LE MICROPROCESSEUR 8086

ENSA de TANGER

SOMMAIRE

INTRODUCTION AU LANGAGE MACHINE

INTRODUCTION AU LANGAGE MACHINE
LE MICROPROCESSEUR INTEL 8086

Dans cette partie du cours, nous allons étudier la programmation en langage machine et en assembleur du microprocesseur 80x86 d'Intel.

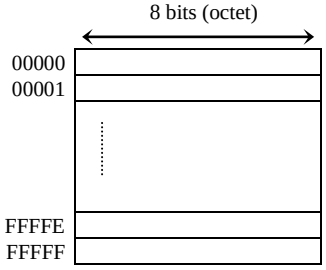
1. CARACTERISTIQUES DU 8086

- Structure 16 bits, adressable par mot de 8 bits,
- Capacité d'adressage de 1 Mo,
- 14 registres internes de 16 bits,
- 24 modes d'adressage,
- Opérations sur des bits, des octets, des mots et des chaînes de caractères,
- Arithmétique signée et non signée,
- Arithmétique binaire ou BCD, sur 8 ou 16 bits.

2. ORGANISATION DE L'ESPACE MEMOIRE

Le 8086 dispose d'un bus d'adresse de 20 bits : A_{19} à A_0 donc d'un espace mémoire adressable de 2^{20} = 1 Mo.

La mémoire peut être vue ainsi :



Le 8086 manipule des données sur 8 bits et sur 16 bits :

- Pour accéder à une donnée sur 8 bits, il suffit de donner son adresse sur 20 bits,
- Les données exprimées sur 16 bits occupent deux octets dans la mémoire. Dans la famille INTEL, les 8 bits de poids faible sont stockés en premiers suivis des 8 bits de poids fort. Cette convention est appelée « **Big Endian** » ou Grand boutiste (la convention rivale est appelée « **Little Endian** » ou Petit Boutiste, elle est adoptée dans les microprocesseurs MOTOROLA).

Ainsi, le 8086 stocke le mot 658Fh à l'adresse 01000h comme suit :

01000h	8Fh
01001h	65h

3. LES REGISTRES DU 8086

Le 8086 a 14 registres de 16 bits classés en 4 groupes :

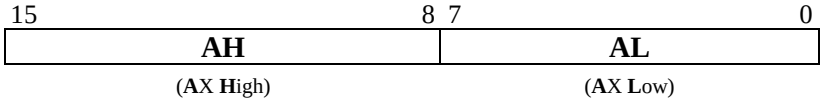
- Les registres de calcul,
- Les registres d'adressage,
- Les registres de segmentation,
- Les registres de contrôle.

3.1. Les registres de calcul

Ce sont les registres AX, BX, CX et DX. Ils sont utilisables dans tout type de traitement (calcul, stockage temporaire,...).

3.1.1. Le registre AX

Appelé **registre accumulateur**. C'est le registre le plus utilisé implicitement par les instructions du 8086. Il peut être sous forme de deux registres indépendants de taille 8 bits : AL et AH.



On écrit : **AX = AH : AL**

Exemple : si AX = 12A4h alors AH = 12h et AL = A4h.

3.1.2. Le registre BX

Il est appelé **registre de base**. Il est utilisé aussi bien pour le calcul que pour l'adressage. Comme AX, le registre BX est constitué de deux registres 8 bits : BL et BH.

BX = BH : BL

3.1.3. Le registre CX

C'est le **registre compteur.** Il est souvent employé d'une manière implicite par certaines instructions comme compteur (instructions de boucle, traitement des chaînes des caractères,...).

CX = CH : CL

3.1.4. Le registre DX

Appelé **registre de donnée**, il est utilisé implicitement par certaines instructions pour stocker des opérandes ou des résultats.

DX = DH : DL

3.1.5. Les registres d'adressage

Il en existe quatre de taille 16 bits : SI, DI, SP et BP.

Les registres SI (Source Index) et DI (Destination Index)
Appelés **Index de source** et **index de destination** respectivement. Ils sont souvent utilisés dans la gestion des tableaux et des chaînes de caractères dans la zone mémoire réservée aux données.

Les registres SP (Stack Pointer) et BP (Base Pointer)

SP et BP s'appellent **pointeur de pile** et **pointeur de base** respectivement. Ils servent à la gestion de la pile.

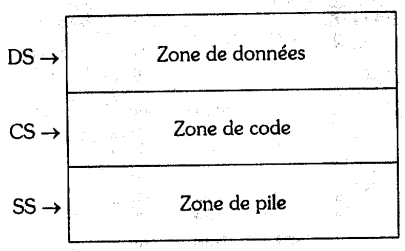
3.2. Les registres de segmentation

Du moment que le 8086 ne dispose que de registres 16 bits, les adresses qu'il gère dans un programme sont aussi exprimé sur 16 bits. Elles s'appellent ″**adresses logiques**″.

