Lab Manual for Computer Organization and Assembly Language

Lab-2

Assembly Language Fundamentals

Table of Contents

1. Introductions

In this Lab we will learn to define variable with initialization as well as in term of taking inputs from user. further that we will perform output operations for variable and check how data is being manipulated in registers and variables.

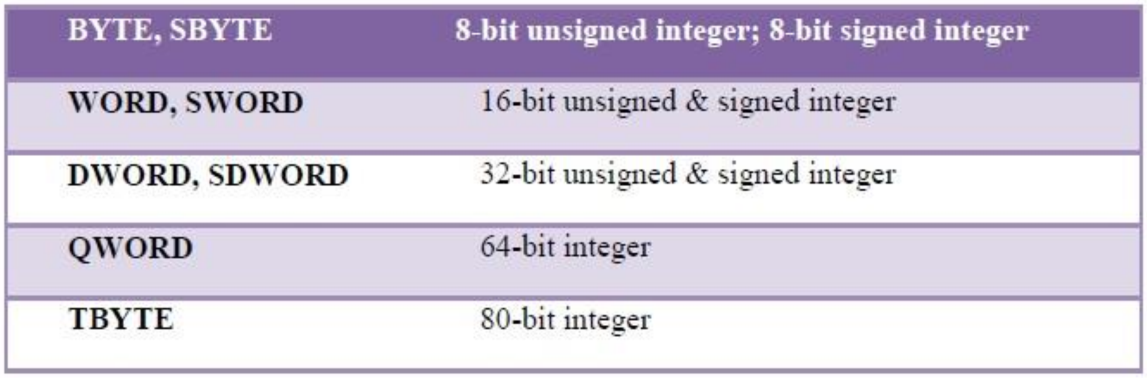
2. Objective

- To be familiar with Assembly Language Fundamentals.

3. Concept Map

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

3.1 Data Types

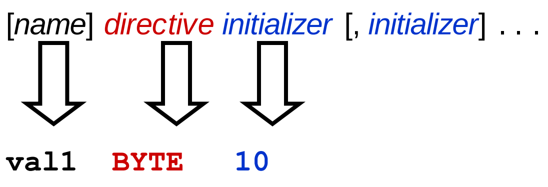


BYTE, SBYTE	8-bit unsigned integer; 8-bit signed integer
WORD, SWORD	16-bit unsigned & signed integer
DWORD, SDWORD	32-bit unsigned & signed integer
QWORD	64-bit integer
TBYTE	80-bit integer

3.1.1 Data Definition Statement:

A data definition statement sets aside storage in memory for a variable.

Syntax:



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

```
val1  BYTE   10
```

All initializers become binary data in memory

We can define a single byte of storage; use multiple initializers, defining Strings.

End-of-line character sequence:

- 0Dh = carriage return

- 0Ah = line feed

```
str1 db "Enter your name:   ",0Dh,0Ah
"Enter your address: $”
```

3.1.2 Integer Constants

An *integer constant* (or integer literal) is made up of an optional leading sign, one or more digits and an optional suffix character (called a *radix)* indicating the number's base:

[{+ | - }] *digits [radix]*

Common radix characters:
h – hexadecimal d
– decimal b –
binary
o – octal

Examples: 30d, 6Ah, 42, 1101b, 777o
Hexadecimal beginning with letter: 0A5h

- If no radix is given, the integer constant is decimal
- A hexadecimal beginning with a letter must have a leading 0

3.1.3 Integer Expressions

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

3.1.4 Character Constants

A *character constant* is a single character enclosed in either single or double quotes. The assembler converts it to the binary ASCII code matching the character. Examples are: 'A'
"x"
ASCII character = 1 byte.

3.1.5 String Constants

String constant is a string of characters enclosed in either single or double quotes:

```
'ABC'
"xyz"
'Say "Goodnight," Gracie'
```

Each character occupies a single byte.

