fournie immédiatement avec l'instruction

c) Mode d'adressage direct

mov BL, [1200H] : transfère le contenu de la case mémoire d'adresse effective (offset) 1200H vers le registre BL. L'instruction comporte l'adresse de la case mémoire où se trouve la donnée. L'adresse effective représente l'offset de la case mémoire dans le segment de données (segment dont l'adresse est contenue dans le registre DS) : segment par défaut. (Fig. II.1)

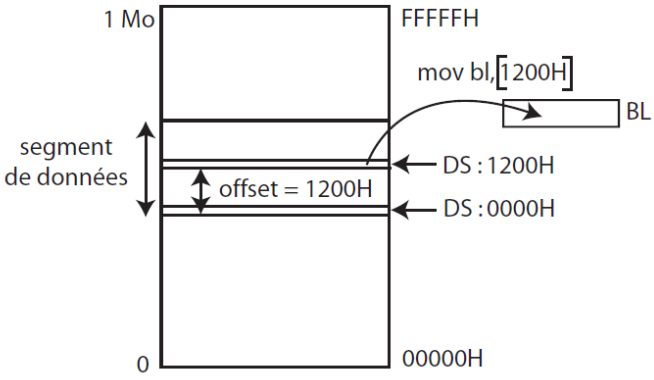


Fig. II.1 Mode d'adressage direct

Remarque :

Dans le cas de l'adressage immédiat de la mémoire, il faut indiquer le format de la donnée : octet ou mot (2 octets) car le microprocesseur 8086 peut manipuler de données sur 8 bits ou 16 bits. Pour cela, on doit utiliser un spécificateur de format **:**

- **mov byte ptr [1100H],65H** : transfère la valeur 65H (sur 1 octet) dans la case mémoire d'offset 1100H ; (Fig. II.2)
- **mov word ptr [1100H],65H** : transfère la valeur 0065H (sur 2 octets) dans les cases mémoire d'offset 1100H et 1101H. (Fig. II.2)

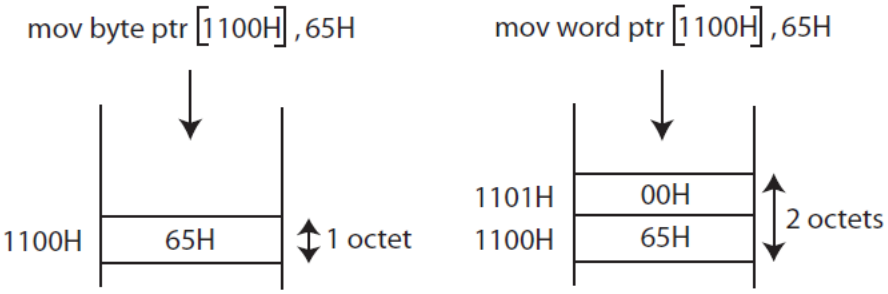


Fig. II.2 Mode d'adressage immédiat avec formats byte ptr et word ptr

d) Mode d'adressage basé

mov al,[bx] : transfère la donnée dont l'offset est contenu dans le registre de base BX vers le registre AL. Le segment associé par défaut au registre BX est le segment de données : on dit que l'adressage est basé sur DS ; (Fig. II.3)

mov al,[bp] : le segment par d´défaut associé au registre de base BP est le segment de pile. Dans ce cas, l'adressage est basé sur SS.

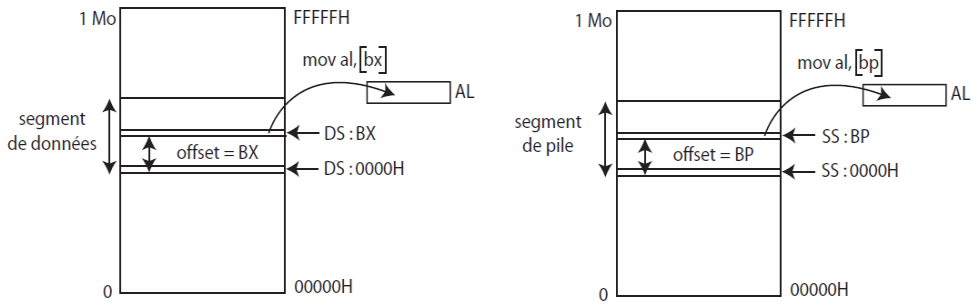


Fig. II.3 Mode d'adressage basée

e) Mode d'adressage Indexé

Semblable à l'adressage basé, sauf que l'offset est contenu dans un registre d'index SI ou DI, associés par défaut au segment de données.

mov al,[si] : charge le registre AL avec le contenu de la case mémoire dont l'offset est contenu dans SI ;

f) Mode d'adressage basé et Indexé

L'offset est obtenu en faisant la somme d'un registre de base bx, d'un registre d'index si et d'une valeur constante 100h. Exemple :

mov ah,[bx+si+100H]

II.1.1 Les instructions arithmétiques :

Les instructions arithmétiques de base sont l'addition, la soustraction, la multiplication et la division qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles.

