Chapitre III : Introduction au langage machine et assembleur

Nous allons étudier la programmation en langage machine et en assembleur d'un microprocesseur. Nous allons ici nous limiter à un sous-ensemble du microprocesseur 80486 (seuls les registres et les instructions les plus simples seront étudiés). De cette façon, nous pourrons tester sur un PC les programmes en langage machine que nous écrirons.

L'étude complète d'un processeur réel, comme le 80486 ou le Pentium fabriqués par Intel, dépasse largement le cadre de ce cours : le nombre d'instructions et de registres est très élevé.

3.1 Caractéristiques du processeur étudié

La gamme de microprocesseurs 80x86 équipe les micro-ordinateurs de type PC et compatibles.

Voici les caractéristiques du processeur simplifié que nous étudierons :

CPU 16 bits à accumulateur :

– bus de données 16 bits ;
– bus d'adresse 32 bits ;

Registres :

– accumulateur AX (16 bits) ;
– registres auxiliaires BX et CX (16 bits) ;
– pointeur d'instruction IP (16 bits) ;
– registres segments CS, DS, SS (16 bits) ;
– pointeur de pile SP (16 bits), et pointeur BP (16 bits).

3.2 Jeu d'instruction

3.2.1 Types d'instructions

- ✓ **Instructions d'affectation**

Déclenchent un transfert de données entre l'un des registres du processeur et la mémoire principale.

– transfert CPU ← Mémoire Principale (MP) (= lecture en MP) ;
– transfert CPU → Mémoire Principale (MP) (= écriture en MP) ;

- ✓ **Instructions arithmétiques et logiques**

Opérations entre une donnée et l'accumulateur AX. Le résultat est placé dans l'accumulateur. La donnée peut être une constante ou une valeur contenue dans un emplacement mémoire.
Exemples :

– addition : AX ← AX + donnée ;
– soustraction : AX ← AX - donnée ;
– incrémentation de AX : AX ← AX + 1 ;
– décrémentation : AX ← AX - 1 ;
– décalages à gauche et à droite ;

- ✓ **Instructions de comparaison**

Comparaison du registre AX à une donnée et positionnement des indicateurs.

- ✓ **Instructions de branchement**

La prochaine instruction à exécuter est repérée en mémoire par le registre IP. Les instructions de branchement permettent de modifier la valeur de IP pour exécuter une autre instruction (boucles, tests, etc.).
On distingue deux types de branchements :
– *branchements inconditionnels* : IP adresse d'une instruction ;
– *branchements conditionnels* : Si une condition est satisfaite, alors branchement, sinon passage simple à l'instruction suivante.

3.2.2 Codage des instructions et mode d'adressage

Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale.
La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande. Chaque instruction est toujours codée sur un nombre entier d'octets, afin de faciliter son décodage par le processeur.
Une instruction est composée de deux champs :
– le code opération, qui indique au processeur quelle instruction réaliser ;
– le champ opérande qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

champ code opération	champ code opérande

Selon la manière dont la donnée est spécifiée, c'est à dire selon le mode d'adressage de la donnée, une instruction sera codée par 1, 2, 3 ou 4 octets.
Nous distinguerons ici quatre modes d'adressage : implicite, immédiat, direct et relatif (nous étudierons plus tard un autre mode, l'adressage indirect).

Adressage implicite

L'instruction contient seulement le code opération, sur 1 ou 2 octets.

code opération (1 ou 2 octets)

L'instruction porte sur des registres ou spécifie une opération sans opérande (exemple : "incrémenter AX").

Adressage immédiat

Le champ opérande contient la donnée (une valeur constante sur 1 ou 2 octets).

code opération (1 ou 2 octets)	valeur (1 ou 2 octets)

Exemple : "Ajouter la valeur 5 à AX". Ici l'opérande 5 est codé sur 2 octets.

Adressage direct

Le champ opérande contient l'*adresse* de la donnée en mémoire principale sur 2 octets.

code opération (1 ou 2 octets)	adresse de la donnée (2 octets)

Attention : dans le 80x86, les adresses sont toujours manipulées sur 16 bits, quelle que soit la taille réelle du bus.

