Une introduction à l'assembleur

1. Introduction générale

L'assembleur permet de contrôler directement la CPU. Cela permet d'avoir une total maîtrise du système et surtout permet de faire des programmes rapides par rapport aux langages de haut niveau (C++, Basic, ...). En effet, bien que ces langages permettent de faire des programmes facilement et rapidement, ils n'optimisent pas le code d'exécution. Cela engendre donc des programmes (beaucoup) plus volumineux. Notons que l'on peut insérer de l'assembleur dans certain langage (Pascal et C par exemple) pour accélérer l'exécution du programme. Mais avant d'aller plus loin, rappelons brièvement la fonction de l’unité centrale (CPU).

Comme nous l’avons vu dans le précédent chapitre, la CPU (Central Processing Unit) charge, analyse et exécute les instructions présentes en mémoire de façon séquentielle, c'est-à-dire une instruction à la suite de l'autre. Elle contient une unité de calculs arithmétiques et logiques (UAL), d'un bus interne, d'un bus externe se connectant au système, d'un décodeur d'instructions qui décode l'instruction en cours, et des registres pour mémoriser des résultats temporairement.

Ce sont les registres qui permettent la communication entre le programme et la CPU. Ils sont 'l'interface' de la CPU. En effet, pratiquement, toutes les données qui passent par la CPU, pour être traitées par celle-ci, doivent se trouver dans les registres de cette dernière. Ces cases mémoire sont les plus rapides de tout le système.

Il existe différents types de registres dans une CPU: les registres de traitements, les registres d'adressages et les registres d’état. Les Registres de traitement sont des registres destinés au traitement des valeurs contenues dans celle-ci; par exemple on peut effectuer une addition d'un registre de traitement avec un autre registre, effectuer des multiplications ou des traitements logiques. Les Registres d'adressage permettent de pointer un endroit de la mémoire; ils sont utilisés pour lire ou écrire dans la mémoire. Les registres d’état (ou volet : FLAG en anglais) sont de petits registres (de 1 Bit) indiquant l'état du processeur et 'le résultat' de la dernière instruction exécutée. Les plus courants sont le Zero Flag (ZF) qui indique que le résultat de la dernière opération est égale a zéro (après une soustraction par exemple), le Carry Flag (CF) qui indique qu'il y a une retenue sur la dernière opération effectuée, Overflow Flag (OF) qui indique un dépassement de capacité de registre, etc.

Pour gérer ces registres, on fait appel aux instructions du processeur. Ces instructions permettent d'effectuer des tâches très simples sur les registres. Elles permettent de mettre des valeurs dans les registres, d'effectuer des traitements logiques, des traitements arithmétiques, des traitements de chaîne de caractères, etc.

Ces instructions sont formées à l'aide du code binaire dans le programme. Or pour nous, ils est beaucoup plus facile d'utiliser des symboles à la place de ces codes binaires. C'est pour cela que l'on fait appel au compilateur qui permet de transformer un programme assembleur fait avec des mots clés compréhensibles pour nous (mais incompréhensible pour la machine), en un programme exécutable compréhensible par le processeur.

Ces mots clés, ou mnémoniques, sont souvent la compression d'un mot ou d'une expression en anglais présentant l'action de l'instruction. Par exemple sur les processeurs 8086 l'instruction MUL permet la multiplication (MULtiply), sur Z80 l'instruction LD permet de charger une valeur dans un registre ou dans la mémoire (Load)

MCours.com

Les instructions sont souvent suivies d'opérandes permettant d'indiquer sur quel(s) registre(s) on veut effectuer le traitement, quelle valeur on veut utiliser.

Exemple: Pour additionner 2 registres (2 registres 16 bits AX et BX) sur 8086: ADD AX,BX
Cette instruction correspond en gros a: AX=AX+BX

La même chose sur 68000: ADD.W D1,D0
Cette instruction correspond en gros a D0=D0+D1