Addition: ADD opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 + opérande2.

Exemples :

- **add ah,[1100H]** : ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH (adressage direct) ;
- **add ah,[bx]** : ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH (adressage basé) ;
- **add byte ptr [1200H],05H** : ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (adressage immédiat).

Soustraction : SUB opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 − opérande2.

Multiplication : MUL opérande, où opérande est un registre ou une case mémoire. Cette instruction effectue la multiplication du contenu de AL par un opérande sur 1 octet ou du contenu de AX par un opérande sur 2 octets. Le résultat est placé dans AX si les données à multiplier sont sur 1 octet (résultat sur 16 bits), dans (DX,AX) si elles sont sur 2 octets (résultat sur 32 bits).

Exemples :

mov al,51

mov bl,32

mul bl

→ AX = 51 × 32

Division : DIV opérande, où opérande est un registre ou une case mémoire**.** Cette instruction effectue la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX,AX)

par un opérande sur 2 octets. Résultat : si l'opérande est sur 1 octet, alors AL = quotient et AH = reste ; si l'opérande est sur 2 octets, alors AX = quotient et DX = reste.

Exemples :

mov ax,35

mov bl,10

div bl

→ AL = 3 (quotient) et AH = 5 (reste)

II.1.2 Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont :

ET logique : AND opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 ET opérande2.

Exemple :

```
mov al,10010110B
mov bl,11001101B      →
and al, bl
```

$$\begin{array}{ll} AL = & 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ BL = & 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ \hline AL = & 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \end{array}$$

OU logique : OR opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 OU op´erande2.

Exemple : dans le mot 10110001B on veut mettre à1 les bits 1 et 3 sans modifier les autres bits.

$$\begin{array}{cccccccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 1 & 0 & 1 & 1 & \mathbf{0} & 0 & \mathbf{0} & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \hline 1 & 0 & 1 & 1 & \mathbf{1} & 0 & \mathbf{1} & 1 \end{array} \leftarrow \textbf{masque}$$

Les instructions correspondantes peuvent s'écrire :

mov ah, 10110001B

or ah, 00001010B

Complement à 1: NOT opérande

L'opération effectuée est : opérande ← opérande.

Exemple :

```
mov al,10010001B
not al
```

→ $AL = \overline{10010001}B = 01101110B$

Complément à 2 : NEG opérande

L'opération effectuée est : opérande ← opérande + 1.

Exemple :

```
mov al,25
mov bl,12
neg bl
add al,bl
```

$\rightarrow \qquad \mathrm{AL} = 25 + (-12) = 13$

OU exclusif : XOR opérande1, opérande2

L'opération effectuée est : op´erande1 ← opérande1 ⊕ opérande2.

Exemple : mise à zéro d'un registre :

```
mov al,25
xor al,al
```

$\rightarrow \qquad \mathrm{AL} = 0$

II.1.3 Les instructions de décalage et de rotation

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).

Dans les rotations, les bits déplacés dans un sens sont réinjectés de l'autre côté du mot.

a) **Décalage logique vers la droite (Shift Right) :** SHR opérande, n
Cette instruction décale l'opérande de n positions vers la droite.

Exemple :

```
mov al,11001011B
shr al,1
```

→ entrée d'un 0 `a la place du bit de poids fort ; le bit sortant passe à travers l'indicateur de retenue CF.

Remarque : si le nombre de bits `a décaler est supérieur `a 1, ce nombre doit ˆêtre placé dans le registre CL ou CX.

Exemple : d´décalage de AL de trois positions vers la droite :

mov cl,3

shr al,cl

b) Décalage logique vers la gauche (Shift Left) : SHL opérande, n

Cette instruction décale l'opérande de n positions vers la droite.

