Langage Assembleur : les bases

.1. Introduction

Dire que « le langage assembleur est simple » a sa part de vérité. Son exécution requiert peu de mémoire et ses instructions sont, pour la plupart, de bas niveau. Alors pourquoi il a la réputation d'être difficile ?

Voici un programme simple en assembleur (masm) d'un programme qui ajoute deux nombres et affiche le résultat :

```
main PROC
mov eax,5          ; move 5 to the EAX register
add eax,6          ; add 6 to the EAX register
call WriteInt      ; display value in EAX
exit               ; quit
main ENDP
```

« WriteInt » simplifie le code considérablement, mais en générale le langage assembleur n'est pas difficile. Le programmeur doit donner plus d'importance aux petits détails ce qui rend le code beaucoup plus volumineux.

.2. Syntaxe du langage : Eléments de base

Dans ce qui suit, nous ferons le tour des éléments de base de la syntaxe du MASM.

.2.1. Constantes et expressions entières :

- La notation Microsoft est utilisée tout au long de ce chapitre. Les éléments entre crochets [..] sont facultatifs et les éléments entre accolades {..} nécessitent un choix de l'un des éléments inclus (séparés par le caractère |). Les éléments en italique désignent des éléments qui ont des définitions ou des descriptions connues.

Une constante entière est composée d'un signe, d'un ou plusieurs digits et d'une base :

[{+ | −}] digits [base]

Les bases possibles sont :

h Hexadecimal	r Encoded real
q/o Octal	t Decimal (alternate)
d Decimal	y Binary (alternate)
b Binary	

Exemples :

26 Decimal	42o Octal
26d Decimal	1Ah Hexadecimal
11010011b Binary	0A3h Hexadecimal
42q Octal	

- Une constante hexadécimale commençant par une lettre doit être précédée d'un zéro pour empêcher l'assembleur de l'interpréter comme un identifiant.

Une expression entière est une expression mathématique impliquant des valeurs entières et des opérateurs arithmétiques. La valeur de l'expression doit correspondre à un entier qui peut être stocké en 32 bits (0 à FFFFFFFFh).

Exemples :

4 + 5 * 2

12 - 1 MOD 5

-5 + 2

(4 + 2) * 6

.2.2. Constantes et expressions réelles :

Syntaxe :

[sign]integer.[integer][exponent]

Avec :

sign {+,-}

exponent E[{+,-}]integer

Exemples :

2.

+3.0

-44.2E+05

26.E5

.2.3. Constantes caractères et chaines de caractères

Un caractère ou une chaine de caractères est entourée de guillemets simples ou doubles.

Exemples :

'A'

"d"

'ABC'

'X'

.2.4. Mots réservés

Les mots réservés ont une signification connue par MASM. Il ne peuvent etre utilisé que dans des contextes bien définis.

Il y a plusieurs types de mots réservés :

- Mnémonique d'instruction
- Nom de registre
- Directive
- Opérateur
- Symboles prédéfinis

Le programmeur peut être amené à choisir un identifiant pour une variable, une procédure ou autre. Il y a un certain nombre de règles à respecter lors du choix d'un identifiant :

- Il peut contenir de 1 à 247 caractères
- Le premier caractère doit être une lettre, le caractère de soulignement _,@, ? ou &
- Il doit être diffèrent des mots réservés

.2.5. Directives

Une directive est une commande qui permet de définir des variables, des macros et des procédures. Elle peut attribuer un nom à un segment de mémoire …

L'exemple suivant explique la différence entre une directive et une instruction

```
myVar DWORD 26          ; DWORD directive
mov eax,myVar           ; MOV instruction
```

➢ Définir les segments :

Parmi les fonctions principales des directive est celle de définition des segments de code :

```
.data
; dans cette partie du code on définit les données
.code
; dans cette partie on écrit les instructions
```

.2.6. Instructions

Une instruction est une commande sui devient exécutable quand le programme est assemblé. Elle est traduite par l'Assembleur en langage machine, qui lui est exécuté par le CPU dans la phase de l'exécution.

