EL213: Computer Org. & Assembly Language Lab

Lab#03 – Basics of Assembly Language

Agenda

Introduction to Assembly Programming

Basic Assembly Commands

You should have basic information about general purpose registers according to IA (Intel Architecture).

1. MOV
2. ADD
3. SUB
4. MUL

MOV (Move)

Copies data from a source operand to a destination operand, it is equivalent to assignment operator as in C/C++. General Format of instruction is as follows:

Syntax

```
MOV destination, source
```

```
MOV reg1,reg2    ;reg1=reg2
MOV reg, mem     ;reg=mem
```

For example

```
MOV eax,512      ;EAX=512
MOV ebx,eax      ;EBX=EAX
                 ;EBX=512
```

ADD (Addition)

A source operand is added to a destination operand, and the sum is stored in the destination. Operand must be the same size. General Instruction format is as follows:

Syntax

```
ADD destination, source
```

```
ADD reg1, reg2   ;reg1 =reg1+reg2

ADD reg, mem     ;reg =reg + mem

ADD mem , reg    ;mem =mem + reg
```

For example

```
MOV eax,512      ;EAX=512
MOV ebx,123      ;EBX=EAX
ADD eax,ebx      ;EAX=EAX+EBX
                 ;EAX=635
```

SUB (Subtraction)

Subtract the source operand from destination operand, and the subtraction is stored the destination. Operand must be the same size. General Instruction format is as follows:

Syntax

```
SUB destination, source
```

```
SUB reg1, reg2   ;reg1 =reg1-reg2
SUB reg, mem     ;reg =reg - mem
SUB mem, reg     ;mem =mem - reg
```

For example

```
MOV eax,512      ;EAX=512
MOV ebx,123      ;EBX=EAX
SUB eax,ebx      ;EAX=EAX-EBX
                 ;EAX=389
```

MUL (Multiplication):

Multiplies AL, AH, AX or EAX by a source operand. i.e. Multiplies the source operand with Accumulator Register(AC). General Instruction format is as follows:

Syntax

```
MUL source
```

```
MUL reg     ;EAX =EAX*reg
MUL mem     ;EAX =EAX*mem
```

For example

```
MOV eax,12 ;EAX=12

MOV ebx,5  ;EBX=5

MUL ebx    ;EAX=EAX*EBX

           ;EAX=60
```

Data Types in Assembly Language

Following data types are used in assembly for variable declaration.

1. **BYTE:** 8-bit unsigned integer;
2. **SBYTE**: 8-bit signed integer
3. **WORD**: 16-bit unsigned
4. **SWORD**: 16 – Bit Signed integer
5. **DWORD**: 32-bit unsigned
6. **SDWORD**: 32 – bit signed integer
7. **QWORD**: 64-bit integer
8. **TBYTE**: 80-bit integer
9. **REAL4**: 4-byte IEEE short real
10. **REAL8**: 8-byte IEEE long real
11. **REAL10**: 10-byte IEEE extended real

Generic syntax to declare a variable is given below.

```
<Variable Name> <Data Type> <Default Value>/?
```

For example

```
myWord WORD ?
myDWord DWORD 5000
```

Note: Instruction XOR is used to clear the registers contents e.g. XOR EAX, EAX

<u>Getting Started</u>

Use

- `Call WriteInt` ; statement to display the signed integer
- `Call WriteDec` ; statement to display the unsigned integer
- `Call WriteChar` ; statement to display a character
- `Call WriteString` ; statement to display a appropriate message.
- `Call Crlf` ; for new line (CR = Carriage Return (\r), LF = Line Feed (\n))
- `Call Clrscr` ; statement to clear the screen

Defining BYTE and SBYTE Data

```
TITLE Data Definitions
; Examples showing how to define data.
INCLUDE Irvine32.inc

; ----------------- Byte Values ----------------

.data          ;identifies the are of program that contains variables
value1 BYTE 'A'
value2 BYTE 0
value3 BYTE 255
value4 SBYTE 120
value5 SBYTE +127
value6 BYTE ?

.code          ;identifies the area of program that contains instructions
main PROC            ;identifies beginning of a procedure
mov al, value1
call WriteChar ;prints 'A'
call crlf ;for new line

xor eax,eax ;to cear the register to avoid un-expected results
mov al, value2
call Writedec
call crlf

mov al,value3
call WriteDec
call crlf

XOR EAX,EAX
mov al,value4
call WriteDec
call writeInt
call crlf

XOR EAX,EAX
mov al,value5
call WriteDec
call writeInt
call crlf

;(insert instructions here to do some more as you want)
exit
main ENDP
END main
```

Defining String Data Types

```
TITLE Data Definitions
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data

; ----------------- Strings ---------------------

      Str1 BYTE "Welcome to this lab", 0
      Str2 BYTE "Welcome to this program", 0dh, 0ah

.code
main PROC
      mov edx, OFFSET Str1
      call WriteString
      call crlf

      xor edx,edx
      mov edx, offset str2
      call WriteString
exit
main ENDP
END main
```

Both **0ah** and **0dh** are hexadecimal values. Hex values can be specified in two ways in assembly - append an h after the hex value or append the value to 0x.
0ah is equivalent to 10 in decimal and to linefeed ('\n') in ASCII which moves the cursor to the next row of the screen but maintaining the same column. 0dh is equivalent to 13 in decimal and to carriage return ('\r') in ASCII which moves the cursor to the beginning of the current row. A combination of the two thus moves the cursor to the beginning of the next row of the screen.