Toutefois, l'adresse logique permet d'accéder uniquement à un espace d'adressage de 2^{16} = 64 Ko, alors que le 8086 peut adresser jusqu'à 1 Mo de mémoire (20 lignes d'adresses). Les registres de segmentation ont été spécialement créés pour étendre l'adresse logique à 20 bits.

Le 8086 dispose de 4 registres de segmentation : DS, CS, SS et ES. Chacun est associé à une zone spécifique de la mémoire. En effet, lorsqu'un programme est chargé, il réserve plusieurs zones mémoire généralement indépendantes :

- Une zone réservée aux données (octets, mots, tableaux,...),
- Une zone réservée aux instructions (le code du programme),
- Une zone réservée à la pile.



3.2.1. <u>Registre de segment de données DS (Data Segment)</u>

Il permet d'accéder à la zone de ***données***.

L'adresse réelle d'une donnée est appelée **adresse physique**. Elle est exprimée sur 20 bits et c'est elle qui est effectivement mise sur le bus d'adresse pour accéder à la donnée.

Dans un programme, la donnée sera désignée par une **adresse logique** sur 16 bits et par une valeur de **segment** inscrite dans le registre **DS.**

L'adresse physique est définie par la formule ci-dessous :

$$Adresse\ physique = Adresse\ logique + (10h \times DS)$$

<u>Exemple</u> :
Dans l'instruction MOV AL, [1000h], l'adresse physique, si DS = 120h, est :
Adresse physique = 1000h + (10h x 120h) = 02200h

<u>Remarques</u> :

- L'adresse logique est aussi appelée *déplacement* ou *offset*.
- Une adresse physique a deux représentations :
  - Représentation sur 20 bits.
  - Représentation ″**Segment : Offset**″.

<u>Exemple</u> :
0120 :1000 (Segment 0100 et Offset 0000h) s'écrit sur 20 bits : **02200h**

En général, en fixant DS, on peut accéder aux adresses physiques comprises entre (DS x 10h) et (DS x 10h) + FFFFh, soit un espace de 64 Ko. Cet espace s'appelle un **segment**.

3.2.2. Le registre de segment de code CS (Code Segment)

La zone réservée aux instructions du programme est associée au registre de segmentation CS. Afin que le microprocesseur puisse lire une instruction dans la mémoire, il doit fournir son adresse logique (contenue dans le registre IP) et son segment dans CS.

3.2.3. Le registre de segment de pile SS (Stack Segment)

La pile est une zone mémoire gérée en **LIFO** (Last In, First Out). Elle est utilisée d'une part par le processeur pour sauvegarder l'état du système lors d'un appel à un sous-programme ou à une interruption, et d'autre part, par le programmeur pour passer des paramètres à un sous-programme ou stocker temporairement des données, …

La pile est un élément vital pour le fonctionnement du microprocesseur. La zone mémoire réservée à la pile est associée au registre de segment **SS**.

Le pointeur de pile SP contient l'adresse logique du sommet de la pile. Pour calculer son adresse physique, la formule utilisée est :

Adresse physique du sommet de la pile = SP + (10h x SS)

3.3. Les registres de contrôle

Les registres de contrôle sont susceptibles de modifier le comportement du microprocesseur et informent le programmeur de l'état de celui-ci.

3.3.1. Regsitre d'état SR (ou Flags)

C'est un registre constitué d'un ensemble de bits portant le nom de ″**drapeaux**″ (flag)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Ces drapeaux sont divisés en deux catégories :

- *Indicateurs d'états :*

Les indicateurs d'état sont mis à jour après chaque instruction. Leur valeur indique la nature du résultat obtenu :

Drapeau	Signification	Remarque
CF	Carry	Mis à 1 en cas d'une retenue
PF	Parity	Mis à 1 lorsque le nombre de 1 dans le résultat est pair
AF	Auxilary Carry	Mis à 1 lorsqu'une opération engendre une retenue entre le quartet faible et le quartet fort d'un octet
ZF	Zero	Mis à 1 en cas d'un résultat nul
SF	Sign	Mis à 1 lorsqu'un résultat est négatif
OF	Overflow	Mis à 1 en cas de dépassement de la capacité de calcul

- *Indicateurs de contrôle :*

Les indicateurs de contrôle permettent de modifier le comportement du microprocesseur. Ils sont positionnés par le programmeur.