3.1.6 Reserved Words

Assembly language has a list of words called *reserved words*. These have special meaning and can only be used in their correct context. Reserved words can be any of the following:

- Instruction mnemonics, such as MOV, ADD or MUL which correspond to built-in operations performed by Intel processors.
- Directives, which tell MASM how to assemble programs.
- Used to declare code, data areas, select memory model, declare procedures, etc.
- not case sensitive
- For example, .data, .code, proc.
- Attributes, which provide size and usage information for variables and operands. Examples are BYTE and WORD.
- Operators, used in constant express ions. For example, (+, -, …)
- Predefined symbols. Such as **@data** which return constant integer values at assembly time.

3.1.7 Identifiers

- They are not case sensitive.
- The first character must be a letter (A. Z, a..z), underscore (_), @ , ?, or $. Subsequent characters may also be digits.
- An identifier cannot be the same as an assembler reserved word.

3.1.8 Comments

Comments can be specified in two ways:

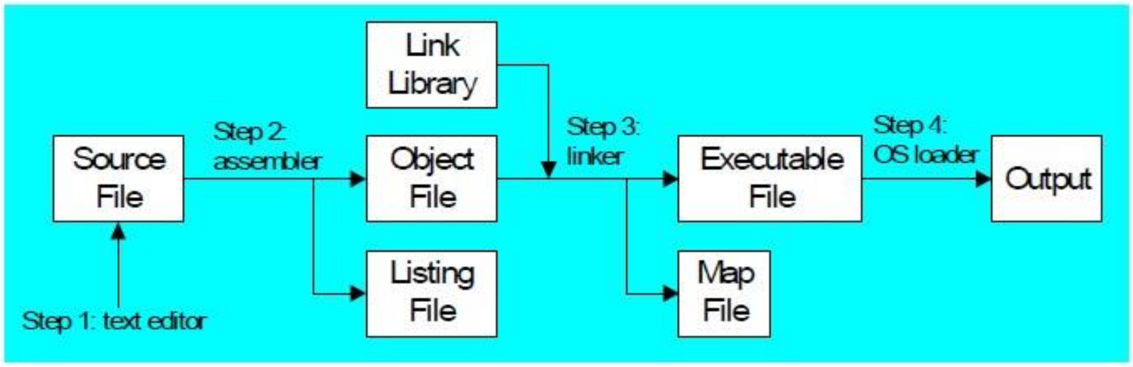
- Single-line comments, beginning with a semicolon character (**;**). All characters following the semicolon on the same line are ignored by the assembler and may be used to comment the program.
- Block comments, beginning with the COMMENT directive and a user-specified symbol.

All subsequent lines of text are ignored by the assembler until the same user-specified symbol appears. For example:

COMMENT ! This line is a comment. This line is also a comment. !	COMMENT & This line is a comment. This line is also a comment. &

3.2 Assemble-Link Execute Cycle (Assignments)

The following diagram describes the steps from creating a source program through executing the compiled program:



Note: If the source code is modified, steps 2 through 4 must be repeated.

.386:

- Enables assembly of no privileged instructions for the 80386 processor; disables assembly of instructions introduced with later processors.
- The **.386** directive identifies the minimum CPU required for the program (Intel386).

Used after the. model directive in our small model. In other models it is used before. model directive.

4. Procedure& Tools

In this section, you will study how to setup and emu 8086.

4.1 Tools

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

5. 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$\

5.1 Practice Task 1 [Expected time = 15mins]

Print the following output on the command window using 0Dh, 0Ah Operator:

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

5.2 Practice Task 2 [Expected time = 15mins]

Write the following message on the console using (0Ah = line feed):

```
This
     is
       my
          “Name”
```

5.3 Practice Task 3

Print the following patterns in console using assembly language.

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

5.4 Out comes

After completing this lab, student will be able to setup Emu 8086. He/ She will also be able to compile and run basic Assembly programs.

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

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

7. 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 the 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

8. Further Reading

This section provides the references to further polish your skills.

8.1 Slides

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