LABORATORY MANUAL

CS/EEE/INSTR F241

Microprocessor Programming & Interfacing

K.R.Anupama

Lab2- Introduction to MASM

2.0 <u>Introduction</u>

Assembly language unlocks the secrets of computers hardware and software. An assembler converts source code to machine own language. Microsoft Macro Assembler (MASM), product of Microsoft Corporation includes several features which makes programming efficient and productive. The following chapter will give an overview how to use MASM for assembling the 80x86 program you code.

<u>Editor</u>

Go to directory MASM611\BIN type – use notepad ++ as the editor.

<u>2.0 Program Template</u>

<u>2.1 Using Model Tiny</u>

All the data and code fit in one segment Tiny programs when compiled give .COM executable - the program must be originated at location 100_H Memory Models map the segments in your program to the segments in memory. When using the model command you need not initialize the segment register this is done by the assembler itself.

.Model Tiny			
.386			;Assembler by default accepts only 8086/8088 instructions, unless a program is preceded by .386/.486 directive to select the microprocessor
.data			; Data Segment
COUNT	EQU	32_H	; variable count is given a value 32 – this is not stored in memory
VAL1	EQU	0030_H	
DAT1	DB	$45_H,67_H,100,$'A'	; DB- Define byte stores bytes of data in memory – starting addr of these bytes is labelled as DAT 1
WRD	DW	$10_H, 3500_H,0910_H$	;DW- Define word stores 16-bit data in memory
DAT2	DD	0902_H	; DD-Define Double word stores 32-bit data in memory
DAT3	DW	2 DUP(0)	;DUP – duplicate command reserves locations in memory for 2 16-bit data and initializes it to '0'
DAT4	DB	56_H	
RES	DB	10 DUP(?)	;DUP – reserves locations in memory for 10 8-bit data, ? indicates that no default data is stored
DWRD	DD	01020304_H	

```
        .CODE
        .STARTUP
                    MOV         SI,DAT3
                    MOV         AL, DAT1 + 1
                    MOV         BX,WORD PTR DAT1+4
                    ADD         BX,20H
                    MOV         AI,[BX]
                    LEA         BX,DAT4
                    MOV         AL,[BX]
                    MOV         BX,VAL1
                    MOV         AL,FS:[BX]
                    MOV         EBX, DWRD
        .EXIT
END
```

2.1 Assembling

There are two methods of assembling

Method 1:

 Type MASM filename.asm <enter>

If no error in code there is .OBJ file is generated

Now type

LINK filename.obj <enter>

Check the files created at each step and examine the content of .lst and .map file

Method 2:

 Typ'e ML filename.asm <enter>

To create list and map file command format is 'ml /Fl /Fm Filename.asm

 Check the files created at each step and examine the content of .lst and .map file

What is the difference between Method 1 and Method 2.

To check the working of the program – execute **debugx Filename.com** and then trace or go in debugx.

2.1 Using Model Small

Questions

(1)What are the errors if you just change Tiny to Small in the .Model Statement?

(2) Is there a PSP in .Model Small?

(3) Remove .386 statement. When you assemble what is the error? Why is there an error?

To check the working of the program – execute ***debugx Filename.exe*** and then trace or go in debugx.

Note:

1. **All your files .asm, .com/.exe must be present in MASM611/BIN**
2. **Make sure that you copy debugx.exe MASM611/BIN**

Tasks:

1. Write an ALP that finds the maximum number from a set of 32-bit numbers
2. Write an ALP to add 2 16-byte nos. using them
   a. as 16-bit data
   b. as 32-bit data
3. Write an ALP that will examine the contents of set of 10 bytes starting from location *'ARRAY1'* for the presence of data '0A$_H$' and replace it with the ASCII character 'E'.
4. Write an ALP that will count the number of negative numbers in an array of 16-bit signed data stored from location 'ARRAY1'. The number of elements in the array is present in location 'COUNT'. The count of negative numbers must be stored in location 'NEG1'
5. Write an ALP that will transfer data from *'ARRAY1'* to *'ARRAY2'.* The number of elements in the array is 10. The array is a double word array. The starting address of *ARRAY2* = starting address of *ARRAY1* + 20$_d$.