Note:
The OFFSET operator returns the offset address of a variable.
OFFSET: The distance in bytes from the segment address to another location within segment.

Using DUP Operator

The DUP operator generates a repeated storage allocation. It is particularly useful when allocating space for a string or array, and can be used with both initialized and uninitialized data definitions

```
TITLE Data Definitions
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data
; ----------------- Usage of DUP Operator ---------------------

      Bytedup1 BYTE 20 DUP(0); 20 bytes, all equal to zero
      Bytedup2 BYTE 20 DUP(?); 20 bytes, uninitialized
      Bytedup3 BYTE 3 DUP("FAST-NU"); 21 bytes, "FAST-NUFAST-NUFAST-NU"

.code
main PROC

      ;(insert instructions here to print & manipulate these values)

exit
main ENDP
END main
```

Defining Word Data along with Array

```
TITLE Data Definitions
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data
; ----------------- Word Values ---------------------

      word1 WORD 65535 ; largest unsigned value
      word3 WORD ? ; uninitialized
      myList WORD 1,2,3,4,5 ; array of words

.code
main PROC

      ;(insert instructions here to print & manipulate these values)

exit
main ENDP
END main
```

Defining Double Word Data along with DUP Operator

```
TITLE Data Definitions
```

```
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data
; --------------- DoubleWord Values --------------

      val1 DWORD 12345678h
      val3 DWORD 20 DUP(?)

.code
main PROC

      ;(insert instructions here to print & manipulate these values)

exit
main ENDP
END main
```

Defining Quad-Word and Ten-Byte

```
TITLE Data Definitions
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data
; ------- QuadWord and TenByte Values ------------

      quad1 DQ 234567812345678h
      ten1 DT 1000000000123456789Ah

.code
main PROC

      ;(insert instructions here to print & manipulate these values)

exit
main ENDP
END main
```

Defining Pointer Data Types

```
TITLE Data Definitions
; Examples showing how to define data.

INCLUDE Irvine32.inc

.data
; ----------------- Pointers ---------------------

      arrayB BYTE 10,20,30,40
      arrayW WORD 1000h,2000h,3000h,4000h

      ptrB DWORD arrayB ; points to arrayB
      ptrW DWORD arrayW ; points to arrayW

.code
main PROC

      ;(insert instructions here to print & manipulate these values)

exit
main ENDP
END main
```

Symbolic Constants

Equal-Sign Directive

The *equal-sign* directive associates a symbol name with an integer expression. The Syntax is:

name = expression

```
TITLE Symbolic Constants: Equal-Sign Directive
; Examples showing how to use Symbolic Constants.

INCLUDE Irvine32.inc

.data
      COUNT = 10
      myarray BYTE COUNT DUP(0)

.code
main PROC

      COUNT = 100;
      mov eax, count;
      call WriteInt
      call crlf         ; for new line

      ; (insert more instructions here to manipulate)

exit
main ENDP
END main
```

Calculating the Sizes of Arrays

Masm uses $ operator (current location counter) to return the offset associated with current program statement. In the following example, `listsize` is calculated by subtracting the offset of list from the current **location counter ($):**

```
TITLE Symbolic Constants: Equal-Sign Directive
; Examples showing how to use Symbolic Constants.

INCLUDE Irvine32.inc

.data

      list BYTE 10,20,30,40
      listsize = ($ - list)

.code
main PROC

      mov eax, listsize
```

```
      call WriteDec
      call Crlf

exit
main ENDP
END main
```

Using EQU Directive

The EQU directive associates a symbolic name with either an integer expression or some arbitrary text. There are three formats

```
Name EQU expression
Name EQU symbol
Name EQU <text>
```

Where,

`expression` must be a valid integer expression

`symbol` is an existing symbol name

and any text may appear within the brackets <…>

```
TITLE Symbolic Constants: EQU Directive
; Examples showing how to use Symbolic Constants.

INCLUDE Irvine32.inc
.data

presskey EQU <"Press any key",0>

prompt BYTE presskey

.code
main PROC

      mov edx, OFFSET prompt
      Call WriteString
      Call Crlf

exit
main ENDP
END main
```

Using TEXTEQU Directive

It is very similar to the EQU directive. There are three different formats:

```
name TEXTEQU <text>
name TEXTEQU textmacro
name TEXTEQU %constExpr
```

The first assigns text, the second assigns an existing text macro, and the third assigns constant integer expression.

```
TITLE Symbolic Constants: Using TEXTEQU Directive
; Examples showing how to use Symbolic Constants.

INCLUDE Irvine32.inc
.data
      rowsize = 5
      count TEXTEQU %(rowsize*2)          ;same as: count TEXTEQU <10>
      move TEXTEQU <mov>
      setupEAX TEXTEQU <move eax,count>
                              ;same as: setupEAX TEXTQU <move eax, 10>
.code
main PROC
      setupEAX
      call WriteDec
      call Crlf
exit
main ENDP
END main
```