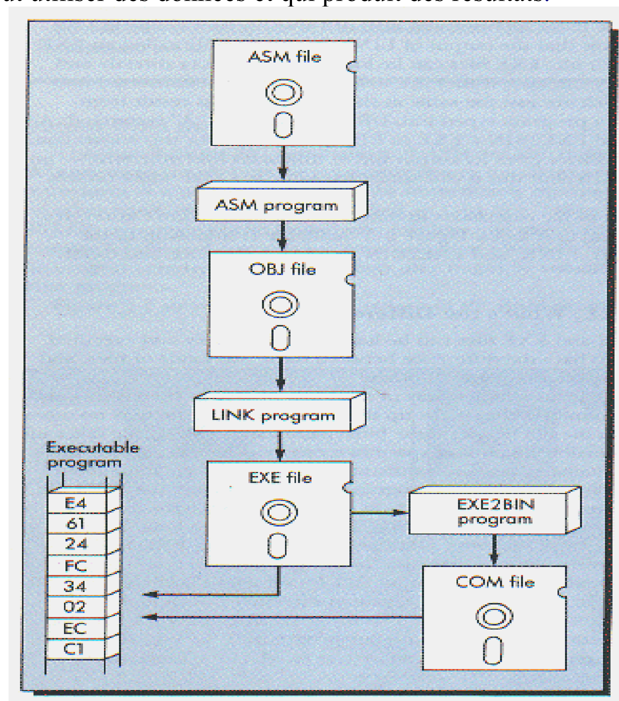
Sur cet exemple, nous pouvons remarquer la différence de syntaxe entre deux processeurs différents: lorsqu'un constructeur conçoit un processeur, il détermine aussi son assembleur. C'est pour cela que l'assembleur n'est pas universel car il est différent sur chaque processeur (ou plutôt concepteur). Et même si par hasard une instruction est identique entre 2 CPU différentes, elle ne sera pas compatible sur chaque processeur car le code binaire correspondant sera différent. Ceci est l'un des plus gros inconvénients de la programmation en assembleur car il faut alors reprogrammer de fond en comble le logiciel si on veut le porter sur un autre processeur. Chaque processeur comporte un nombre plus ou moins important d'instructions.

Dans ce chapitre, nous allons nous restreindre au langage assembleur du microprocesseur intel 8086.

2. L'assemblage

Un programme écrit en langage d'assemblage doit subir un certain nombre de transformations avant de pouvoir être exécuté. La figure suivante présente les différentes étapes du traitement d'un programme en langage d'assemblage.

La première phase de traitement est l'assemblage (TASM). Le programme source est traduit en langage machine et le programme objet ainsi obtenu est rangé dans un fichier. La seconde phase de traitement est l'édition des liens. La troisième phase du traitement est l'exécution du programme proprement dite qui peut utiliser des données et qui produit des résultats.



L'assembleur assigne aux instructions qu'il traduit des adresses dites relatives car elles sont attribuées de façon séquentielle à partir du début du programme.

3. Edition de liens

Le fichier .OBJ contient donc le produit binaire de l'assemblage. Mais ce fichier est inutilisable tel quel. Le DOS ne peut pas le charger et encore moins l'exécuter. Pour ce faire, nous avons encore deux étapes à franchir. La première est celle de l'édition de liens (LINK, c'est-à-dire "lier" en anglais). L'utilitaire LINK ou TLINK continue le travail commencé par l'assembleur en lisant le fichier .OBJ pour créer un fichier "exécutable" (.EXE). On peut "lier" plusieurs fichiers .OBJ en un seul et unique fichier .EXE, et c'est pourquoi on le nomme "éditeur" de liens. On pourrait être plus précis en disant : un éditeur automatique de liens.

4. Chargement du programme

On ne peut exécuter un programme que s'il se trouve en mémoire centrale. L'assembleur range le programme objet en mémoire secondaire et l'éditeur de lien en fait un produit exécutable: mais avant de pouvoir exécuter ce programme, il faut le charger en mémoire centrale. Chaque instruction du programme possède une adresse relative qui lui a été attribuée par l'assembleur; il serait simple de placer le programme en mémoire de façon à ce que les adresses relatives correspondent aux adresses réelles des instructions (appelées adresses absolues). Ceci n'est pas possible car une partie de la mémoire centrale est occupée en permanence par le système d'exploitation; le programme ne pourra alors être placé en mémoire à partir de l'adresse zéro ou de l'adresse 100h. Si la zone de mémoire disponible pour le programme commence à l'adresse physique 2C8F0 (2C8F:0000), la première instruction du programme (d'adresse relative zéro) sera placée à l'adresse 2C8F0 pour un programme .exe et l'adresse 2C9F0 pour un programme.com (d'adresse relative 100h) et les instructions suivantes aux adresses qui suivent: on dit alors que le chargeur a translaté le programme.

