Daffodil International University

Department of Computer Science & Engineering

Lab Manual

Version: 2017.04

Course Code: **CSE 232**

Course Title: **Microprocessor and Assembly language Lab**

Table of Contents

Session 1: Introduction to Assembly Language Programming Environment

Intended Learning Outcome:

a. Introduction to Assembly Language Tools and Familiarization with Emu8086 environment.
b. Learn to install EMU 8086 and execute sample assembly program

Expected Skills:

a. Capability of installing EMU 8086 and working with it.

b. Write, compile and execute assembly language programs using EMU 8086.

Tools Required:

a. EMU 8086

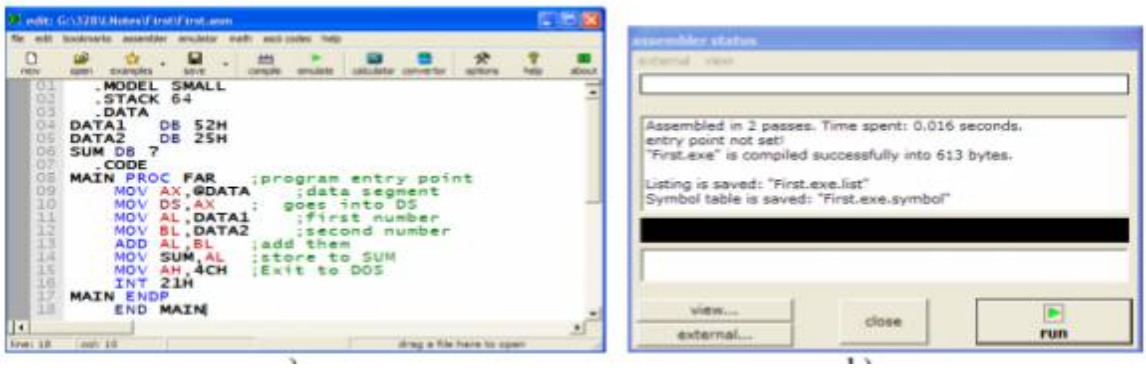
Session Detail:

Emu8086

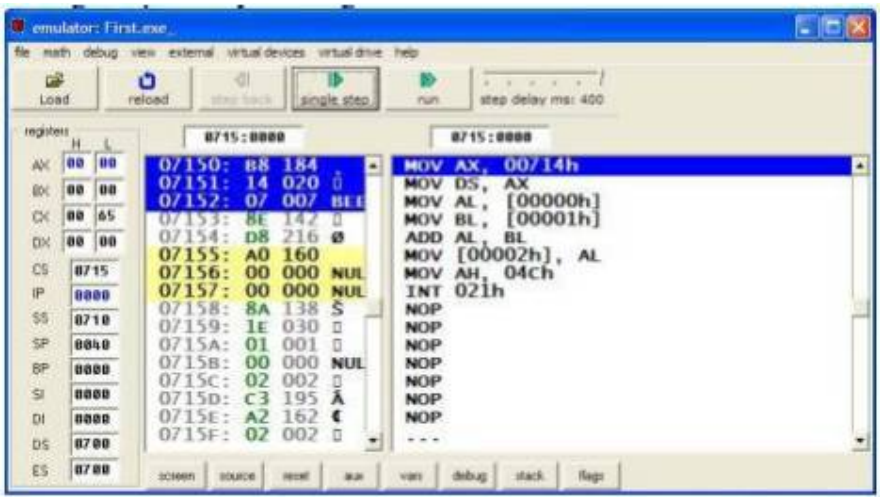
Emu8086 combines an advanced source editor, assembler, disassemble and software emulator (Virtual PC) with debugger. It compiles the source code and executes it on emulator step by step. Visual interface is very easy to work with. You can watch registers, flags and memory while your program executes. Arithmetic & Logical Unit (ALU) shows the internal work of the central processor unit (CPU). Emulator runs programs on a Virtual PC, this completely blocks your program from accessing real hardware, such as hard-drives and memory, since your assembly code runs on a virtual machine, this makes debugging much easier. 8086 machine code is fully compatible with all next generations of Intel's microprocessors, including Pentium II and Pentium 4. This makes 8086 code very portable, since it runs both on ancient and on the modern computer systems. Another advantage of 8086 instruction set is that it is much smaller, and thus easier to learn.