Exemple :

```
mov al,11001011B
shl al,1
```

→ entrée d'un 0 `a la place du bit de poids faible ; le bit sortant passe à travers l'indicateur de retenue CF.

Même remarque que précédemment si le nombre de positions à décaler est supérieur à 1.

Décalage arithmétique vers la droite : SAR opérande, n

Ce décalage conserve le bit de signe bien que celui-ci soit décalé.

Exemple :

```
mov al,11001011B
sar al,1
```

→ le bit de signe est réinjecté.

c) Décalage arithmétique vers la gauche : SAR opérande, n

Identique au d´décalage logique vers la gauche.

d) Rotation à droite (Rotate Right) : ROR opérande, n

Cette instruction décale l'opérande de n positions vers la droite et réinjecte par la gauche les bits sortant.

Exemple :

```
mov al,11001011B
ror al,1
```

→ réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

e) **Rotation à gauche (Rotate Left) :** ROL opérande, n

Cette instruction décale l'opérande de n positions vers la gauche et réinjecte par la droite les bits sortant.

Exemple :

```
mov al,11001011B
rol al,1
```

→ réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

II.1.4 Les instructions de branchement

Les instructions de branchement (ou saut) permettent de modifier l'ordre d'exécution des instructions du programme en fonction de certaines conditions. Il existe 3 types de saut :

- **Instruction de saut inconditionnel :** JMP label

Cette instruction effectue un saut (jump) vers le label spécifié. Un label (ou étiquette) est une représentation symbolique d'une instruction en mémoire :

```
            ⋮        ← instructions précédant le saut
        jmp suite
            ⋮        ← instructions suivant le saut (jamais exécutées)
suite :     ...      ← instruction exécutée après le saut
```

Exemple :

```
boucle : inc ax
         dec bx          →      boucle infinie
         jmp boucle
```

Remarque : l'instruction JMP ajoute au registre IP (pointeur d'instruction) le nombre d'octets (distance) qui sépare l'instruction de saut de sa destination. Pour un saut en arrière, la distance est n´négative (codée en complément à 2).

- **Instructions de sauts conditionnels :** Jcondition label

Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite, sinon l'exécution se poursuit séquentiellement à l'instruction suivante.

La condition du saut porte sur l'état de l'un (ou plusieurs) des indicateurs d'état (flags) du microprocesseur :

instruction	nom	condition
`JZ label`	Jump if Zero	saut si ZF = 1
`JNZ label`	Jump if Not Zero	saut si ZF = 0
`JE label`	Jump if Equal	saut si ZF = 1
`JNE label`	Jump if Not Equal	saut si ZF = 0
`JC label`	Jump if Carry	saut si CF = 1
`JNC label`	Jump if Not Carry	saut si CF = 0
`JS label`	Jump if Sign	saut si SF = 1
`JNS label`	Jump if Not Sign	saut si SF = 0
`JO label`	Jump if Overflow	saut si OF = 1
`JNO label`	Jump if Not Overflow	saut si OF = 0
`JP label`	Jump if Parity	saut si PF = 1
`JNP label`	Jump if Not Parity	saut si PF = 0

Remarque : les indicateurs sont positionnés en fonction du résultat de la dernière opération.

Exemple :

```
        ⋮                ← instructions précédant le saut conditionnel
    jnz suite
        ⋮                ← instructions exécutées si la condition ZF = 0 est vérifiée
suite : ...              ← instruction exécutée à la suite du saut
```

Remarque : il existe un autre type de saut conditionnel, les sauts arithmétiques. Ils suivent en général l'instruction de comparaison : CMP opérande1, opérande2

condition	nombres signés	nombres non signés
$=$	`JEQ label`	`JEQ label`
$>$	`JG label`	`JA label`
$<$	`JL label`	`JB label`
$\neq$	`JNE label`	`JNE label`

Exemple :

```
            cmp ax,bx
            jg superieur
            jl inferieur
superieur : ...
              ⋮
inferieur : ...
```

- **Appel de sous-programmes :** pour ´éviter la répétition d'une même s´séquence d'instructions plusieurs fois dans un programme, on rédige la s´séquence une seule fois en lui attribuant un nom (au choix) et on l'appelle lorsqu'on en a besoin. Le programme appelant est le programme principal. La s´séquence appelée est un sous-programme ou procédure.