5. Les registres

Un registre est un élément de mémoire interne du microprocesseur. Sa capacité est très réduite, son rôle n'est pas de stocker de données, mais plutôt des adresses ou les résultats intermédiaires de calculs. La plupart des commandes de l'assembleur utilisent les registres. Il est donc important de connaître leurs utilités respectives. Un registre n'est qu'une zone de stockage de chiffres binaires, à l'intérieur même du processeur. Le 8086 dispose d'un certain nombre de registres qui sont:

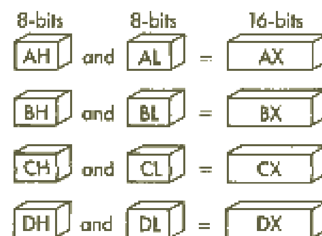
AX	accumulateur
BX, CX, DX	registres banalisés
DI, SI, BP	registres d'index
IP	pointeur d'instructions
SP	pointeur de pile
CS	registre de segment de code
DS	registre de segment de données
SS	registre de segment de pile
ES	extra segment
FLAGS	registre d'état.

Tous ces registres sont des registres 16 bits. Néanmoins les registres **AX, BX, CX, DX** peuvent être subdivisées en deux registres de huit bits supérieurs et inférieurs, et être référencés sous ***x*H** et ***x*L:** par exemple AH et AL, pour AX. En d'autre termes **AX**=256***AH**+**AL**.

5.1. Registres de données

Il existe quatre registres de 16 bits : AX, BX, CX, DX. Ils servent pour le calcul et le stockage. Cependant, chacun d'eux peut être scindé en deux registres de 8 bits. L'octet de gauche d'un registre 16 bits est dit de poids fort, il est le plus significatif tandis que celui de droite (le moins significatif) est dit de poids faible. Ainsi, AX peut être décomposé en deux registres: AH et AL. II en va de même des registres BX, CX et DX. D'une manière générale, les programmes en assembleur placent dans ces registres des nombres issus de la mémoire, effectuent des opérations sur ces registres et replacent ensuite les résultats en mémoire.

Peu de programmes s'écartent de ce schéma de base. Quant aux registres autres que AX, BX, CX et DX, ils sont souvent utilisés pour les déplacements de nombres entre la mémoire et ces 4 registres. Ces registres sur 16 bits peuvent être utilisés sous la forme de 2 registres 8 bits chacun. Les registres 8 bits correspondant à AX, BX, CX et DX se nomment AL, AH, BL, BH, CL, CH, DL, DH.



Par conséquent, on peut inscrire une valeur quelconque sur 8 bits dans le registre BL, par exemple, sans que cela n'affecte le contenu du registre BH. Les registres de données sont en général interchangeables : malgré leurs fonctions, ils sont en effet d'un usage général. Cependant, on utilise souvent AX pour des opérations simples avec des instructions qui adressent implicitement ce registre sans qu'il soit nécessaire de le spécifier.

- **AX** est l'accumulateur;
- **BX,** le registre de " Base ", peut parfois être utilisé comme base d'indexage.
- **CX** est le registre " Compteur ". Vous vous rendrez compte que de nombreuses opérations de comptage sont effectuées directement avec CX par le processeur ;
- **DX** est le registre propre aux manipulations des " Données ". Dans les opérations sur 32 bits, avant l'apparition du processeur INTEL 386, les 16 bits de poids fort sont placés dans CX et ceux de poids faible, dans DX.

Ce sont donc les tâches spécifiques des registres CX, DX, AX et BX. Mais rappelons-le : ils sont interchangeables !

5.2. Registres d'index

Les registres d'index DI et SI et les registres "pointeurs de pile" (BP et SP) sont des registres spécifiques au langage assembleur.

- **SI** est le registre Source (Source Index) et DI le registre de destination (Destination Index) dans les opérations d'indexage.
- **BP** est le registre de base de la pile. C'est lui qui pointe sur l'adresse de base de la pile (nous verrons cela plus tard).
- **SP** (Stack pointer), est le pointeur de pile qui pointe sur le sommet de la pile. On dit de ces

registres d'indexage qu'ils "pointent" sur un emplacement mémoire. Ainsi, si SI contient F000h, on dira qu'il pointe sur l'offset F000h.