EMU8086 Source Editor

The source editor of EMU86 is a special purpose editor which identifies the 8086 mnemonics, hexadecimal numbers and labels by different colors as seen in Figure below.



The compile button on the taskbar starts assembling and linking of the source file. A report window is opened after the assembling process is completed. Figure below shows the emulator of 8086 which gets opened by clicking on emulate button.



Sample Programs to practice on the lab (Programs to be performed may be decided by the course teacher):

Program 1.1

Source: Example 4.1 in section 4.9 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 2: Introduction to basic syntax of Assembly language

Intended Learning Outcome:

a. Introduction to Assembly Language basic syntaxes
b. Use these syntaxes to solve small problems

Expected Skills:

a. Use of basic I/O, movement & arithmetic instructions.

Tools Required:

a. EMU 8086

Session Detail:

Basic syntax will be discussed with example and then they will be asked to implement the solution of some small problems.

I/O DOS Function Calls: Table below summarizes the main I/O functions. These functions are mainly used to read a character or a string from the keyboard, which could be an input data to a program, and display characters or strings, which could be results, or an output, of a program:

Function	Input in	Output in	Effect
01H	AH	AL	Read a character with echo on the screen.
02H, 06H	AH, Character in DL	No output	Display a character on the screen. Note: Interrupted by Ctrl + Break
08H	AH	AL	Read character without echo.
09H	AH	No output	Display a string terminated by a '$' sign
0AH	AH	Offset in DX	Read a string of characters from the keyboard

Program 2.1:

Write a program in assembly to read a character from the keyboard and display on the screen using interrupt 21H.

Source: Ref. book [1]

Program 2.2:

Write a program to displays a string terminated by a $ sign using INT 21H function 09H.

Source: Example 4.2 in section 4.9 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 3: Arithmetic Operations in Assembly Language

Intended Learning Outcome:

a. Learn to implement arithmetic operations on data
b. Learn to use these operations to solve problems

Expected Skills:

a. Ability to solve problems using arithmetic & other basic instructions

Tools Required:

a. EMU 8086

Session Detail:

Students will be introduced with basic arithmetic operations and they will be asked to solve problems using these operations.

Program 3.1:

Write a program that reads two numbers from the keyboard and gives their sum as output.

Source:

```
.MODEL SMALL
.STACK  100H
.DATA
      CRLF    DB 0DH,0AH,'$'
      PROMPT1 DB 'Enter the first positive integer: ','$'
      PROMPT2 DB 'Enter the second positive integer: ','$'
      PROMPT3 DB 'The sum of the two numbers is: ','$'
.CODE
MAIN PROC
      LEA DX,PROMPT1   ;DISPLAY PROMPT1
      MOV AH,09H
      INT 21H

      MOV AH,01H      ;READ FIRST NUMBER
      INT 21H
      SUB AL,30H ;Convert character to number
      MOV CL,AL  ;SAVE THE NUMBER IN CL

      LEA DX,CRLF     ;MOVE CURSOR TO NEXT LINE
      MOV AH,09H
      INT 21H
      LEA DX,PROMPT2 ;DISPLAY PROMPT2
      MOV AH,09H
      INT 21H

      MOV AH,01H      ;READ SECOND NUMBER
      INT 21H
      SUB AL,30H ;Convert character to number
```

```
    ADD AL,CL  ;PERFORM ADDITION AND SAVE RESULT IN CL

    MOV CL,AL
    ADD CL,30H ;CONVERT DIGIT TO CHARACTER

    LEA DX,CRLF    ;MOVE CURSOR TO NEXT LINE
    MOV AH,09H
    INT 21H
    LEA DX,PROMPT3 ;DISPLAY PROMPT3
    MOV AH,09H
    INT 21H

    MOV DL,CL  ;DISPLAY SUM
    MOV AH,02H
    INT 21H
MAIN ENDP
END MAIN
```

Program 3.2:

Write a program to convert a given lowercase letter to its uppercase form and prints it.

Source: Example 4.3 in section 4.9 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 4: Branching operations in assembly language

Intended Learning Outcome:

a. Learn to implement branching instructions in assembly language
b. Learn to use these instructions to solve problems

Expected Skills:

a. Aptitude to solve more complex decision making problems using branching instructions

Tools Required:

a. EMU 8086

Session Detail:

Students will be introduced with branching operations and they will be asked to solve problems using these operations.

