Lab Manual for Computer Organization and Assembly Language

Lab-11

Macros

Table of Contents

1. Introduction

Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. If you declared a macro and never used it in your code, compiler will simply ignore it. emu8086.inc is a good example of how macros can be used, this file contains several macros to make coding easier for you.

2. Objective

- To know more about Assembly language, such as how to work with macros structure in assembly language.

3. Concept Map

This section provides you the overview of the concepts that will be discussed and implemented in this lab.

3.1 Macros

```
Macro definition:

name    MACRO [parameters,...]

          <instructions>

ENDM
```

Unlike procedures, macros should be defined above the code that uses it, for example:

```
MyMacro    MACRO p1, p2, p3

    MOV AX, p1
    MOV BX, p2
    MOV CX, p3

ENDM

ORG 100h

MyMacro 1, 2, 3

MyMacro 4, 5, DX

RET
```

The above code is expanded into:

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```

Some important facts about **macros** and **procedures**:

- When you want to use a procedure, you should use **CALL** instruction, for example: CALL MyProc

- When you want to use a macro, you can just type its name, for example: MyMacro

- Procedure is located at some specific address in memory, and if you use the same procedure 100 times, the CPU will transfer control to this part of the memory. The control will be returned back to the program by **RET** instruction. The **stack** is used to keep the return address. The **CALL** instruction takes about 3 bytes, so the size of the output executable file grows very insignificantly, no matter how many times the procedure is used.

- Macro is expanded directly in program's code. So, if you use the same macro 100 times, the compiler expands the macro 100 times, making the output executable file larger and larger, each time all instructions of a macro are inserted.

- You should use **stack** or any general-purpose registers to pass parameters to procedure.

- To pass parameters to macro, you can just type them after the macro name. For example: MyMacro 1, 2, 3

- To mark the end of the macro **ENDM** directive is enough.

- To mark the end of the procedure, you should type the name of the procedure before the **ENDP** directive.

Macros are expanded directly in code, therefore if there are labels inside the macro definition you may get "Duplicate declaration" error when macro is used for twice or more. To avoid such problem, use **LOCAL** directive followed by names of variables, labels or procedure names. For example:

```
MyMacro2   MACRO
           LOCAL label1, label2

           CMP AX, 2
```

```
        JE label1
        CMP AX, 3
        JE label2
        label1:
                INC AX
        label2:
                ADD AX, 2
ENDM


ORG 100h

MyMacro2

MyMacro2

RET
```

If you plan to use your macros in several programs, it may be a good idea to place all macros in a separate file. Place that file in **Inc** folder and use **INCLUDE *file-name*** directive to use macros.

Library Usage

To make programming easier there are some common functions that can be included in your program. To make your program use functions defined in another file you should use the **INCLUDE** directive followed by a file name. Compiler automatically searches for the file in the same folder where the source file is located, and if it cannot find the file there - it searches in **Inc** folder.
Currently you may not be able to fully understand the contents of the **emu8086.inc** (located in **Inc** folder), but it's OK, since you only need to understand what it can do.
To use any of the functions in **emu8086.inc** you should have the following line in the beginning of your source file:
include 'emu8086.inc'
emu8086.inc defines the following **macros**:

- **PUTC char** - macro with 1 parameter, prints out an ASCII char at current cursor position.
- **GOTOXY col, row** - macro with 2 parameters, sets cursor position.
- **PRINT string** - macro with 1 parameter, prints out a string.
- **PRINTN string** - macro with 1 parameter, prints out a string. The same as PRINT but automatically adds "carriage return" at the end of the string.
- **CURSOROFF** - turns off the text cursor.
- **CURSORON** - turns on the text cursor.

To use any of the above macros simply type its name somewhere in your code, and if required parameters, for example:

```
include emu8086.inc
ORG    100h
PRINT 'Hello World!'
GOTOXY 10, 5
PUTC 65         ; 65 is an ASCII code for “A”
PUTC 'B'
RET
END
```

