LAB - 12

Title:

Introduction to Assembly language

Introduction

Like any other programming language, Assembly language also follows strict Syntax rules.

To understand the Syntax please consider the following Skeleton.

Structure of program in assembly

```
TITLE Template      (Template.asm);  you can give your own title
; Program Description:                  ; The semi colon is used
to start a COMMENT
; Author:                    ; your name
; Creation Date:                    ; date and version
INCLUDE Irvine32.inc          ; INCLUDE is a directive to
Assembler
; Irvine32 is the Library name that gives you
;  built in routies
.data
; (you declare Variables here)
.code
Main PROC                                    ; executable code
begins here
; (write your code here)
Exit                                                ; return
control to Operating System
Main ENDP
; (insert additional procedure here we will do this is later
labs)
End main
```

All lines beginning with semicolon (;) are comments, are ignored by the assembler.

TITLE: The TITLE directive mark the entire line as a comment, you can put anything here.

Include Irvine32.inc: The include directive copies necessary definitions and setup information from a text file Irvine32.inc

.data: The .data directive identifies the area of a program that contains variables.

.code: The .code directive marks the beginning of the code segment, where all executable statements in a program located.

Main PROC: The PROC directive identifies the beginning of a procedure. The name chosen for the procedure in our program is main.

.Exit: The Exit statement calls a predefine MS-Window function that halts the program.

.ENDP: The ENDP directive marks the end of the main procedure.

END main: The END directive marks the last line of the program to be assembled.it identifies the name of program's startup procedure.

Defining Data:

Data Types available in Assembly Language

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

Defining variables:

```
Syntax:
       [Name] directive initializer [, initializer]….
Example:
.data
Val1 BYTE 10 h
Val2 BYTE 20h
```

Suppose that Val1 is located at offset 0 in the data segment and consumed one byte of storage. Then Val2 would be located at offset 1. Similarly you can define variable of type WORD, DWORD, SWORD etc.

```
.data
 Val1 WORD 10h
Val2 WORD 20h
```

Suppose Val1 is located at offset 0 in the data segment and consumed two byte of storage then Val2 would be located at offset 2.

A variable can be left uninitialized by using a question mark for the initializer

Val2 BYTE ?

Program#1

Now try the concepts introduced above in a Simple program to Add and subtract three numbers, and print the result.

(Please follow the template)

```
; declare following data
 .data
      Val1 DWORD 30000h
          Val2 DWORD 20000h
Val3 DWORD 10000h
; write following code
.code
main PROC
moveax,Val1 ;eax=30000h
add eax,Val2; eax=eax+Val2            ;50000h
sub eax,Val3      ;eax= eax-Val3           ;40000h
callDumpRegs; Call Irvine library procedure to print Register
Contents.
exit
main ENDP
END main
```

Explanation

Mov: mov instruction copies data from source to destination. In above instruction value of Val1 copies to eax . Here is a list of general variant of MOV

Mov reg, reg

Mov mem, reg

Mov reg, mem

Mov mem, immmediate

Mov reg, immediate

NOTE: Be careful when writing code
MOV is very flexible in its use of operands, as long as the following rules are observed:
- Both operands must be the same size.
- Both operands cannot be memory operands.
- CS, EIP, and IP cannot be destination operands.
- An immediate value cannot be moved to a segment register.

Add: the add instruction adds a source operand to a destination operand of the same size. In above instruction Val2 is added to eax .
Sub: The sub instructions subtract a source operand from a destination operand. In above instruction Val3 is subtracted from eax.
Call DumpRegs: The call DumpRegs procedure displays the EAX, EBX, ECX, EDX, ESI, EDI, ESP, EIP, EFL register in hexadecimal.

More concepts …

Defining Array: You can also define arrays in Assembly language.

If multiple initializers are used in same data definition, its label refers to only to the (Memory Address) offset of theFirst byte. But since consecutive memory locations are allocated by system , we can access any value by using Offset of first byte + Distance (in bytes) of the desired value.

```
Array of Byte:

.data
     List     BYTE    10,30,20,40
```

In this example ,assume that the label List is at OFFSET 0, If so the value 10 is at offset 0,30 is at offset 1,20 is at offset 2,and 40 is at offset 3

Array of Word

```
          .data
   List      WORD    10,30,20,40
```

In this example ,assume that the label List is at OFFSET 0,If so the value 10 is at OFFSET 0 ,30 is at offset 2, 20 is at offset 4 ,40 is at offset 6.

How to access array Element

Program#2

(Please follow template)

```
.data
array1 WORD 1000h, 3000h, 4000h

.code

Movesi Offset array1
Mov ax,[esi]                ;ax=1000h
Add esi,2
Mov ax,[esi]                 ;ax=3000h
Add esi,2
Mov ax,[esi]                  ;ax=4000h
```

DUP Operator

The Dup operator generates a repeated storage allocation, using a constant expression as a counter. It is particularly useful when allocating space for a string or array.

```
Byte 10 DUP(0)     ;10 bytes all equal to zero
Byte 10 DUP(?)      ;10 bytes ,uninitialized
```

Defining String

To create String data definition, enclose a sequence of character in quotation marks. The most common type of string ends with null byte, a byte containing the value 0.

```
Msg1 BYTE "hello world", 0
```

A string can be spread across multiple lines without the necessity of supplying a label for each Line

```
Msg1 Byte "An assembly language is a low-level
programming language",0dh,0ah
Byte "Assembly language uses a mnemonic",0dh,0ah
Byte " Many assemblers offer additional mechanisms",0
```

0dh,0ah: are called either CR/LF or end-of-line character.

Program#3

Display string on console.

(Please follow template)

```
.data
  Msg1 Byte "Enter a number:",0
.code
      Main PROC
```

```
Movedx , OFFSET msg1
      Call WriteString
      Call ReadInt
      Call WriteInt
      Exit
    Main ENDP
  END main
```

OFFSET: The offset operator returns the offset of a data label. The offset represent the distance, in bytes, of label from the beginning of the data segment
Writestring: The writestring procedure writes a null terminated string to standard output.
Call ReadInt: The ReadInt procedure reads a 32-bit signed integer from standard input and store value in eax register.
Call WriteInt: ThewriteInt procedure writes a 32-bit signed integer to standard output in decimal format. Before calling it,place the integer in eax.

LAB Tasks

1. Write a program that prints Sum , and Average of numbers stored in an array of integers.
2. Write a program that copies integers from Array1 to Array2

Link to Irvine Library Help

http://programming.msjc.edu/asm/help/index.html