Adressage relatif
Ce mode d'adressage est utilisé pour certaines instructions de branchement. Le champ opérande contient un entier relatif codé sur 1 octet, nommé *déplacement*, qui sera ajouté à la valeur courante de IP.

code opération (1 octet)	déplacement (1 octet)

3.2.3 Temps d'exécution
Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer.
Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage.
La durée d'un cycle dépend bien sur de la fréquence d'horloge de l'ordinateur. Plus l'horloge bat rapidement, plus un cycle est court et plus on exécute un grand nombre d'instructions par seconde.

3.2.4 Ecriture des instructions en langage symbolique
Voici un programme en langage machine 80486, implanté à l'adresse 0100H :

A1 01 10 03 06 01 12 A3 01 14

Ce programme additionne le contenu de deux cases mémoire et range le résultat dans une troisième. Nous avons simplement transcrit en hexadécimal le code du programme. Il est clair que ce type d'écriture n'est pas très utilisable par un être humain.
A chaque instruction que peut exécuter le processeur correspond une représentation binaire sur un ou plusieurs octets, comme on l'a vu plus haut. C'est le travail du processeur de décoder cette représentation pour effectuer les opérations correspondantes.
Afin de pouvoir écrire (et relire) des programmes en langage machine, on utilise une notation symbolique, appelée *langage assembleur*. Ainsi, la première instruction du programme ci-dessus (code A1 01 10) sera notée :

MOV AX, [0110]

elle indique que le mot mémoire d'adresse 0110H est chargé dans le registre AX du processeur.
On utilise des programmes spéciaux, appelés *assembleurs*, pour traduire automatiquement le langage symbolique en code machine.
Voici une transcription langage symbolique du programme complet. L'adresse de début de chaque instruction est indiquée à gauche (en hexadécimal).

```
Adresse  Contenu MP   Langage Symbolique   Explication en francais

0100     A1 01 10     MOV AX, [0110]       Charger AX avec le contenu
                                            de 0110.
0103     03 06 01 12  ADD AX, [0112]       Ajouter le contenu de 0112
                                            a AX (resultat dans AX).
0107     A3 01 14     MOV [0114], AX       Ranger AX en 0114.
```

- ✓ **Sens des mouvements de données**

La plupart des instructions spécifient des mouvements de données entre la mémoire principale et le microprocesseur. En langage symbolique, on indique toujours la destination, puis la source. Ainsi l'instruction

MOV AX, [0110]

transfère le contenu de l'emplacement mémoire 0110H dans l'accumulateur, tandis que MOV [0112], AX transfère le contenu de l'accumulateur dans l'emplacement mémoire 0112.

L'instruction MOV (de l'anglais move, déplacer) s'écrit donc toujours :

MOV destination, source

- ✓ **Modes d'adressage**

– En *adressage immédiat*, on indique simplement la valeur de l'opérande en hexadécimal. Exemple :

MOV AX, 12

– En *adressage direct*, on indique l'adresse d'un emplacement en mémoire principale en hexadécimal entre crochets :

MOV AX, [A340]

– En *adressage relatif*, on indique simplement l'adresse (hexa). L'assembleur traduit automatiquement cette adresse en un déplacement (relatif sur un octet). Exemple :

JNE 0108

Retour au DOS

A la fin d'un programme en assembleur, on souhaite en général que l'interpréteur de commandes du DOS reprenne le contrôle du PC. Pour cela, on utilisera la séquence de deux instructions :

MOV AH, 4C

INT 21

3.2.5 Utilisation du programme *debug*

debug est un programme qui s'exécute sur PC (sous DOS) et qui permet de manipuler des programmes en langage symbolique. Il est normalement distribué avec toutes les versions du système MS/DOS. Nous l'utiliserons en travaux pratiques.