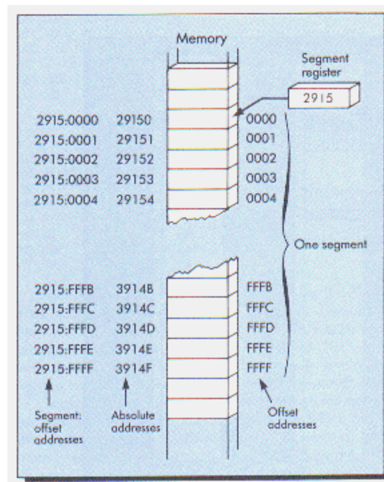
5.3. Registres de segment

Dans l'environnement du système d'exploitation DOS, nous devons donc être capables de représenter des nombres allant de 0 à 1 048 575. Pour générer les adresses nécessaires à une mémoire de 1 méga-octet il faut donc avoir 20 bits. Le processeur x86 utilise des mots de 20 bits pour adresser la mémoire réelle. Le jeu complet des 1 048 576 adresses différentes est appelé "espace adressable de 1 méga-octet". Cependant l'unité centrale du x86 ne comporte aucun registre de 20 bits puisque les registres sont de 16 bits. Un registre unique ne peut donc référencer que 64K de mémoire (2^{16} = 65 536). II est donc nécessaire de combiner deux registres pour accéder à la mémoire. Le x86 en mode réel combine un des neuf registres généraux avec l'un des quatre registres de segments pour former une adresse de 20 bits. Les registres de segments sont appelés

- segment code (CS) ;
- segment données (DS) ;
- segment extra (ES) ;
- segment pile (SS).

À chaque type d'accès en mémoire correspond par défaut un registre de segment et parfois un registre général. Par exemple, CS et IP sont combinés pour accéder à la prochaine instruction à exécuter sous la forme CS :IP. SS et SP sont combinés en SS:SP pour toutes les opérations concernant la pile. II est parfois possible d'utiliser un registre de segment autre que celui utilisé par défaut en le spécifiant explicitement.

Si nous mettons bout à bout deux registres de 16 bits, nous obtenons une adresse sur 32 bits. Nous pourrions ainsi adresser 4 gigaoctets (2^{32} = 4 294 967 296). Pour adresser 1 méga-octet, nous n'avons besoin que de 4 bits en plus des 16 bits d'un registre général. Cette combinaison est obtenue en décalant le registre de segment de 4 bits et en ajoutant le résultat au contenu du registre général pour obtenir l'adresse sur 20 bits. Le résultat est appelé adresse effective (EA). L'adresse effective est placée sur le bus d'adresses afin de pouvoir accéder à l'emplacement mémoire correspondant. On peut donc considérer l'adresse effective comme constituée de deux parties. Le registre de segment définit une zone mémoire de 64K, et le registre général spécifie un déplacement à partir de l'origine de ce segment de 64K (c'est-à-dire une adresse sur 16 bits à l'intérieur de ce segment). L'adresse effective est exprimée en donnant le registre segment et le registre général séparés par deux points. Une adresse mémoire peut donc être symboliquement représentée par: DS:SI ou explicitement par : 2915:0004. Le segment de 64K est défini par le registre DS et le déplacement dans ce segment est contenu dans le registre SI. L'adresse du segment est 2915 et le déplacement dans le segment est 0004 en hexadécimal. L'adresse effective est donc 29154 hexadécimal. Notez que le décalage de l'adresse du segment est obtenu en ajoutant un zéro à droite. Ainsi, 2915:0004 devient 29150 + 0004, soit 29154, qui est l'adresse effective. Le même calcul se fait pour le cas de 2915:FFFB; 29150 + FFFB = 3914B.



5.4. Pointeur d'instructions

Le registre IP (Instruction pointer) est le pointeur d'instructions. Il indique la prochaine instruction à exécuter par le processeur. Dans certains autres processeurs, on l'appelle aussi le "Program Counter" - le compteur d'instruction de programme). Il a la responsabilité de guider le processeur lors de ses déplacements dans le programme en langage machine, instruction par instruction. Le registre IP est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante. Le x86 dépend entièrement du registre IP pour connaître l'instruction suivante.