➢ La syntaxe d'une instruction est la suivante :

```
[label:]  mnémonique  [opérandes] [;comment]
```

- Label

C’est un identifiant qui permet de référencer une instruction. Un label placé juste avant une instruction équivaut à son adresse (de la même manière, le label d’une variable équivaut à son adresse)

- Exemple :

```
compteur DWORD 100
```

Dans cet exemple, l’Assembleur affecte une valeur numérique au label compteur.

Il est possible de définir plusieurs données avec un seul label

- Exemple :

```
Tableau  DWORD 100,5467
         DWORD 200,400
```

Un « code label » est un label de code. Il est utilisé pour référencer l’instruction et servent dans des instructions telles que JMP et LOOP. Un label de code doit être suivi de « : »

- Exemple :

```
target:
mov ax,bx
...
jmp target
```

- Mnémonique d’instruction

C’est un mot court qui identifie une instruction.

- Exemple :

Mov Move (assign) one value to another

Add Add two values

Sub Subtract one value from another

mul Multiply two values

jmp Jump to a new location

call Call a procedure

- Opérandes

Une instruction peut avoir de zéro à trois opérandes.

Un opérande peut être un nom de registre, un opérande mémoire ou une constante.

Example	Operand Type
96	Constant (*immediate value*)
2 + 4	Constant expression
eax	Register
count	Memory

- Exemples :

```
stc                 ; set Carry flag
inc eax             ; add 1 to EAX
mov count,ebx       ; move EBX to count
imul eax,ebx,5      ; ebx is multiplied by 5, and the product is stored in the EAX register
```

- Commentaires

Les commentaires sur une seule ligne sont précédés par « ; »

Un ensemble de commentaires est précédé par COMMENT

- Exemples :

```
                ; set Carry flag
COMMENT &
```

Knqsdf

Sqdf

Sqdf

Sgert

&

- Structure d'un programme

TITLE titre du programme

INCLUDE ….

.data

; (insert variables here)

.code

main PROC

; (insert executable instructions here)

exit

main ENDP

; (insert additional procedures here)

END main

- Premier programme

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
.386
.model flat,stdcall

include \masm32\include\masm32rt.inc

.code
main PROC

mov eax,10000h ; EAX = 10000h
add eax,40000h ; EAX = 50000h
sub eax,20000h ; EAX = 30000h

exit
main ENDP
END main
```

.3. Définition de données

.3.1. Syntaxe

La définition d'une donnée permet d'allouer de l'espace mémoire pour cette donnée. Le nom de la donnée est optionnel.

La syntaxe pour définir (déclarer) une donnée (variable) est :

[name] directive initializer [,initializer]...

La directive dans cette syntaxe peut être une parmi les types dans la table suivante :

Type	Usage
BYTE	8-bit unsigned integer. B stands for byte
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer (can also be a Near pointer in real-address mode)
SWORD	16-bit signed integer
DWORD	32-bit unsigned integer (can also be a Near pointer in protected mode). D stands for double
SDWORD	32-bit signed integer. SD stands for signed double
FWORD	48-bit integer (Far pointer in protected mode)
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit (10-byte) integer. T stands for Ten-byte
REAL4	32-bit (4-byte) IEEE short real
REAL8	64-bit (8-byte) IEEE long real
REAL10	80-bit (10-byte) IEEE extended real

Ou dans la table suivante :

Directive	Usage
DB	8-bit integer
DW	16-bit integer
DD	32-bit integer or real
DQ	64-bit integer or real
DT	define 80-bit (10-byte) integer

- Exemple :

```
compteur DWORD 12345
value1 BYTE 'A' ; character constant
value2 BYTE 0 ; smallest unsigned byte
value3 BYTE 255 ; largest unsigned byte
value4 SBYTE −128 ; smallest signed byte
value5 SBYTE +127 ; largest signed byte
val1 DB 255 ; unsigned byte
val2 DB -128 ; signed byte
```