Ecriture d'un sous-programme :

```
nom_sp    PROC
              ⋮   ← instructions du sous-programme
          ret     ← instruction de retour au programme principal
nom_sp    ENDP
```

Remarque : une procédure peut être de type NEAR si elle se trouve dans le même segment ou de type FAR si elle se trouve dans un autre segment.

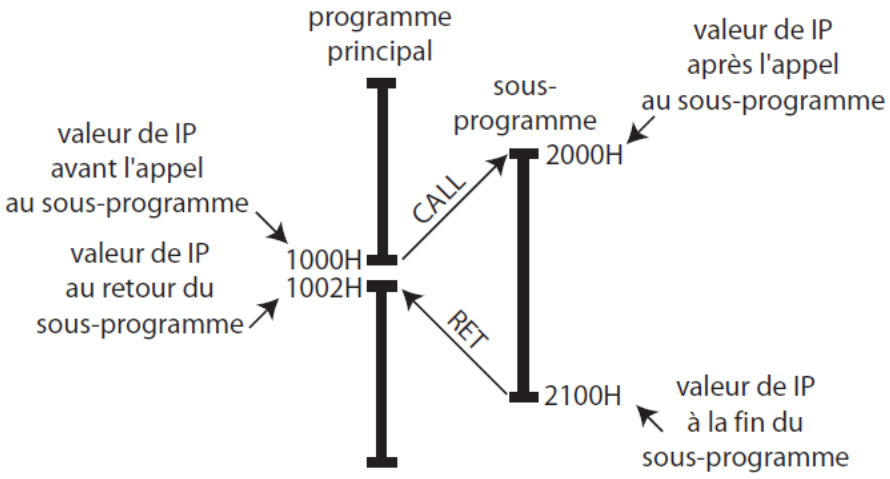
Exemple :

```
ss_prog1    PROC    NEAR
ss_prog2    PROC    FAR
```

Appel d'un sous-programme par le programme principal : CALL procédure

```
      ⋮     ← instructions précédant l'appel au sous-programme
call nom_sp ← appel au sous-programme
      ⋮     ← instructions exécutées après le retour au programme principal
```

Lors de l'exécution de l'instruction CALL, le pointeur d'instruction IP est chargé avec l'adresse de la première instruction du sous-programme. Lors du retour au programme appelant, l'instruction suivant le CALL doit être exécutée, c.-à-d. que IP doit être rechargé avec l'adresse de cette instruction.



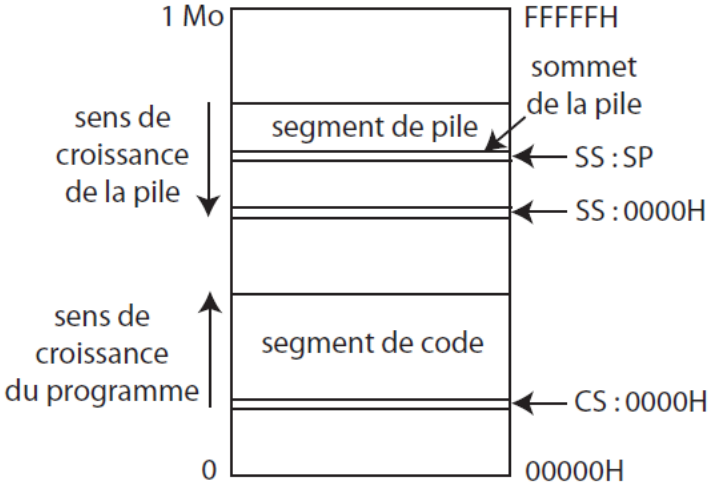
Avant de charger IP avec l'adresse du sous-programme, l'adresse de retour au programme principal, c-`a-dire le contenu de IP, est sauvegardée dans une zone mémoire particulière appelée pile. Lors de l'exécution de l'instruction RET, cette adresse est récupérée à partir de la

pile et rechargée dans IP, ainsi le programme appelant peut se poursuivre. Fonctionnement de la pile : la pile est une zone mémoire fonctionnant en mode LIFO (Last In First Out : dernier entre, premier sorti). Deux opérations sont possibles sur la pile :

- **empiler** une donnée : placer la donnée au sommet de la pile ;
- **dépiler** une donnée : lire la donnée se trouvant au sommet de la pile.