Drapeau	Signification	Remarque
IF	Interrupt	Autorisations des interruptions : 0 : interruptions inhibées 1 : interruptions autorisées
DF	Direction	Sens de traitement des chaînes de caractères : 0 : sens direct 1 : sens inversé
TF	Trace	Mode de fonctionnement du microprocesseur : 0 : mode normal 1 : mode pas à pas

3.3.2. Registre pointeur d'instructions IP (Instruction Pointer)

IP est un registre qui contient en permanence l'adresse de la *prochaine* instruction à exécuter. Il est automatiquement mis à jour par le microprocesseur après chaque instruction.

IP contient uniquement l'adresse logique de l'instruction, le segment se trouve dans le registre CS :

Adresse physique de la prochaine instruction à exécuter = IP + (10h x CS)

4. LES INSTRUCTIONS DE LA FAMILLE INTEL 80X86

Tous les successeurs du 8086 (80286, 80386, 80486, Pentium) sont compatibles avec son jeu d'instruction. Ainsi, tous les programmes écrits pour le 8086 peuvent être exécutés sur un PC équipé d'un microprocesseur Pentium sans le moindre problème.

4.1. Résumé des principales instructions

Sur les pages suivantes vous trouverez un résumé des principales instructions du 8086 .

INSTRUCTIONS DU 8086

(Résumé des principales instructions)

1. Instructions de transfert

Syntaxe	Rôle	Remarques	Exemples
MOV *destination , source*	Transfère le contenu de *source* vers *destination* : *destination* ← *source*	*destination* et *source* doivent être de même taille (8 bits ou 16 bits). Sont interdits : le transfert mémoire, mémoire (MOV [100h],[1200h]) le transfert registre de segmentation, valeur immédiate (MOV DS,1200h)	MOV AX,5BC3h (registre 16 bits) MOV CL,4Eh (registre 8 bits) MOV AX,[1200h] (adressage direct) MOV [BX],CL (adressage basé) ...

2. Instructions arithmétiques

Syntaxe	Rôle	Remarques	Exemples

ADD *destination , source*	Effectue une addition entre *destination* et *source* :
	destination ← *destination* + *source*
		S'il y a une retenue le drapeau CF (Carry) est mis à 1.
		destination et *source* doivent être de même taille (8 bits ou 16 bits).
		Tous les registres sont utilisables sauf les registres de segment.
		l'addition mémoire, mémoire est interdite.
			ADD DX,1000h
			(addition sur 16 bits : DX←DX+1000h)
			ADD BH,F1h
			(addition sur 8 bits : BH←BH+F1h)
			ADD AL,[1205h] (adressage direct)
			ADD CL,[SI] (adressage indexé)
			...

ADC *destination , source*	Effectue une addition entre *destination* et *source* avec la retenue *CF* : *destination* ← *destination* + *source* + *CF*	Mêmes remarques que ADD	**ADC DX,300h** (DX←DX+300h + CF) Si DX=500h et CF=1 alors DX prendra la valeur 801h
INC *destination*	Incrémente *destination* destination ← destination + 1	*destination* est un registre de 8 ou 16 bits.	**INC DI**

Syntaxe	Rôle	Remarques	Exemples
SUB *destination , source*	Effectue une soustraction entre *destination* et *source* : *destination* ← *destination - source*	Mêmes remarques que ADD	**SUB CH,BL** (CH←CH-BL)
DEC *destination*	Décrémente *destination* destination ← destination - 1	*destination* est un registre de 8 ou 16 bits.	**DEC BH**
MUL *source*	Effectue une multiplication entre AX (ou AL) et *source* : ***Sur 8 bits :*** AX ← AL x *source* ***Sur 16 bits :*** DX:AX ← AX x *source*	***Multiplication sur 8 bits :*** *source* est un registre 8 bits ou une adresse mémoire ***Multiplication sur 16 bits :*** *source* est un registre 16 bits ou une adresse mémoire	**MUL DH** (AX←AL x DH)
DIV *source*	Effectue une division entre AX (ou DX:AX) et source : ***Sur 8 bits :*** AL ← AX / *source* (reste dans AH) ***Sur 16 bits :*** AX ← DX:AX / *source* (reste dans DX)	***Division sur 8 bits :*** *source* est un registre 8 bits ou une adresse mémoire ***Division sur 16 bits :*** *source* est un registre 16 bits ou une adresse mémoire	**DIV CL** (AL←AX / CL, reste dans AH)