```
list BYTE 10,20,30,40
list BYTE 10,20,30,40
 BYTE 50,60,70,80
 BYTE 81,82,83,84
list1 BYTE 10, 32, 41h, 00100010b
list2 BYTE 0Ah, 20h, 'A', 22h ; list1 and list2 have the same contents
greeting1 BYTE "Good afternoon",0 ; greeting1 BYTE 'G','o','o','d'....etc.
greeting2 BYTE 'Good night',0
word1 WORD 65535 ; largest unsigned value
word2 SWORD -32768 ; smallest signed value
word3 WORD ? ; uninitialized, unsigned
quad1 QWORD 1234567812345678h
```

- L'opérateur DUP :

Il permet d'allouer de l'espace pour plusieurs données à la fois.

- Exemple :

```
BYTE 20 DUP(0) ; 20 bytes, all equal to zero
BYTE 20 DUP(?) ; 20 bytes, uninitialized
BYTE 4 DUP("STACK") ; 20 bytes: "STACKSTACKSTACKSTACK"
array WORD 5 DUP(?) ; 5 values, uninitialized
```

- Structure little-endian

Les CPU x86 enregistrent et lisent les données de la mémoire en utilisant la structure little-endian. (low to hight).

Par exemple la représentation de 12345678h en mémoire est la suivante :

0000:	78
0001:	56
0002:	34
0003:	12

- Si on modifie le programme vu dans la section précédante en ajoutant les variables :

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
.386
.model flat,stdcall

include \masm32\include\masm32rt.inc

.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
mov eax,val1 ; start with 10000h
add eax,val2 ; add 40000h
sub eax,val3 ; subtract 20000h
mov finalVal,eax ; store the result (30000h)

exit
main ENDP
END main
```

.3.2. Déclarer des données non initialisées

La directive .data ? est utilisée au lieu de la directive .data pour déclarer des données non initialisées.

- Exemple :

```
.data

smallArray DWORD 10 DUP(0) ; 40 bytes

.data?

bigArray DWORD 5000 DUP(?) ; 20,000 bytes, not initialized
```

.3.3. Les constantes symboliques

Une constante symbolique (ou définition de symbole) est créée en associant un identifiant (un symbole) à une expression entière ou du texte. Les symboles ne sont utilisés que par l'Assembleur lors de l'analyse d'un programme, et ils ne peuvent pas être changés à l'exécution.

- La directive « = »

Cette directive associe un nom à une expression entière. Sa syntaxe est la suivante :

Name = expression (x = 30 par exemple)

Quand le programme est assemblé toutes les occurrence de name sont remplacées par sa valeur.

- Exemple :

```
COUNT = 5
mov al,COUNT ; AL = 5
COUNT = 10
mov al,COUNT ; AL = 10
COUNT = 100
mov al,COUNT ; AL = 100
```

- Le compteur d'emplacement courant :

Comme son nom l'indique, il permet de représenter l'emplacement courant dans le segment courant (l'offset)

- Exemple :

```
selfPtr DWORD $        ; declares a variable named selfPtr and initializes it with its own location counter
```

.3.4. Calculer la taille des tableau et des chaines de caractère

Dans l'exemple qui suit, nous utilisons le compteur d'emplacement courant pour connaitre la taille d'une liste (tableau) d'éléments.

```
list BYTE 10,20,30,40

ListSize = ($ - list)
```

Les deux instructions doivent se suivre !! (la raison est évidente)

Cet exemple donne donne une taille erronée :

```
list BYTE 10,20,30,40

var2 BYTE 20 DUP(?)

ListSize = ($ - list)
```

Nous calculons la taille des chaines de caractère de la meme manière :

```
myString BYTE "This is a long string, containing"

 BYTE "any number of characters"

myString_len = ($ − myString)
```