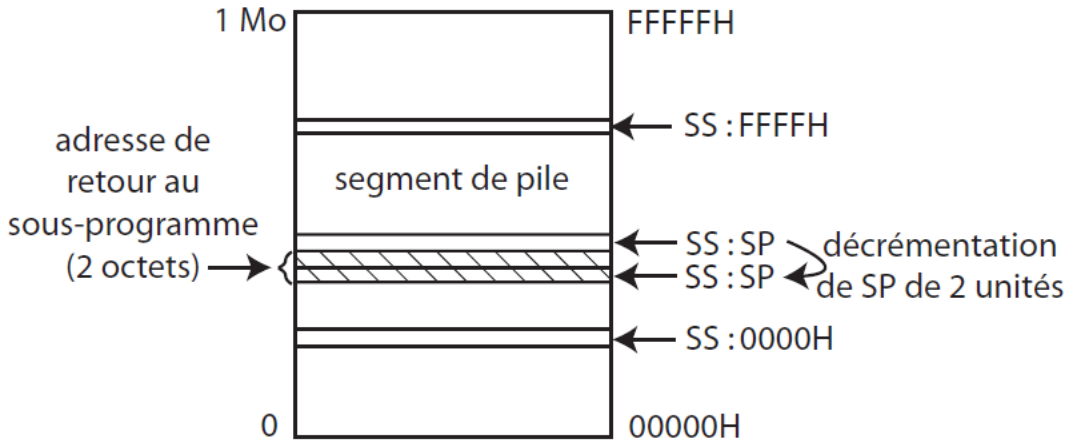
Le sommet de la pile est repéré par un registre appelé pointeur de pile (SP : Stack Pointer) qui contient l'adresse de la dernière donnée empilée.

La pile est d´définie dans le segment de pile dont l'adresse de départ est contenue dans le registre SS.



Remarque : la pile et le programme croissent en sens inverse pour diminuer le risque de collision entre le code et la pile dans le cas où celle-ci est placée dans le même segment que le code (SS = CS).

Lors de l'appel à un sous-programme, l'adresse de retour au programme appelant (contenu de IP) est empilée et le pointeur de pile SP est automatiquement décrémenté. Au retour du sous-programme, le pointeur d'instruction IP est rechargé avec la valeur contenue sommet de la pile et SP est incrémenté.



La pile peut également servir à sauvegarder le contenu de registres qui ne sont pas automatiquement sauvegardés lors de l'appel à un sous-programme :

- instruction d'empilage : PUSH opérande
- instruction de dépilage : POP opérande

Où opérande est un registre ou une donnée sur 2 octets (on ne peut empiler que des mots de 16 bits).

Exemple :

```
push ax         ; empilage du registre AX ...
push bx         ; ... du registre BX ...
push [1100H]    ; ... et de la case mémoire 1100H-1101H
     :
pop  [1100H]    ; dépilage dans l'ordre inverse de l'empilage
pop  bx
pop  ax
```

II.2 Méthodes de programmation

Etapes de la réalisation d'un programme :

• Définir le problème à résoudre : que faut-il faire exactement ?

• Déterminer des algorithmes, des organigrammes : comment faire ? Par quoi commencer, puis poursuivre ?

• Rédiger le programme (code source) :

– utilisation du jeu d'instructions (mnémoniques) ;

– création de documents explicatifs (documentation).

• Tester le programme en réel ;

• Corriger les erreurs (bugs) éventuelles : déboguer le programme puis refaire des

Tests jusqu''a obtention d'un programme fonctionnant de manière satisfaisante.

Langage machine et assembleur :

• Langage machine : codes binaires correspondant aux instructions ;

• Assembleur : logiciel de traduction du code source écrit en langage assembleur (mnémoniques).

Réalisation pratique d'un programme :

• Rédaction du code source en assembleur à l'aide d'un éditeur (logiciel de traitement de texte ASCII) :

– édit sous MS-DOS,

– Notepad (bloc-note) sous Windows,

• Assemblage du code source (traduction des instructions en codes binaires) avec un

Assembleur :

– MASM de Microsoft,

– TASM de Borland,

– A86 disponible en shareware sur Internet, ...

Pour obtenir le code objet : code machine exécutable par le microprocesseur ;

• Chargement en mémoire centrale et exécution : rôle du système d'exploitation (ordinateur) ou d'un moniteur (carte de d´développement à base de microprocesseur).

Pour la mise au point (d´débogage) du programme, on peut utiliser un programme d'aide

à la mise au point (comme DEBUG sous MS-DOS) permettant :

• l'exécution pas à pas ;

• la visualisation du contenu des registres et de la mémoire ;

• la pose de points d'arrêt ...