Les fonctionnalités principales de *debug* sont les suivantes :

– Affichage du contenu d'une zone mémoire en hexadécimal ou en ASCII ;
– Modification du contenu d'une case mémoire quelconque ;
– Affichage en langage symbolique d'un programme ;
– Entrée d'un programme en langage symbolique ; *debug* traduit les instructions en langage machine et calcule automatiquement les déplacements en adressage relatif.
– Affichage et modification de la valeur des registres du processeur ;

3.3 Branchements

Normalement, le processeur exécute une instruction puis passe à celle qui suit en mémoire, et ainsi de suite séquentiellement. Il arrive fréquemment que l'on veuille faire répéter au processeur une certaine suite d'instructions, comme dans le programme :

Repeter 3 fois:
ajouter 5 au registre BX

Tableau des instructions
Ce tableau donne la liste de quelques instructions importantes du 80x86.

Symbole	Code Op.	Octets	
MOV AX, *valeur*	B8	3	AX ←*valeur*
MOV AX, [*adr*]	A1	3	AX ← contenu de l'adresse *adr*.
MOV [*adr*], AX	A3	3	range AX à l'adresse *adr*.
ADD AX, *valeur*	05	3	AX ← AX + *valeur*
ADD AX, [*adr*]	03 06	4	AX ← AX + contenu de *adr*.
SUB AX, *valeur*	2D	3	AX ← AX - *valeur*
SUB AX, [*adr*]	2B 06	4	AX ← AX - contenu de *adr*.
SHR AX, 1	D1 E8	2	décale AX à droite.
SHL AX, 1	D1 E0	2	décale AX à gauche.
INC AX	40	1	AX← AX + 1
DEC AX	48	1	AX ← AX - 1
CMP AX, *valeur*	3D	3	compare AX et *valeur*.
CMP AX, [*adr*]	3B 06	4	compare AX et contenu de *adr*.
JMP *adr*	EB	2	saut inconditionnel (adr. relatif).
JE *adr*	74	2	saut si =
JNE *adr*	75	2	saut si ≠
JG *adr*	7F	2	saut si >
JLE *adr*	7E	2	saut si ≤
JA *adr*			saut si CF = 0
JB *adr*			saut si CF = 1
Fin du programme (retour au DOS) :			
MOV AH, 4C INT 21	B4 4C CD 21	2 2	

TAB. 3.1 – Quelques instructions du 80x86. Le code de l'instruction est donné en hexadécimal dans la deuxième colonne. La colonne suivante précise le nombre d'octets nécessaires pour coder l'instruction complète (opérande inclus). On note *valeur* une valeur sur 16 bits, et *adr* une adresse sur 16 bits également.

Pour modifier le déroulement normal d'un programme, il suffit que l'exécution de l'instruction modifie la valeur de IP. C'est ce que font les instructions de branchement.
On distingue deux catégories de branchements, selon que le saut est toujours effectué (sauts *inconditionnels*) ou qu'il est effectué seulement si une condition est vérifiée (sauts *conditionnels*).

3.3.1 Saut inconditionnel

La principale instruction de saut inconditionnel est JMP. En adressage relatif, l'opérande de JMP est un *déplacement*, c'est à dire une valeur qui va être ajoutée à IP.

L'action effectuée par JMP est :

IP = IP + déplacement

Le déplacement est un entier relatif sur codée 8 bits. La valeur du déplacement à utiliser pour atteindre une certaine instruction est :

déplacement = adr. instruction visée - adr. instruction suivante

Exemple : le programme suivant écrit indéfiniment la valeur 0 à l'adresse 0140H. La première instruction est implantée à l'adresse 100H.

```
Adresse  Contenu MP   Langage Symbolique   Explication en francais

0100     B8 00 00     MOV AX, 0            met AX a zero
0103     A3 01 40     MOV [140], AX        ecrit a l'adresse 140
0106     EB FC        JMP 0103             branche en 103
0107                  xxx -> instruction jamais executee
```

Le déplacement est ici égal à FCH, c'est à dire -4 (=103H-107H).

3.3.2 Indicateurs

Les instructions de branchements conditionnels utilisent les *indicateurs*, qui sont des bits spéciaux positionnés par l'UAL après certaines opérations. Les indicateurs sont regroupés dans le *registre d'état* du processeur. Ce registre n'est pas accessible globalement par des instructions ; chaque indicateur est manipulé individuellement par des instructions spécifiques.