5.5. Registre Drapeau

Le registre Drapeau est un ensemble de 16 bits. La valeur représentée par ce nombre de 16 bits n'a aucune signification en tant qu'ensemble: ce registre est manipulé bit par bit, certains d'entre eux influenceront le comportement du programme. Les bits de ce registre sont appelés "indicateurs", on peut les regrouper en deux catégories:

5.5.1. Indicateurs d'état

Bit	signification	abréviation	TDEBUG
0	"Carry" ou Retenue	CF	c
2	Parité	PF	p
9	Retenue auxiliaire	AF	a
6	Zéro	ZF	z
7	Signe	SF	s
11	débordement (overflow)	OF	o

5.5.2. Indicateurs de contrôle

Bit	signification	abréviation	TDEBUG
8	trap	TF	
9	interruption	IF	i
10	direction	DF	d

Les bits l, 5, 12, 13,14, IS de ce registre FLAG de 16 bits ne sont pas utilisés. Les indicateurs sont en relation directe avec certaines instructions: Les instructions arithmétiques, logiques et de comparaison modifient la valeur des indicateurs.

DEC AX ; affecte les indicateurs CF, ZF, SF, OF, AF
CMP AX,1 ; affecte les indicateurs CF, ZF, SF, OF, AF

Les instructions conditionnelles testent la valeur des indicateurs et agissent en fonction du résultat.

JA - teste les indicateurs CF et ZF
JNE - teste l'indicateur ZF
JS - teste l'indicateur SF

Lors d'une instruction non arithmétique ou logique, les indicateurs CF et OF sont remis à 0. Nous verrons que dans certains cas les indicateurs ne sont pas modifiés, dans certains cas ils le sont et, parfois, ils sont indéfinis...

```
[■]=CPU 80486                                          1=[↑][↓]
 cs:0100►B401        mov    ah,01        ax 0000    c=0
 cs:0102 CD21        int    21           bx 0000    z=0
 cs:0104 8AD0        mov    dl,al        cx 0000    s=0
 cs:0106 80EA30      sub    dl,30        dx 0000    o=0
 cs:0109 80FA09      cmp    dl,09        si 0000    p=0
 cs:010C 7E03        jle    0111         di 0000    a=0
 cs:010E 80EA07      sub    dl,07        bp 0000    i=1
 cs:0111 B104        mov    cl,04        sp FFFE    d=0
 cs:0113 D2E2        shl    dl,cl        ds 5750
 cs:0115 CD21        int    21           es 5750
 cs:0117 2C30        sub    al,30        ss 5750
 cs:0119 3C09        cmp    al,09        cs 5750
 cs:011B 7E02        jle    011F         ip 0100

 ds:0000 CD 20 FF 9F 00 9A F0 FE  = ƒ Ü-■
 ds:0008 1D F0 E0 01 5B 1E AA 01  ↔-Ó☻[▲¬☺    ss:0000 20CD
 ds:0010 5B 1E 89 02 B6 18 28 0A  [▲ë☻Â↑(◙    ss:FFFE►0000
 ds:0018 01 01 01 00 02 FF FF FF  ☺☺☺ ☻
```

Le registre Drapeau dans TDEBUG se situe en haut à droite des autres registres

5.5.3. Signification des principaux indicateurs:

L'indicateur CF (CARRY ou retenue): Il sera mis à 1 s'il y a eu retenue lors de la dernière instruction arithmétique.

L'indicateur OF (OVERFLOW ou débordement) : Il sera mis à 1 si le résultat d'une addition de 2 nombres positifs donne un nombre négatif et inversement. En règle générale, cet indicateur est activé quand le signe change alors qu'il ne devrait pas. En arithmétique non signée, la valeur de cet indicateur n'aura pas de signification.

L'indicateur ZF (ZERO): Il sera mis à 1 (activé, set) si le résultat d'une instruction arithmétique a donné zéro et il le restera jusqu'à la prochaine instruction arithmétique.

L'indicateur SF (SIGN ou signe) Il sera mis à 1 si le résultat d'une instruction a donné un nombre négatif (bit de poids fort =1), il sera mis à 0 dans le cas contraire.

L'indicateur DF (DIRECTION) Il est modifié par une instruction CLD ou STD et pas par le résultat d'une instruction (cf. manipulation de chaînes de caractères).

MCours.com

L'indicateur PF (PARITÉ) L'indicateur est mis à 1 si le résultat d'une opération contient un nombre pair de bits 1.