Compare instruction:

The compare instruction is used to compare two numbers. At most one of these numbers may reside in memory. The compare instruction subtracts its source operand from its destination operand and sets the value of the status flags according to the subtraction result. The result of the subtraction is not stored anywhere. The flags are set as indicated in Table below.

Instruction	Example	Meaning
CMP	CMP AX, BX	If (AX = BX) then ZF ← 1 and CF ← 0
		If (AX < BX) then ZF ← 0 and CF ← 1
		If (AX > BX) then ZF ← 0 and CF ← 0

Jump Instructions:

The jump instructions are used to transfer the flow of the process to the indicated operator.
An overview of all the jump instructions is given in Table below.

Signed Jumps

Symbol	*Description*	*Condition for Jumps*
JG/JNLE	jump if greater than jump if not less than or equal to	ZF = 0 and SF = OF
JGE/JNL	jump if greater than or equal to jump if not less than or equal to	SF = OF
JL/JNGE	jump if less than jump if not greater than or equal	SF <> OF
JLE/JNG	jump if less than or equal jump if not greater than	ZF = 1 or SF <> OF

Unsigned Conditional Jumps

Symbol	*Description*	*Condition for Jumps*
JA/JNBE	jump if above jump if not below or equal	CF = 0 and ZF = 0
JAE/JNB	jump if above or equal jump if not below	CF = 0
JB/JNAE	jump if below jump if not above or equal	CF = 1
JBE/JNA	jump if equal jump if not above	CF = 1 or ZF = 1

Single-Flag Jumps

Symbol	Description	Condition for Jumps
JE/JZ	jump if equal jump if equal to zero	ZF = 1
JNE/JNZ	jump if not equal jump if not zero	ZF = 0
JC	jump if carry	CF = 1
JNC	jump if no carry	CF = 0
JO	jump if overflow	OF = 1
JNO	jump if no overflow	OF = 0
JS	jump if sign negative	SF = 1
JNS	jump if nonnegative sign	SF = 0
JP/JPE	jump if parity even	PF = 1
JNP/JPO	jump if parity odd	PF = 0

Program 4.1:

Suppose AL and BL contains extended ASCII characters. Display the one that comes first in the character sequence.

Source: Example 6.2 in reference book [1].

Program 4.2:

IF AX contains a negative number, put -1 in BX; if AX contains 0, put 0 in BX; if AX contains a positive number, put 1 in BX.

Source: Example 6.3 in reference book [1].

Program 4.3:

Read a character and if it's an uppercase letter, display it.

Source: Example 6.4 in reference book [1].

Program 4.4:

Read a character. If it's "y" or "Y", display it; otherwise, terminate the program.

Source: Example 6.6 in reference book [1].

Program 4.5:

Read two numbers and print the maximum number.

Source: Left as an exercise.

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 5: Looping operations in assembly language

Intended Learning Outcome:

a. Learn to implement looping instructions in assembly language
b. Learn to use these instructions to solve problems

Expected Skills:

a. Capability of solving problems using looping techniques.

Tools Required:

a. EMU 8086

Session Detail:

Students will be introduced with looping operations and they will be asked to solve problem using these operations.

Types of looping:

- For
- While
- Do While

In this lab session we will see the use of all of these looping operations.

The LOOP Instructions:

The LOOP instruction is a combination of a DEC and JNZ instructions. It causes execution to branch to the address associated with the LOOP instruction. The branching occurs a number of times equal to the number stored in the CX register.

Program 5.1:

Write a count-controlled loop to display a row of 80 stars.

Source: Example 6.8 in reference book [1].

Program 5.2:

Write some code to count the number of characters in an input line.

Source: Example 6.9 in reference book [1].

Program 5.3:

Write some code to read characters until a blank is read.

Source: Example 6.10 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 6: Solving complex problems using branching and looping operations

Intended Learning Outcome:

a. Learn to more about combining branching and looping operations for solving problems
b. Learn to use these operations to solve problems

Expected Skills:

a. Aptitude to solve problems associated with these operations

Tools Required:

a. EMU 8086

Session Detail:

Program 6.1:

Prompt the user to enter a line of text. On the next line, display the capital letter entered that comes first alphabetically and the one that comes last. If no capital letters are entered, display "No capital letters".

Source: Example 6.11 in section 6.5 in reference book [1].

*More practice problems like the one above can be used by the course teacher.

Program 6.2:

Store the sum of the series 1 + 4 + 7+ ...+ 148 in AX.

Source: Left as an exercise.