Nous étudierons ici les indicateurs nommés ZF, CF, SF et OF.

ZF *Zero Flag*

Cet indicateur est mis à 1 lorsque le résultat de la dernière opération est zéro.
Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF=1 indique que les deux opérandes étaient égaux. Sinon, ZF est positionné à 0.

CF *Carry Flag*

C'est l'indicateur de report (retenue), qui intervient dans les opérations d'addition et de soustractions sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP.
CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes.

SF *Sign Flag*

SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0. SF est utile lorsque l'on manipule des entiers relatifs, car le bit de poids fort donne alors le signe du résultat.

OF *Overflow Flag*

Indicateur de débordement OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable *en relatif* dans l'accumulateur (par exemple si l'addition de 2 nombres positifs donne un codage négatif).

Lorsque l'UAL effectue une addition, une soustraction ou une comparaison, les quatre indicateurs sont positionnés. Certaines autres instructions que nous étudierons plus loin peuvent modifier les indicateurs.

- ✓ **Instruction CMP**

Il est souvent utile de tester la valeur du registre AX sans modifier celui-ci. L'instruction CMP effectue exactement les même opération que SUB, mais ne stocke pas le résultat de la soustraction. Son seul effet est donc de positionner les indicateurs.
Exemple : après l'instruction

```
CMP AX, 5
```

on aura ZF = 1 si AX contient la valeur 5, et ZF = 0 si AX est différent de 5.

- ✓ **Instructions STC et CLC**

Ces deux instructions permettent de modifier la valeur de l'indicateur CF.

Symbole	
STC	CF ← 1 (*SeT Carry*)
CLC	CF ← 0 (*CLear Carry*)

3.3.3 Sauts conditionnels

Les instructions de branchements conditionnels effectuent un saut (comme JMP) si une certaine condition est vérifiée. Si ce n'est pas le cas, le processeur passe à l'instruction suivante (l'instruction ne fait rien).
Les conditions s'expriment en fonction des valeurs des indicateurs. Les instructions de branchement conditionnel s'utilisent en général immédiatement après une instruction de comparaison CMP.
Voici la liste des instructions de branchement les plus utiles :

JE *Jump if Equal* saut si ZF = 1 ;
JNE *Jump if Not Equal* saut si ZF = 0 ;
JG *Jump if Greater* saut si ZF = 0 et SF = OF ;
JLE *Jump if Lower or Equal* saut si ZF=1 ou SF6=OF ;
JA *Jump if Above* saut si CF=0 et ZF=0 ;
JBE *Jump if Below or Equal* saut si CF=1 ou ZF=1.
JB *Jump if Below* saut si CF=1.

Note : les instructions JE et JNE sont parfois écrites JZ et JNZ (même code opération).

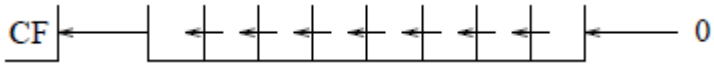
3.4 Instructions Arithmétiques et logiques

Les instructions arithmétiques et logiques sont effectuées par l'UAL. Nous avons déjà vu les instructions d'addition et de soustraction (ADD, SUB). Nous abordons ici les instructions qui travaillent sur la représentation binaire des données : décalages de bits, opérations logiques bit à bit.
Notons que toutes ces opérations modifient l'état des indicateurs.

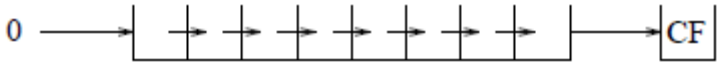
3.4.1 Instructions de décalage et de rotation

Ces opérations décalent vers la gauche ou vers la droite les bits de l'accumulateur.
Elles sont utilisées pour décoder bit à bit des données, ou simplement pour diviser ou multiplier rapidement par une puissance de 2. En effet, décaler AX de n bits vers la gauche revient à le multiplier par 2^n (sous réserve qu'il représente un nombre naturel et qu'il n'y ait pas de dépassement de capacité). De même, un décalage vers la droite revient à diviser par 2^n.
Voici les variantes les plus utiles de ces instructions. Elles peuvent opérer sur les registres AX ou BX (16 bits) ou sur les registres de 8 bits AH, AL, BH et BL.