5.6. Comparaisons préalables au branchement

Pour effectuer un branchement, il faut dans bien des situations faire une comparaison. Il existe une seule instruction de cette catégorie permettant de comparer deux registres ou emplacements de mémoire: **CMP**. Les opérations de comparaison sont toujours suivies d'une instruction de branchement conditionnel car elles affectent les indicateurs du registre Drapeau. Le résultat de la comparaison est indiqué par les indicateurs. Les cinq formes disponibles de cette instruction sont:

CMP reg, imm	CMP AX, 0Ah
CMP reg, mem	CMP AX, [BX]
CMP mem, imm	CMP [BX], 0Ah
CMP mem, reg	CMP [BX], AX
CMP reg, reg	CMP AX, BX

"reg" étant un registre de 8 ou 16 bits (sauf un registre de segment);
"mem" étant une adresse (ou un identificateur);
"imm" est une valeur immédiate (constante).

Note: une opération de comparaison est en fait une soustraction qui n'affecte aucune opérande (cf. instructions arithmétiques):

CMP A,B <==> (A-B)

Dans le cas de comparaison de caractères, le 8086 effectue la soustraction des codes ASCII des caractères.

5.7. Conditions de branchements

Il existe deux catégories d'instructions pour tester les conditions de branchements. La première peut s'exercer sans le recours de comparaisons préalables. On peut employer JNZ après une comparaison mathématique du type DEC ou INC.

5.7.1. Premier groupe

JO	7O	JUMP IF O=1	(si débordement) ;	JNO	71	JUMP IF O=0	(si pas de débordement)
JC	72	JUMP IF C=1	(si "carry");	JNC	73	JUMP IF C=0	(si pas de carry)
JZ	74	JUMP IF Z=1	(si zéro);	JNZ	75	JUMP IF Z=0	(si pas de zéro)
JS	78	JUMP IF S=1	(si signe) ;	JNS	79	JUMP IF S=0	(si non signé)
JP	7A	JUMP IF P=1	(si parité) ;	JNP	7B	JUMP IF P=0	(si pas de parité)

5.7.2. Deuxième groupe

JBE	76	JUMP IF(C=1) OR (Z=1);	JA	77	JUMP IF(C=0) AND (Z=0)
JL	7C	JUMP IF S <> 0;	JGE	7D	JUMP IF S = 0
JLE	7E	JUMP IF((S XOR 0) OR Z)= 1;	JG	7F	JUMP IF((S XOR 0) OR Z)= 0

Le deuxième groupe comprend des tests constitués de combinaisons d'indicateurs que l'on utilise généralement en relation avec CMP (comparer) pour construire des boucles. Cette instruction

de comparaison est capable de calculer des relations plus complexes entre deux valeurs : pour savoir si un nombre est inférieur ou égal à un autre, par exemple.

5.8. Codage de quelques instructions :

Regardons maintenant le code assembleur nécessaire pour exécuter quelques instructions fréquentes.

IF THEN ELSE ... **En assembleur, on obtient :**

```
Si (ax==1)                  If:      CMP  AX, 1
    bx = 10;                         JNZ  Else
Sinon {                     Then:    MOV  BX,10
        bx = 0;                      JMP  EndIf
        cx  =10;            Else:    MOV  BX,0
      }                              MOV  CX,10
EndIf:
```

La boucle FOR ...

```
bx = 0 ;                             MOV BX, 0
Pour K = 0 jusqu'à 10                MOV CX,0
    bx  = bx + K ;            For:   CMP CX,10
                                     JA EndFor
                                     ADD BX, CX
                                     INC CX
                                     JMP For
                              EndFor:
```

La boucle while .

```
  Bx = 5 ;                             MOV BX,5
  Tant que bx > 0              While :  CMP BX,0
     faire bx = bx-1                    JLE EndWhile
                                        DEC BX
                                        JMP While
                              EndWhile:
```

La boucle repeat

```
 bx =  10 ;                          MOV BX,10
 Répéter                 Repeat1:    DEC BX
    bx = bx – 1 ;                    CMP BX,0
 jusqu'à bx <= 0                     JG Repeat1
                          Endrepeat1 :
```

Si le nombre de répétitions est connu et au moins égal à 1, on pourra procéder comme suit:

```
          MOV BX, 0
          MOV CX, 0Ah
Repeat2:
          ADD BX, CX
```

```
            LOOP Repeat2; LOOP décrémente automatiquement CX jusqu'à atteindre 0
EndRepeat2:
```