Lorsqu'on calcule la taille d'un tableau d'un type autre que BYTE, il faut faire attention à diviser par la taille du dit type :

Exemple :

```
list WORD 1000h,2000h,3000h,4000h

ListSize = ($ − list) / 2

list DWORD 10000000h,20000000h,30000000h,40000000h

ListSize = ($ − list) / 4
```

.4. Affectation de données, adressage et arithmétique

.4.1. Instructions d'affectation

.4.1.1. Types d'opérandes

Pour donner plus de flexibilité au code, le langage assembleur utilise plusieurs types d'opérandes d'instructions. Les plus utilisés sont :

- Valeur immédiate
  Mov eax,32
- Nom d’un registre
  Mov eax,ebx
- Opérande mémoire :
  var1 BYTE 10h
  mov AL,var1

.4.1.2. L’instruction mov

MOV registre1, registre2 a pour effet de copier le contenu du registre2 dans le registre1, le contenu préalable du registre1 étant écrasé. Cette instruction vient de l'anglais « move » qui signifie « déplacer » mais attention, le sens de ce terme est modifié, car l'instruction MOV ne déplace pas mais place tout simplement. Cette instruction nécessite deux opérandes qui sont la destination et la source. Ceux-ci peuvent être des registres généraux ou des emplacements mémoire. Cependant, les deux opérandes ne peuvent pas être toutes les deux des emplacements mémoire. De même, la destination ne peut pas être ce qu'on appelle une valeur immédiate (les nombres sont des valeurs immédiates, des valeurs dont on connait immédiatement le résultat) donc pas de MOV 10, AX. Ceci n'a pas de sens, comment pouvezvous mettre dans le nombre 10, la valeur d’AX ? 10 n'est pas un registre.

MOV reg,reg

MOV mem,reg

MOV reg,mem

MOV mem,imm

MOV reg,imm

- Exemples :

Mov ax, bx ;

Mov ah, cl ;

Mov esi,edi ;

- Règles d'utilisation de mov :

- Il est interdit de transférer le contenu d'une case mémoire vers une autre case mémoire.
- Cs, EIP et IP ne sont jamais utilisés comme registre destination.
- On ne peut pas transférer un registre segment vers un autre registre segment.
- Les opérandes doivent avoir la même taille
- On ne peut pas utiliser une valeur immédiate avec un registre segment

- Exemple

```
.data
oneByte BYTE 78h
oneWord WORD 1234h
oneDword DWORD 12345678h
.code
mov eax,0              ; EAX = 00000000h
mov al,oneByte         ; EAX = 00000078h
mov ax,oneWord         ; EAX = 00001234h
mov eax,oneDword       ; EAX = 12345678h
mov ax,0               ; EAX = 12340000h
```

- Copier le plus petit dans le plus grand

Quoique mov ne le permet pas, on peut contourner le problème.

Supposons que count(16 bits) doit être copiée dans ECX (32 bits), on peut faire ceci :

```
.data
count WORD 1
.code
```

```
mov ecx,0
mov cx,count
```

- L'instruction MOVX

Cette instruction copie le contenu de la source dans la destination tout en complétant avec des zero :

MOVZX reg32,reg/mem8

MOVZX reg32,reg/mem16

MOVZX reg16,reg/mem8

- exemple :

```
.data
byteVal BYTE 10001111b
.code
Movzx ax,byteVal      ; AX= 0000000010001111b
```

- L'instruction XCHG

Permet d'échanger les valeurs de deux opérandes

XCHG reg,reg

XCHG reg,mem

XCHG mem,reg

- Exemple

```
xchg ax,bx          ; exchange 16-bit regs
xchg ah,al          ; exchange 8-bit regs
xchg var1,bx        ; exchange 16-bit mem op with BX
```

```
xchg eax,ebx          ; exchange 32-bit regs
;pour échanger deux opérandes mémoire :
mov ax,val1
xchg ax,val2
mov val1,ax
```

.4.1.3. Les opérandes offset

On peut ajouter un déplacement au nom d'une variable, ceci permet de créer un opérande d'offset direct. Cette opération permet d'accéder à des localisations mémoire qui n'ont pas forcement de labels.