SHL *registre*, 1 *(Shift Left)*
Décale les bits du registre d'une position vers la gauche. Le bit de gauche est transféré dans l'indicateur CF. Les bits introduits à droite sont à zéro.

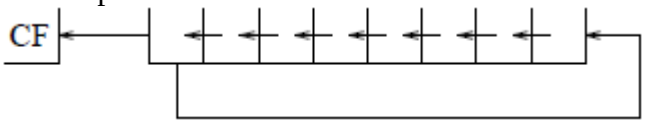


SHR *registre*, 1 *(Shift Right)*
Comme SHL mais vers la droite. Le bit de droite est transféré dans CF.

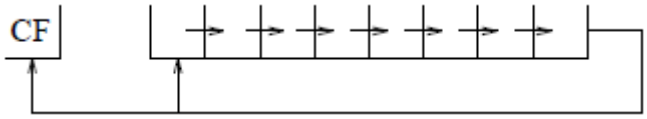


SHL et SHR peuvent être utilisé pour multiplier/diviser des entiers *naturels* (et non des relatifs car le bit de signe est perdu5).

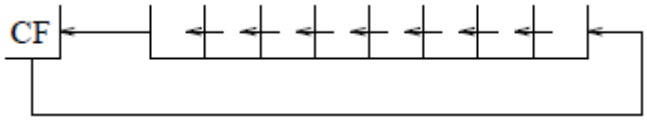
ROL *registre*, 1 *(Rotate Left)*
Rotation vers la gauche : le bit de poids fort passe à droite, et est aussi copié dans CF. Les autres bits sont décalés d'une position.



ROR *registre*, 1 *(Rotate Right)*
Comme ROL, mais à droite.



RCL *registre*, 1 *(Rotate Carry Left)*
Rotation vers la gauche en passant par l'indicateur CF. CF prend la place du bit de poids faible ; le bit de poids fort part dans CF.



RCR *registre*, 1 *(Rotate Carry Right)*
Comme RCL, mais vers la droite.

3.4.2 Instructions logiques

Les instructions logiques effectuent des opérations logiques bit à bit. Si on dispose de trois opérateurs logiques : ET, OU et OU exclusif, il n'y aura jamais de propagation de retenue lors des opérations (chaque bit du résultat est calculé indépendamment des autres).
Exemple : instructions OR

OR *destination, source*.

destination désigne le registre ou l'emplacement mémoire (adresse) où doit être placé le résultat. *source* désigne une constante (adressage immédiat), un registre (adressage implicite), ou une adresse (adressage direct).
Exemples :

```
OR    AX, FF00    ; AX <-  AX ou FFOO
OR    AX, BX      ; AX <-  AX ou BX
OR    AX, [1492]  ; AX <-  AX ou [1492]
```

OR *destination, source* *(OU)*
OU logique. Chaque bit du résultat est égal à 1 si au moins l'un des deux bits opérande est 1.
OR est souvent utilisé pour forcer certains bits à 1. Par exemple après OR AX, FF00, l'octet de poids fort de AX vaut FF, tandis que l'octet de poids faible est inchangé.

AND *destination, source* *(ET)*
ET logique. Chaque bit du résultat est égal à 1 si les deux bits opérandes sont à 1.
AND est souvent utilisé pour forcer certains bits à 0. Après AND AX, FF00, l'octet de poids faible de AX vaut 00, tandis que l'octet de poids fort est inchangé.

XOR *destination, source* *(OU EXCLUSIF)*
OU exclusif. Chaque bit du résultat est égal à 1 si l'un ou l'autre des bits opérandes (mais *pas les deux*) vaut 1.
XOR est souvent utilisé pour inverser certains bits. Après XOR AX, FFFF, tous les bits de AX sont inversés.