Le test à plusieurs alternatives...

```
Switch(BX){                              CMP BX,1
  case 1: ax =1 ; break;                 JNZ case2
  case  2: ax = 5 ; break ;              MOV AX,1
  case  3: ax  = 10; break;              JMP endswitch
  default: ax = 0;                case2: CMP BX,2
  }                                      JNZ case3
                                         MOV AX,5
                                         JMP endswitch
                                  case 3: CMP BX, 3
                                         JNZ default
                                         MOV AX,10
                                         JMP endswitch
                                  default:
                                          MOV AX,0
                                  endswitch:
```

5.9. Le compteur d'instructions (IP)

Puisque le pointeur d'instruction est le registre servant à exécuter les instructions, il est nécessairement modifié par un saut conditionnel. Si la condition du test est remplie, on place une nouvelle valeur dans le compteur du programme. Si, par contre, les conditions du test ne sont pas remplis (comme dans le cas d'une instruction JNZ (Jump if Not Zero) avec l'indicateur Z activé), alors le pointeur d'instruction suit son cours normalement, et le programme se déroule comme si aucun saut conditionnel n'avait été rencontré. Si les conditions du test sont remplies, l'octet de données (c'est-à-dire l'octet qui suit l'instruction de test) est additionné l'IP (Instruction Pointer). Par exemple, si l'IP contient 102h, si le processeur se rend compte que l'indicateur Z est désactivé (en lisant l'octet complémentaire, disons 10h, de JNZ), la nouvelle valeur de l'IP après une instruction JNZ 10 sera le résultat l'addition : 10h plus 102h. L'instruction qui sera exécutée immédiatement après le JNZ est celle qui se trouve à l'adresse 122h (102h+10h).

5.10. Structure d'un programme assembleur

Comme dans tout programme le fichier source doit être saisi de manière rigoureuse. Chaque définition et chaque instruction doivent ainsi s'écrire sur une nouvelle ligne (pour que l'assembleur puisse différencier les différentes instructions) Le fichier source contient:

1. Un nom du programme sous TITLE sui vie d'un nom. Cette partie est facultative.
2. Une partie pour déclarer une pile qui est définie dans le segment de pile délimité par les directives SEGMENT STACK et ENDS
3. Des définitions de données déclarées par des directives. Celles-ci sont regroupées dans le segment de données délimité par les directives SEGMENT et ENDS
4. Puis sont placées les instructions (qui sont en quelque sorte le coeur du programme), la première devant être précédée d'une étiquette, c'est-à-dire par un nom qu'on lui donne. Celles-

ci sont regroupées dans le segment d'instructions délimité par les directives SEGMENT et ENDS

5. Enfin, le fichier doit être terminé par la directive END suivi du nom de l'étiquette de la première instruction (pour permettre au compilateur de connaître la première instruction à exécuter (Les points-virgules marquent le début des commentaires, c'est-à-dire que tous les caractères situés à droite d'un point virgule seront ignorés) Voici à quoi ressemble un fichier source (fichier .ASM):

```
TITLE  nomprogramme     ;cette directive permet de nommer votre programme

Pile    SEGMENT  STACK; segment de pile dont le nom est Pile (ou tout autre nom)

        ; déclarer la pile et sa taille

 Pile   ENDS

Donnees  SEGMENT; voici le segment de données dont l'étiquette est Donnees
                ; (ou tout autre nom)

        ;Placez ici les déclarations de données

Donnees       ENDS; ici se termine le segment de donnees

Lecode   SEGMENT  ; voici le segment d'instructions dont l'étiquette est Lecode
                  ; (ou tout autre nom)

           ASSUME DS:donnee, CS: Lecode


         debut:        ; placez ici votre première instruction (son étiquette est nommée debut)

                       ; placez ici le reste de vos instructions

Lecode   ENDS; fin du segment d'instructions

          END debut; fin du programme suivie de l'étiquette de la première instruction
```

5.10.1. Déclaration d'un segment

Pour l'instant, oublions le segment de pile. Les données sont regroupées dans une zone de la mémoire appelée segment de données, tandis que les instructions se situent dans un autre segment qui est le segment d'instructions. Le registre DS (Data Segment) contient le segment de données, tandis que le registre CS (Code Segment) contient le segment d'instructions. C'est la directive ASSUME qui permet d'indiquer à l'assembleur où se situe le segment de données et le segment de code. Puis il s'agit d'initialiser le segment de données:

```
MOV AX, nom_du_segment_de_donnees
MOV DS, AX
```

On fera pareil quand viendra le temps où le segment de pile va être utilisé.

MCours.com