Soit la définition de données suivante :

```
arrayB BYTE 10h,20h,30h,40h,50h
mov al,arrayB                ; AL = 10h
```

On peut accéder au deuxième octet en ajoutnat 2 :

```
mov al,[arrayB+2]          ; AL = 30h
```

Dans une expression telle que arrayB+2, nous ajoutons une constante à l'offset d'une variable. Le résultat est une adresse.

Cas d'un word :

```
.data
arrayW WORD 100h,200h,300h
.code
mov ax,arrayW ; AX = 100h
mov ax,[arrayW+2] ; AX = 200h
```

Cas d'un DWORD :

```
.data
arrayD DWORD 10000h,20000h
```

```
.code
mov eax,arrayD ; EAX = 10000h
mov eax,[arrayD+4] ; EAX = 20000h
```

Le programme suivant résume cette section :

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
.386
.model flat,stdcall
include \masm32\include\masm32rt.inc
.data
val1 WORD 1000h
val2 WORD 2000h
arrayB BYTE 10h,20h,30h,40h,50h
arrayW WORD 100h,200h,300h
arrayD DWORD 10000h,20000h
.code
main PROC
; Demonstrating MOVZX instruction:
mov bx,0A69Bh
movzx eax,bx ; EAX = 0000A69Bh
movzx edx,bl ; EDX = 0000009Bh
movzx cx,bl ; CX = 009Bh
; Demonstrating MOVSX instruction:
mov bx,0A69Bh
```

```
movsx eax,bx ; EAX = FFFFA69Bh
movsx edx,bl ; EDX = FFFFFF9Bh
mov bl,7Bh
movsx cx,bl ; CX = 007Bh
; Memory-to-memory exchange:
mov ax,val1 ; AX = 1000h
xchg ax,val2 ; AX=2000h, val2=1000h
mov val1,ax ; val1 = 2000h
; Direct-Offset Addressing (byte array):
mov al,arrayB ; AL = 10h
mov al,[arrayB+1] ; AL = 20h
mov al,[arrayB+2] ; AL = 30h
; Direct-Offset Addressing (word array):
mov ax,arrayW ; AX = 100h
mov ax,[arrayW+2] ; AX = 200h
; Direct-Offset Addressing (doubleword array):
mov eax,arrayD ; EAX = 10000h
mov eax,[arrayD+4] ; EAX = 20000h
mov eax,[arrayD+4] ; EAX = 20000h
exit
main ENDP
END main
```

.4.1.4. Addition et soustraction

- instructions d'incrémentation et de décrémentation :

INC reg/mem

DEC reg/mem

- Exemples :

```
.data
myWord WORD 1000h
.code
inc myWord ; myWord = 1001h
mov bx,myWord
dec bx ; BX = 1000h
```

- Instruction ADD

```
data
var1 DWORD 10000h
var2 DWORD 20000h
.code
mov eax,var1 ; EAX = 10000h
add eax,var2 ; EAX = 30000h
```

- Instruction SUB

```
.data
var1 DWORD 30000h
var2 DWORD 10000h
.code
mov eax,var1 ; EAX = 30000h
sub eax,var2 ; EAX = 20000h
```

- Instruction NEG

Cette instruction inverse le signe de son opérande.

Application :

L’instruction en C : Rval = -Xval + (Yval - Zval);

Peut être traduite assembleur :

```
Rval SDWORD ?
Xval SDWORD 26
Yval SDWORD 30
Zval SDWORD 40
; first term: -Xval
mov eax,Xval
neg eax ; EAX = -26
; add the terms and store:
add eax,ebx
mov Rval,eax ; -36
```