3. Instructions logiques

Syntaxe	Rôle	Remarques	Exemples
AND *destination , source*	Effectue un ET logique entre *destination* et *source* : *destination* ← *destination* ET *source*	*destination* et *source* doivent être de même taille. Ils peuvent être des registres de 8 ou 16 bits ou des cases mémoire.	**AND DH,CL** Si DH=F4h et CL=3Fh alors : DH ←DH ET CL = 34h
OR *destination , source*	Effectue un OU entre *dest.* et *source* : *destination* ← *destination* OU *source*	même remarque que AND	**OR SI,BP**
XOR *destination , source*	Effectue un OU exclusif entre *destination* et *source* : *destination* ← *destination* ⊕ *source*	même remarque que AND	**XOR CH,AL** Si CH=42h et AL=9Ah alors : CH ← CH ⊕ AL = D8h
NOT *destination*	Effectue le complément de *destination* : *destination* ← $\overline{destination}$	*destination* peut être un registres de 8 ou 16 bits ou une case mémoire.	**NOT DL** Si DL=C1 alors : DL ← $\overline{DL}$ = 3Eh

4. Instructions de manipulation de bits

Syntaxe	Rôle	Remarques	Exemples
(1) **SHR** *destination , 1* (2) **SHR** *destination , CL*	Réalise un décalage à droite de *destination*	décalage d'un seul bit à droite décalage de n bits à droite tel que n est la valeur stockée dans CL	**SHR AL,CL** Si AL=7Ah et CL=4 alors AL←07h
(1) **SHL** *destination , 1* (2) **SHL** *destination , CL*	Réalise un décalage à gauche de *destination*	(1) décalage d'un seul bit à gauche (2) décalage de n bits à gauche (CL=n)	**SHL BL,1** Si BL=41h alors BL←82h
(1) **ROR** *destination , 1* (2) **ROR** *destination , CL*	Réalise un rotation à droite de *destination*	(1) rotation d'un seul bit à droite (2) rotation de n bits à droite (CL=n)	
(1) **ROL** *destination , 1* (2) **ROL** *destination , CL*	Réalise un rotation à gauche de *destination*	(1) rotation d'un seul bit à gauche (2) rotation de n bits à gauche (CL=n)	
(1) **RCR** *destination , 1* (2) **RCR** *destination , CL*	Réalise un rotation à droite de *destination* à travers le bit CF	(1) rotation d'un seul bit à droite (2) rotation de n bits à droite (CL=n)	
(1) **RCL** *destination , 1* (2) **RCL** *destination , CL*	Réalise un rotation à gauche de *destination* à travers CF	(1) rotation d'un seul bit à gauche (2) rotation de n bits à gauche (CL=n)	**RCL BL,CL** Si BL=72h, CL=2 et CF=0 alors : BL←C8h et CF←1

5. Instructions de branchement

Syntaxe	Rôle	Remarques	Exemples
JMP *Etiquette*	Effectue un saut inconditionnel à l'instruction marquée par *Etiquette*		
CALL *Etiquette*	Effectue un appel au sous-programme marqué par *Etiquette*.	Le sous-programme appelé doit obligatoirement se terminer par l'instruction de retour RET	
CMP *destination , source*	Effectue l'opération : *destination - source* et positionne les drapeaux du registre d'état.	*source* et *destination* ne sont pas modifiés Cette instruction précède en général un saut conditionnel	
JE *Etiquette*	Effectue un saut à *Etiquette* si égal		**CMP AL,5Ah** **JE suite** (saut à suite si AL=5Ah)
JNE *Etiquette*	Effectue un saut à *Etiquette* si différent		**CMP SI,3B00h** **JNE boucle** (saut à boucle si SI≠3B00h)
JL,JLE *Etiquette*	Effectue un saut à *Etiquette* si inférieur ou inférieur ou égal respectivement	La comparaison est effectuée en binaire signé	**CMP CL,30h** **JL fleur** (saut à fleur si CL<30h)

Syntaxe	Rôle	Remarques	Exemples
JG,JGE *Etiquette*	Effectue un saut à *Etiquette* si supérieur ou supérieur ou égal respectivement	La comparaison est effectuée en binaire signé	**CMP DH,FFh - JG fleur2** Si DH=2, il y aura saut car 2>-1 (FF en binaire signé)
JB,JBE *Etiquette*	Effectue un saut à *Etiquette* si inférieur ou inférieur ou égal respectivement	La comparaison est effectuée en binaire non signé	
JA,JAE *Etiquette*	Effectue un saut à *Etiquette* si supérieur ou supérieur ou égal respectivement	La comparaison est effectuée en binaire non signé	**CMP DH,FFh - JA fleur2** Si DH=2, il n'y aura pas de saut car 2<255 (FF en binaire non signé)
JZ,JC,JP,JS,JO *Etiquette*	Effectue un saut à *Etiquette* si le drapeau ZF, CF, PF, SF, OF respectivement est à 1		**ADD AL,14h** **JC Trai_Re** (Saut à Trai_Re en cas de retenue)
JNZ,JNC,JNP,JNS,JNO *Etiquette*	Effectue un saut à Etiquette si le drapeau ZF, CF, PF, SF, OF respectivement est à 0		**SUB AX,185Eh** **JNZ suite** (Saut à suite si le résultat de la soustraction n'est pas nul)