Structure d'un fichier source en assembleur : pour faciliter la lisibilité du code source en assembleur, on le rédige sous la forme suivante :

```
labels        instructions        commentaires
label1 :      mov ax,60H          ; ceci est un commentaire ...
    ⋮             ⋮                   ⋮
sous_prog1    proc near           ; sous-programme
    ⋮             ⋮                   ⋮
sous_prog1    endp
    ⋮             ⋮                   ⋮
```

Directives pour l'assembleur :

- Origine du programme en m´emoire : ORG offset

Exemple : org 1000H

- Définitions de constantes : nom constante EQU valeur

Exemple : escape equ 1BH

- Réservation de cases mémoires :

Nom_variable DB valeur initiale

Nom_variable DW valeur initiale

DB: Define Byte, réservation d'un octet;

DW: Define Word, réservation d'un mot (2 octets).

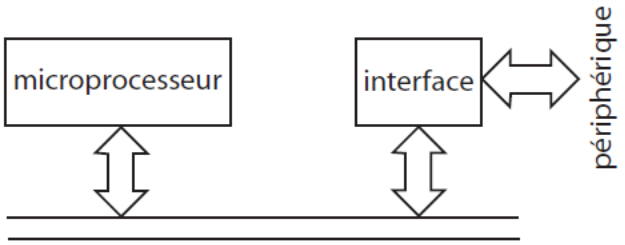
Exemples :

```
nombre1   db   25
nombre2   dw   ?              ; pas de valeur initiale
buffer    db   100 dup (?)    ; réservation d'une zone mémoire
                              ; de 100 octets
```

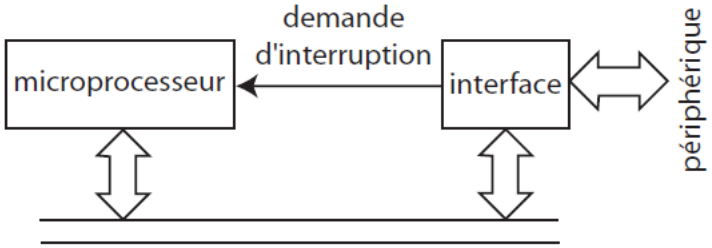
II.3 Les interruptions

II.3.1 Définition d'une interruption

Soit un microprocesseur qui doit ´échanger des informations avec un périphérique :



Lorsqu'une donnée apparaît sur un périphérique, le circuit d'E/S le signale au microprocesseur pour que celui-ci effectue la lecture de la donnée : c'est une demande d'interruption (IRQ : Interrupt Request) :



Avantage : le microprocesseur effectue une lecture des ports d'E/S seulement lorsqu'une donnée est disponible, ce qui permet de gagner du temps et d'´éviter de perdre des données.

Exemples de périphériques utilisant les interruptions :

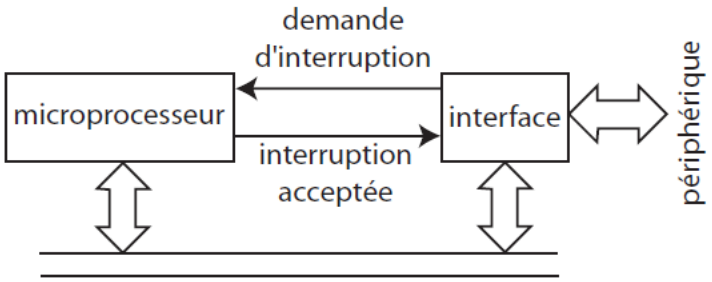
- clavier : demande d'interruption lorsqu'une touche est enfoncée ;
- port série : demande d'interruption lors de l'arrivée d'un caractère sur la ligne de transmission.

Remarque : les interruptions peuvent être générées par le microprocesseur lui-même en cas de problèmes tels qu'une erreur d'alimentation, une division par zéro ou un circuit mémoire d´défectueux (erreurs fatales). Dans ce cas, la demande d'interruption conduit `a l'arrˆet du microprocesseur.