*More practice problems like the one above can be used by the course teacher.

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 7 & 8: Logic, Shift and Rotate operations

Intended Learning Outcome:

a. Learn to implement logical operations on data
b. Learn to use these operations to solve problems

Expected Skills:

a. Ability to solve problems associated with these operations

Tools Required:

a. EMU 8086

Session Detail:

Students will be introduced with basic logical operations and they will be asked to solve problems using these operations.

Logical Instructions:

Logic shift and rotate instructions are called bit manipulation operations. These operations are designed for low-level operations, and are commonly used for low level control of input/output devices. The list of the logic operations of the 8086 is given below along with examples.

Instruction	Example	Meaning
AND	AND AX, FFDFH	AX ← AX AND FFDFH
OR	OR AL, 20H	AL ← AL OR 20H
XOR	XOR NUM1, FF00	[NUM1]←[NUM1]XOR FF00
NOT	NOT NUM2	$[NUM2] \leftarrow \overline{[NUM2]}$

The Shift Operations:

The shift operations are used to multiply or divide a number by another number that is a power of 2 (i.e. 2^n or 2^{-n}). Multiplication by 2 is achieved by a one-bit left shift, while division by 2 is achieved by a one-bit right shift. The Shift Arithmetic Right (SAR) instruction is used to manipulate signed numbers. The regular Right Shift (SHR) of a signed number affects the sign bit, which could cause numbers to change their sign. The SAR preserves the sign bit by filling the vacated bits with the sign of the number. Shift Arithmetic Left (SAL) is identical in operation to SAR.

The Rotate Operations:

The rotate operations are very similar to the shift operations, but the bits are shifted out from one end of a number and fed back into the other end to fill the vacated bits. They are provided to facilitate the shift of long numbers (i.e. numbers of more than 16 bits). They are also used to

reposition a certain bit of a number into a desired bit location. The rotate right or left instructions through the carry flag (RCL and RCR) are similar to the regular rotate instructions (ROL and ROR), but the carry flag is considered as a part of the number. Hence, before the rotate operation, the carry flag bit is appended to the number as the least significant bit in the case of RCL, or as the most significant bit in the case of RCR.

Type	Instruction	Example	Meaning
Shift	SHL	SHL AX,1	Shift AX left by 1 bit. Fill vacated bit with 0.
	SAL	SAL AX,1	Shift AX left by 1 bit. Fill vacated bit with 0.
	SHR	SHR NUM2,CL	Shift NUM2 right by the number of bits indicated in CL. Fill vacated bits with 0.
	SAR	SAR NUM2,CL	As SHR but fill vacated bits with the sign bit.
Rotate	ROL	ROL BH,CL	Same as SHL, but shifted bits are fed back to fill vacated bits.
	RCL	RCL BH,CL	Same as ROL, but carry flag is appended as MSB, and its content is shifted with the number.
	ROR	ROR NUM1,1	Same as SHR, but shifted bits are fed back to fill vacated bits.
	RCR	RCR NUM1,1	Same as ROR, but carry flag is appended as LSB, and its content is shifted with the number.

Program 7.1:

This program shows the effect of the logic instructions.

Source:

```
.MODEL SMALL
.STACK 100H
.DATA
NUM1 DW 0FA62H
NUM2 DB 94H
.CODE

MAIN PROC
        MOV AX,  NUM1  ;load AX with number NUM1
        AND  AX,  0FFDFH  ;Reset 6th bit of AX
        OR AL,  20H  ;Set 6th bit of AL
        XOR NUM1, 0FF00H; Complement the high order byte of NUM1
        NOT NUM2   ; Complement NUM2
        XOR AX, AX   ; Clear AX
        MOV AX, NUM1
        AND AX, 0008H; Isolate bit 4 of NUM1
        XOR AX, 0080H; Complement 4th bit of AX
```

```
MAIN PROC
END MAIN
```

Program 7.2:

Use ROL to count the number of I bits In BX, without changing BX. Put the answer In AX.

Source: Example 7.12 in reference book [1].

Program 7.3:

Reverse a binary number stored in AL.

Source: Example 7.13a in reference book [1] pp. no. 130.

Program 7.4:

Read a binary number from input and store it in AX.

Source: Section 7.4 in reference book [1].

Program 7.5:

Write a binary number stored in AX into the output.