6. Instructions de contrôle et de gestion de la pile

Syntaxe	Rôle	Remarques	Exemples
CLC, CLD, CLI	Met à zéro les drapeaux CF, DF et IF respectivement		
STC, STD, STI	Met à un les drapeaux CF, DF et IF respectivement		
CMC	Complémente le bit de retenue CF		
NOP	N'effectue aucune opération	Instruction utilisée en général pour allonger la durée d'une boucle ou occuper un espace non utilisé.	
PUSH *source*	Empile *source*	*source* doit être un registre de 16 bits ou un pointeur vers une adresse. Le pointeur de pile SP est décrémenté de 2	**PUSH AX** PUSH [SI]
POP *destination*	Dépile le sommet de la pile vers *destination*	*destination* doit être un registre de 16 bits ou un pointeur vers une adresse. Le pointeur de pile SP est incrémenté de 2	**POP DS** POP [BX]

4.2. Les sauts

Ils sont également appelés des **branchements**, ils permettent de poursuivre l'exécution du programme à un point spécifique *avec* ou *sans* condition préalable. Les sauts sont classés en deux catégories :

- Les sauts inconditionnels,
- Les sauts conditionnels.

4.2.1. Notion d'étiquette

Quand on écrit un programme en langage assembleur, on peut désigner certaines instructions par des **étiquettes** (labels). Une étiquette joue le rôle de référence lorsqu'on désire effectuer un saut à une instruction marquée.

Une étiquette peut être n'importe quel texte suivi de deux points (**:**). Elle est mise juste avant l'instruction à marquer.

Exemple :

```
DEBUT :     MOV  AX, 1000h
            MOV  DS, AX
TestRS :    MOV  AH, 01
            INT   21H
            XOR  AH, AH
            MOV  BL, OAH
Suite1 :    DIV   BL
            JZ   TestRS
```

DEBUT, *TestRS* et *Suite1* sont des étiquettes. La dernière ligne du programme fait référence à l'étiquette *TestRS* par l'instruction de saut conditionnel **JZ**.

4.2.2. Les sauts inconditionnels

Ces sauts ont lieu après l'exécution d'une instruction de saut sans respect préalable d'une quelconque condition.

L'instruction JMP

Syntaxe : JMP Etiquette

Dès que le microprocesseur rencontre cette instruction, il continue l'exécution à partir de l'instruction indiquée par « *Etiquette* ». L'exécution se poursuit normalement et aucune mémorisation de l'endroit d'appel n'est faite.

Exemple :

```
            MOV  CH, AL
            MOV  CL, 4
            SHR   AL, CL
            JMP   Suite
            MOV  AL, CH
Suite :     AND  AL, 0FH
            XOR DX, DX
```

Dans cet exemple, l'instruction MOV AL,CH sera ignorée. L'instruction exécutée après SHR AL,CL sera AND AL,0Fh.

L'instruction CALL

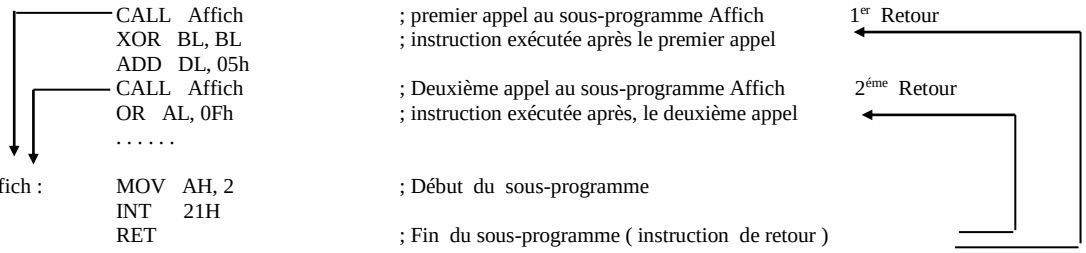
Syntaxe : CALL Etiquette

Cette instruction permet d'appeler un sous-programme.

Un sous-programme doit être terminé par l'instruction de retour **RET** et sa première instruction doit être désignée par une **étiquette** qui servira de référence d'appel.