II.3.2 Prise en charge d'une interruption par le microprocesseur

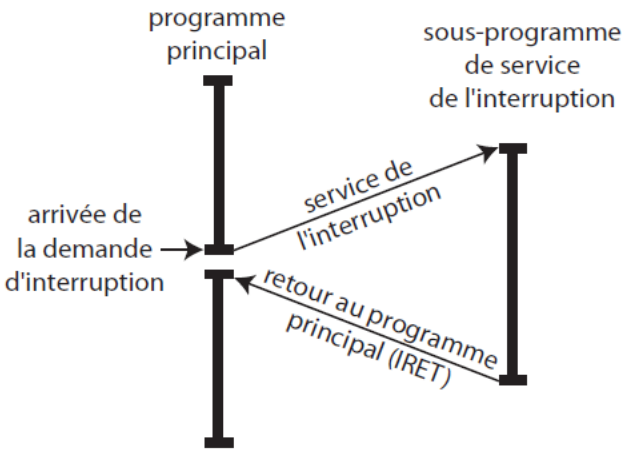
A la suite d'une demande d'interruption par un périphérique :

- le microprocesseur termine l'exécution de l'instruction en cours ;
- il range le contenu des principaux registres sur la pile de sauvegarde : pointeur d'instruction, flags, ...
- il ´émet un accusé de réception de demande d'interruption (Interrupt Acknowledge) indiquant au circuit d'E/S que la demande d'interruption est acceptée :



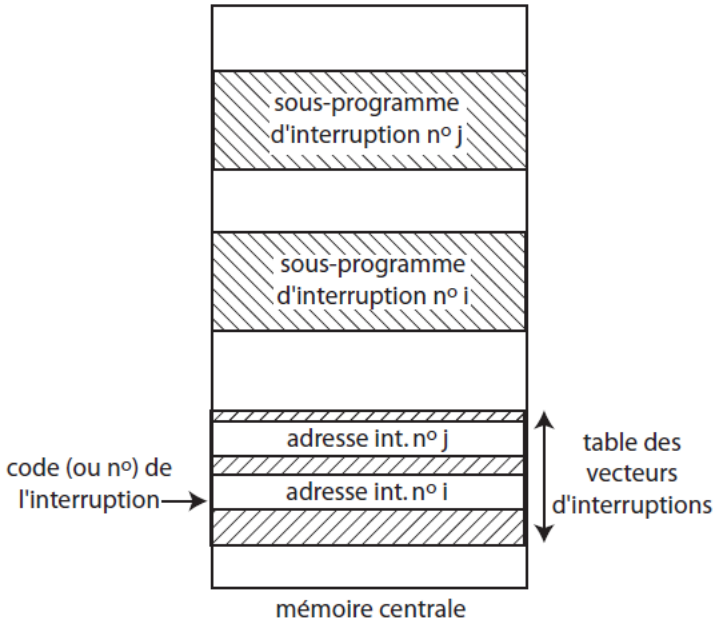
Remarque : le microprocesseur peut refuser la demande d'interruption : celle-ci est alors masquée. Le masquage d'une interruption se fait généralement en positionnant un flag dans le registre des indicateurs d'état. Il existe cependant des interruptions non masquables qui sont toujours prises en compte par le microprocesseur.

- il abandonne l'exécution du programme en cours et va exécuter un sous-programme de service de l'interruption (ISR : Interrupt Service Routine) ;
- après l'exécution de l'ISR, les registres sont restaurés à partir de la pile et le microprocesseur reprend l'exécution du programme qu'il avait abandonné :



II.3.3 Adresses des sous-programmes d'interruptions

Lorsqu'une interruption survient, le microprocesseur a besoin de connaître l'adresse du sous-programme de service de cette interruption. Pour cela, la source d'interruption place sur le bus de données un code numérique indiquant la nature de l'interruption. Le microprocesseur utilise ce code pour rechercher dans une table en mémoire centrale l'adresse du sous-programme d'interruption à exécuter. Chaque ´élément de cette table s'appelle un vecteur d'interruption :