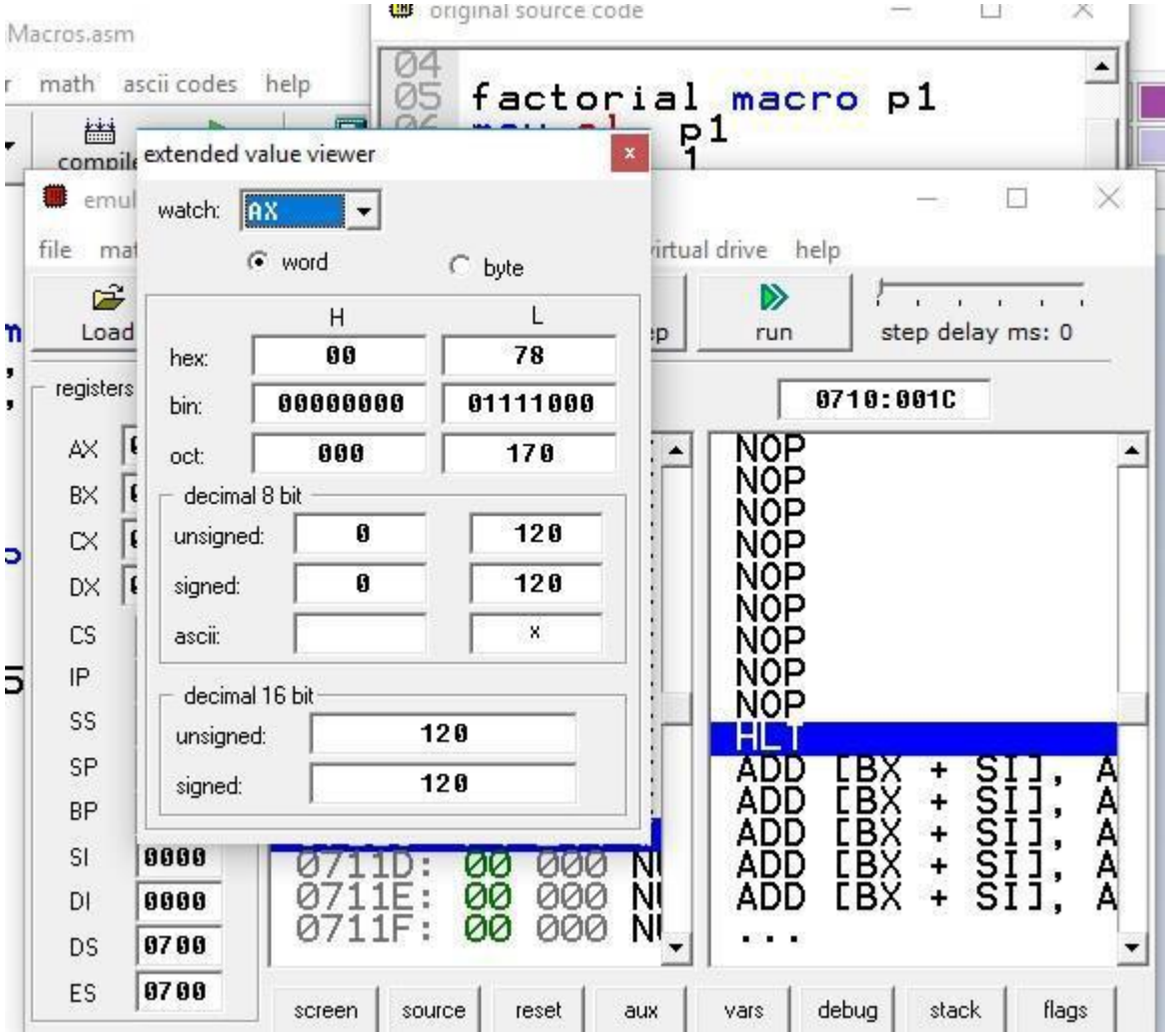
When compiler process your source code it searches the **emu8086.inc** file for declarations of the macros and replaces the macro names with real code. Generally, macros are relatively small parts of code, frequent use of a macro may make your executable too big (procedures are better for size optimization).

4. Walkthrough Task

The following program will find the factorial of a given number.

```
01 .model small
02 .data
03 .code
04
05     factorial macro p1
06         mov cl, p1
07         mov al, 1
08
09         while:
10             mul cl
11
12             loop while
13
14     endM
15
16     factorial 5
17
18
```

The output of the program is shown below.



5. Procedure& Tools

In this section you will study how to setup and MASM Assembler.

5.1 Tools

- Download emu 8086 from (http://www.emu8086.com/files/emu8086v408r11.zip)
- Just extract the emu8086.15.zip on C
- Install emu8086

6. Practice Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time. When you finish them, put these tasks in the following folder:

\\fs\assignments$\

6.1 Practice Task 1 [Expected time = 15mins]

Write a program to make macro for taking input of 5 numbers and printing their sum. Call the macro 5 times using the loop and each time change one value and keep rest for values same.

6.2 Practice Task 2 [Expected time = 15mins]

Write a program to make macro to calculate the product of two numbers. The program will continuously ask user to enter two numbers from user and find their product unless user enters both numbers zero.

6.3 Practice Task 3 [Expected time = 15mins]

Write a program to print the triangle pattern. Use loops to print pattern and print code should be written as macro. Call the macro repeatedly to print the triangle.

```
   *
  **
 ***
****
*****
```

7. Out comes

After completing this lab, student will be able to understand and work with macros in assembly language.

8. Evaluation Task (Unseen) [Expected time = 30mins for tasks]

The lab instructor will give you unseen task depending upon the progress of the class.

8.1 Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 3: Evaluation of Lab

Sr. No.	Task No	Description	Marks
1	4	Problem Modeling	20
2	6	Procedures and Tools	10
3	7	Practice tasks and Testing	35
4	8	Evaluation Tasks (Unseen)	20
5		Comments	5
6		Good Programming Practices	10

9. Further Reading

This section provides the references to further polish your skills.

9.1 Slides

The slides and reading material can be accessed from the folder of the class instructor available at \\fs\lectures$