Exemple :

```
        MOV  DL,  41h
        CALL   Affich      ; premier appel au sous-programme Affich      1er Retour
        XOR   BL, BL       ; instruction exécutée après le premier appel
        ADD   DL, 05h
        CALL   Affich      ; Deuxième appel au sous-programme Affich     2éme Retour
        OR   AL, 0Fh       ; instruction exécutée après, le deuxième appel
        . . . . . .
Affich : MOV   AH, 2       ; Début  du  sous-programme
        INT     21H
        RET                ; Fin  du sous-programme ( instruction  de retour )
```



Remarque

L'adresse de retour est mémorisée dans la pile au moment de l'appel. En arrivant à la fin du sous-programme (instruction RET), le microprocesseur récupère l'adresse de retour de la pile et continue l'exécution à cette adresse (pour plus de détails, voir chapitre : la pile).

4.2.3. Les sauts conditionnels

Les sauts conditionnels ont lieu si une condition est vérifiée. Ils dépendent en réalité des différents drapeaux du registre d'état (CF , ZF, PF ,...).

L'instruction CMP

Syntaxe : CMP destination, source

CMP réalise une soustraction entre les deux opérandes : destination – source. En fonction du résultat de cette opération, les drapeaux du registre d'état sont positionnés. Les valeurs de « destination » et « source » restent inchangées.

Exemple :

```
CMP AL, 50h
```

Si AL = 50h, la soustraction donne un résultat nul. Par conséquent, ZF sera mis à 1. Ainsi, en examinant ZF après cette instruction, on peut vérifier si AL = 50h ou non.

L'instruction TEST

Syntaxe : TEST destination, source

TEST effectue un **ET** logique entre les deux opérandes : destination AND source. Les drapeaux sont positionnés en fonction du résultat de cette opération. Les valeurs de « destination » et « source » restent inchangées.

Exemple :

```
TEST BL, BL
```

- Si BL = 00h, le ET donnera un résultat nul, par conséquent, ZF sera mis à 1.
- Si BL ≠ 00h, le résultat du ET sera non nul, donc ZF sera mis à 0.

Donc, pour vérifier si BL est nul ou non, il suffit de tester l'état de ZF juste après cette instruction.

5. CODAGE DES INSTRUCTIONS

Les instructions et leurs opérandes sont stockés en mémoire principale. La taille du code d'une instruction dépend du type de l'instruction ainsi que de la taille de l'opérande et du mode d'adressage utilisé.

Une instruction se compose en général de deux champs :

- Le code opération qui représente l'instruction,
- Le champ opérande qui représente la donnée ou la référence à la donnée.

Code opération	Opérande

(1 ou 2 octets)	(0, 1, 2,... octets)

Exemples
- MOV AX, 125Eh est codée sur 3 octets : B8 5E 12 (B8 : code opération MOV AX ; 5E 12 : opérande donné en immédiat).
- INC AX est codée sur un seul octet : 40.

6. LA PILE ET SES UTILISATIONS

6.1. Structure de la pile

La pile est une zone mémoire gérée en **LIFO** (Last In, First Out). Afin de la repérer dans la mémoire, le registre SP (avec SS comme registre de segment) pointe en permanence sur le sommet de la pile.

La pile est une zone mémoire qui doit être réservée pour le programme (gestion des appels aux sous-programmes et aux interruptions, …), et pour d'autres usages généraux (sauvegarde de données temporaires, passage de paramètres à une procédure, …)

6.2. Accès à la pile : l'empilement

L'empilement d'une donnée est réalisé par l'instruction :

PUSH donnée

Où « **donnée** » est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect.

L'empilement s'effectue en deux étapes :

- SP ← SP – 2 : on fait reculer SP de deux cases (donnée sur 16 bits), pour qu'il pointe sur le nouveau sommet susceptible de recevoir la donnée à empiler.
- [SP] ← donnée : le sommet de la pile reçoit la donnée.

6.3. Accès à la pile : le dépilement

Le dépilement est l'opération qui permet de retirer le sommet de la pile et le stocker dans une destination. Il est réalisé par l'instruction :

POP destination

« destination » est un registre 16 bits ou un emplacement mémoire désigné par adressage direct ou indirect. Le dépilement engendre deux opérations :

- destination ← [SP] : le sommet de la pile est copié dans destination.
- SP ← SP + 2 : la donnée du sommet étant retirée, on fait avancer SP pour pointer sur la donnée suivante.