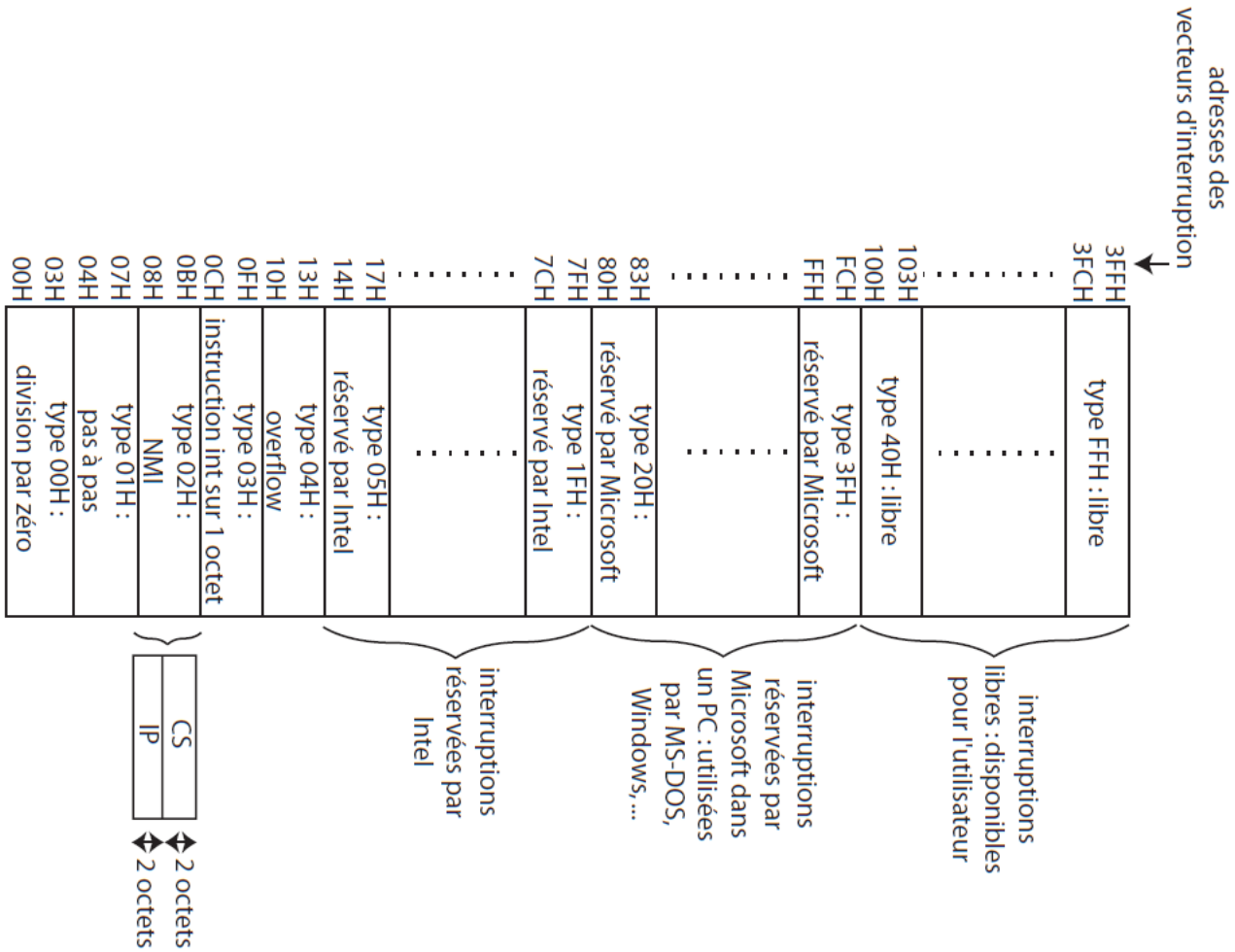


II.3.4 Les interruptions du 8086

Le microprocesseur 8086 peut gérer jusqu'à 256 interruptions. Chaque interruption reçoit un numéro compris entre 0 et 255 appelé type de l'interruption. Trois sortes d'interruptions sont reconnues par le 8086 :

- interruptions matérielles produites par l'activation des lignes INTR et NMI du microprocesseur ;
- interruptions logicielles produites par l'instruction INT n, où n est le type de l'interruption.
- interruptions processeur générées par le microprocesseur en cas de dépassement, de division par zéro ou lors de l'exécution pas à pas d'un programme.

Les interruptions du 8086 sont vectorisées. La table des vecteurs d'interruptions doit obligatoirement commencer `a l'adresse physique 00000H dans la mémoire centrale. Chaque vecteur d'interruption est constitué de 4 octets représentant une adresse logique du type CS : IP.



Remarque : correspondance entre le type de l'interruption et l'adresse du vecteur correspondant :

Adresse vecteur d'interruption = 4 × type de l'interruption

Exemple : interruption 20H, adresse du vecteur = 4 × 20H = 80H.