Source: Section 7.4 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 9: Solving problems using Stack

Intended Learning Outcome:

a. Learn to implement stack in assembly language

b. Learn to use stack as a means to solve relevant problems

Expected Skills:

a. Capability to solve problems using stack.

Tools Required:

a. EMU 8086

Session Detail:

Students will be acquainted with stack and its implementation and then they will be given relevant problems which can be solved using stack.

The Stack:

The stack is a special segment in memory used to facilitate subroutine handling. The SS register contains the Stack Segment number where the stack is stored. The ".STACK" directive instructs the assembler to reserve a stack segment of a desired size. The stack always starts at a high address and grows towards the beginning of the stack segment at a lower address. When a program starts, the stack is empty, and its size is zero. The microprocessor stores data on the stack as needed, and uses the SP register to point to the last item stored on the stack. The stack size dynamically changes as data is stored or retrieved from the stack.

Instruction	Example	Meaning
PUSH	PUSH AX	[SP] ← AH [SP-1] ← AL SP ← SP –2
POP	POP NUM1	[NUM1] ← [SP] [NUM+1] ← [SP+1] SP ← SP + 2
PUSHF	PUSHF	[SP-1] ← MSB(FR) [SP-2] ← LSB(FR) SP ← SP –2
POPF	POPF	LSB(FR) ←[SP] MSB(FR) ←[SP+1] SP ← SP + 2

Note: FR = Flag Register

Program 9.1:

Write a program that read a string and prints its reverse string.

Source: Section 8.2 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.

Session 10: Solving problems using string manipulation operations

Intended Learning Outcome:

a. Learn to use string manipulation operations in assembly language

b. Learn to solve relevant problems using string manipulation

Expected Skills:

a. Capability of solving problems using string manipulation operations.

Tools Required:

a. EMU 8086

Session Detail:

String Handling Instructions:

String handling instructions are very powerful because they allow the programmer to manipulate large blocks of data with relative ease. Block data manipulation occurs with the string instructions. Each of the string instructions defines an operation for one element of a string only. Thus, these operations must be repeated to handle a string of more than one element.

String handling instructions use the direction flag, SI and DI registers. The Direction Flag (DF) selects auto-increment or auto-decrement operation for the DI and SI registers during string operations. Whenever a string instruction transfers a byte, the contents of SI and/or DI get increased or decreased by 1. If a word is transferred, the contents of SI and/or DI get increased or decreased by 2.

Basic String Handling Instructions

Mnemonics	Meaning	Format	Operation As per Direction Flag	Flags affected
LODS	Load string	LODSB LODSW	(AL or AX) ← ((DS)0+(SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB STOSW	((ES)0 + (DI)) ← (AL or AX)) (DI) ← (DI) ± 1 or 2	None
MOVS	Move string	MOVSB MOVSW	((ES)0 + (DI)) ← ((DS)0+(SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

String Movement Instructions

Mnemonics	Meaning	Format	Operation As per Direction Flag	Flags affected
LODS	Load string	LODSB LODSW	(AL or AX) ← ((DS)0+(SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB STOSW	((ES)0 + (DI)) ← (AL or AX)) (DI) ← (DI) ± 1 or 2	None
MOVS	Move string	MOVSB MOVSW	((ES)0 + (DI)) ← ((DS)0+(SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

Note: B stands for Byte and W for Word.

String Comparison Instructions

Mnemonics	Meaning	Format	Operation	Flags affected
CMPS	Compare strings	CMPSB CMPSW	Set flags as per: ((ES)0 + (SI)) – ((ES)0+(DI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF,PF,AF, ZF,SF,OF
SCAS	Scan string	SCASB SCASW	Set flags as per: (AL or AX) – ((ES)0+(DI)) (DI) ← (DI) ± 1 or 2	CF,PF,AF, ZF,SF,OF

Program 10.1:

Write a program that reads and stores *a* character string.

Source: Program listing 11.1 in chapter 11 in reference book [1].

Program 10.2:

Write a program that writes a character string in the output.

Source: Program listing 11.2 in chapter 11 in reference book [1].

Program 10.3:

Write a program that reads a character string and displays the number of vowels and consonants in the output.

Source: Program listing 11.4 in chapter 11 in reference book [1].

Post Lab Exercise:

Individual Assignments will be given based on the Skills developed in this session by the course teacher.

References:

1. Assembly Language Programming and Organization of the IBM PC by Ytha Yu, Charles Marut.

Further Readings:

1. Assembly Language for x86 Processors by Kip Irvine.