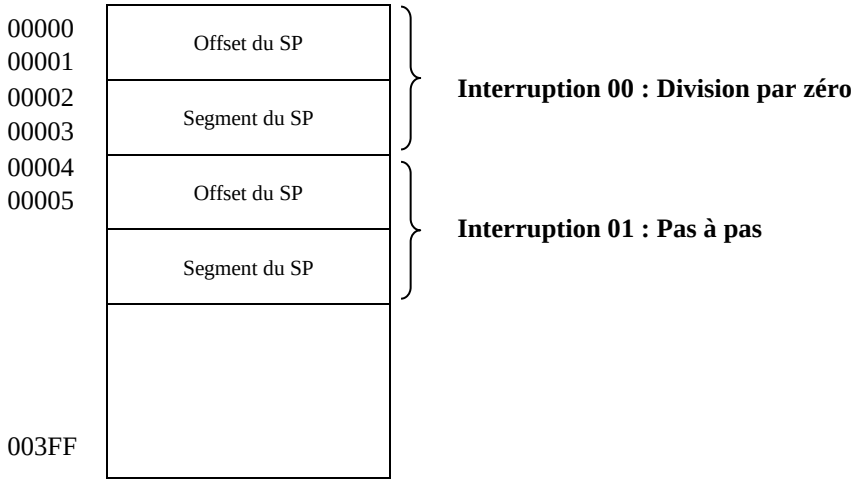
7. LES INTERRUPTIONS

Elles sont déclenchées par l'instruction **INT n** ou n désigne le numéro d'interruption. Les sous-programmes associés aux interruptions logicielles sont en général des routines du BIOS ou du DOS. Elles permettent au programme d'accéder aux différents périphériques de l'ordinateur.

Exemple

Interruption 10h : interruption BIOS (gestion de l'affichage),
Interruption 21h : interruption DOS (gestion des périphériques, des fichiers, …).

La table des vecteurs d'interruption contient l'adresse des sous-programmes de toutes les interruptions du microprocesseur (256 au total) et commence à partir de 00000. L'adresse d'un sous-programme d'interruption prend 4 octets : 2 octets pour son segment et 2 octets pour son offset :



La table occupe l'espace mémoire entre 00000h et 003FFh. Pour une interruption logicielle **n**, l'adresse du sous-programme associé se trouve à **4n** (offset du Sous-Programme) et **4n+2** (segment du S-P).

Attention :
Il est interdit de modifier directement le contenu de la table des vecteurs d'interruption. Cette opération se fait à l'aide de la fonction 25h de l'interruption 21h.

1. STRUCTURE D'UN PROGRAMME EN ASSEMBLEUR 80X86

Un programme en assembleur se compose en général de 3 parties définissant les différents segments :

1. le segment de code (pointé par CS),
2. le segment de données (pointé par DS),
3. le segment de pile (pointé par SS).

1.1. Exemple de programme

Le programme suivant permet de remplir un tableau par une suite de nombres de 0 à 10 :

```
Pile      SEGMENT      PARA  STACK        ; Définition du segment de pile
          DB  256  DUP  ('P')             ; Pile de 256 octets remplie par
Pile      ENDS                            ; le caractère  'P'

Donnees   SEGMENT      PARA               ; Définition du segment de données
          Table   DB  11  DUP  (00)        ; Table de 11 octets initialisée par 0
Donnees   ENDS

Code      SEGMENT      PARA               ; Définition du segment de code
          ASSUME  CS   : Code             ; Affectation des registres
          ASSUME  DS   : Donnees          ; de segments
          ASSUME  SS   : Pile

Debut :   MOV    AX, Donnees              ; Initialisation du registre DS
          MOV    DS,AX
LEA       BX, Table                  ; BX reçoit l'adresse logique de table
          XOR    AL,AL                    ; AL = 0
Boucle :  MOV  Byte  Ptr[BX], AL          ; Byte Ptr [BX] veut dire octet …
          INC    BX                       ; … pointé  par  BX
          INC    AL
          CMP    AL,0Bh                   ; Teste la fin de la boucle
          JNE    Boucle                   ; AL  #  0Bh        aller à Boucle
          MOV    AH,   4Ch                ; Appel à la fonction  4C de
          INT    21h                      ; l'interruption 21h (Retour au DOS)
Code      ENDS
END       Debut                            ; END indique la fin du programme
                                           ; Debut désigne le label de la
                                           ; première instruction du programme
```

Remarque
Les registres CS et SS sont automatiquement affectés aux segments *Code* et *Pile* respectivement. Par contre, le registre DS doit être initialisé dans le programme.

1.2. Déclaration des segments

La déclaration d'un segment se fait selon la syntaxe suivante :

```
Nom_segment    SEGME NT      PARA    ;début du segment
        << **** Instructions ou données appartenant au segment ***>>>
Nom_segment ENDS                       ; fin du programme
```

1.3. Déclaration des données et de la pile

Le segment de données est réservé aux données que le programme utilise pendant son exécution. Ces données sont représentées sous forme d'octets, de mots, de tableaux ou de chaînes de caractères et peuvent être traitées comme des constantes ou des variables.

1.3.1. Définition d'un octet :

```
Nombre  DB   18h
  ↑          ↑
Label      Valeur initiale
```

Les valeurs initiales peuvent être exprimées en décimale, en hexadécimal (h) ou en binaire (b). Le compilateur se charge de faire la conversion.

1.3.2. Définition d'un mot (16 bits)

```
Prod     DW    5C0Fh
  ↑             ↑
Label        valeur initiale
```

1.3.3. Définition d'un tableau d'octets

```
Tab      DB      15h, 5Ah,  21h,  82h
  ↑                ↖
Label marquant le début du tableau      Données appartenant au tableau
```

- Pour définir un tableau de 10 octets sans initialisation, on écrit :

```
Tab   DB   10   Dup   ( ? )
```

- Pour définir un tableau de 7 octets contenant des valeurs identiques (2Ah par exemple), on écrit :

```
Tab  DB   7   Dup   (2Ah)
```

1.3.4. Définition d'une chaîne de caractères

```
MaChn    DB     ' Ceci est un exemple '
Chn2     DB     ' FIN ' , 0Dh , 0Ah
```

Une chaîne de caractères est stockée sous forme d'octets représentant le code ASCII de chaque caractère. La chaîne est écrite entre guillemets et c'est le compilateur qui se charge de convertir chaque caractère en code ASCII.

1.3.5. Déclaration du segment de données

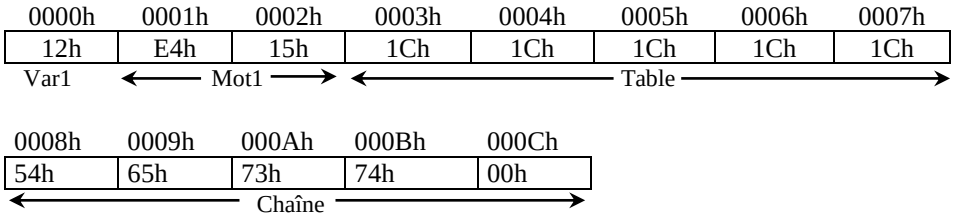
La syntaxe est identique à celle du segment du code. Dans un segment de données on regroupe toutes les données utilisées par le programme.

Les variables déclarées sont placées l'une après l'autre dans la mémoire. Chaque variable prend l'espace mémoire nécessaire.

Exemple

```
Données    SEGMENT    PARA    ; Début du segment
           Var1     DB       12h                    ; 1 octet
           Mot1     DW       15E4h                  ; 2 octets (mot de 16 bits)
           Table    DB       5 DUP (1Ch)            ; 5 octets (taille du tableau)
           Chaîne   DB       ' Test ' , 0           ; 5 octets (longueur de la chaîne)
Données    ENDS                                    ; Fin du segment
```

Si le premier octet du segment de données se trouve à l'offset 0000h, le contenu de la mémoire sera :



Les labels qui identifient chaque variable reçoivent l'offset de celle-ci dans le segment de données :

Label	**offset**	**attribué**
Var1	0000h	
Mot1	0001h	
Table	0003h	
Chaîne	0008h	

1.3.6. Déclaration du segment de pile

Dans la plupart des cas, la déclaration d'un segment de pile est obligatoire surtout en présence de sous-programmes.

La taille de la pile varie selon les applications. Un programme récursif aura besoin d'une pile de taille beaucoup plus importante qu'un programme ordinaire.

La réservation d'une zone mémoire pour le segment de pile se fait de la manière suivante :

```
MaPile  SEGMENT     PARA  STACK
        DB  256  DUP  ( ? )
MaPile  ENDS
```

Ici, la taille de la pile est de 256 octets. Ce qui est suffisant pour des petits programmes en assembleur.

1.4. Appel aux fonctions du DOS et du DIOS

L'appel des fonctions du DOS et du BIOS se fait par le biais des interruptions logicielles. Avant d'appeler l'interruption, on charge les éventuels paramètres dans les registres du microprocesseur. Certaines fonctions renvoient des résultats après l'exécution de l'interruption. Ceux-ci sont en général stockés dans les registres du microprocesseur.

Comme le nombre des fonctions disponibles est très grand, il n'est pas possible de donner des explications détaillées dans le cadre de ce document (voir à ce sujet les ouvrages spécialisés).

Exemple : Affichage d'un caractère sur l'écran
Interruption : 21h
Paramètres :

- AH = 02h (fonction n°2)
- DL = Code ASCII du caractère à afficher

Le programme suivant permet d'afficher le caractère "A" sur l'écran :

```
MOV   DL,41h        ; Code ASCII du caractère "A"
MOV   AH,2          ; Fonction n°2
INT   21h           ; Appel de l'interruption 21h
```