Laboratory Manual

for

Computer Organization and Assembly Language

Course Instructors

Lab Instructor(s)

Section

Semester

Department of Computer Science

COAL Lab 3Manual

Objectives:

- Structure of an Assembly language program
- Protected mode
  - Overview of flat Segmentation
- Basic instructions set
- Example

1. Structure of an Assembly language code:

An assembly language program is written according to the following structure and includes the following assembler directives. Each of the segments is called a logical segment. Depending upon the memory, the code and the data segments may be in the same or different physical segments.

```
TITLE Program Template (Template.asm)
; Program Description:
; Author:
; Creation Date:
; Revisions:
; Date:
INCLUDE Irvine32.inc
.data
; (insert variables here)
.code
main PROC
; (insert executable instructions here)
exit
main ENDP
; (insert additional procedures here)
END main
```

Fig. 3.1: Assembly Program Template

- `INCLUDE Irvine32.inc`

The INCLUDE directive copies necessary definitions and setup information from a text file named *Irvine32.inc*, located in the assembler's INCLUDE directory.

- `.code`

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

- `main PROC`

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

- `exit`
  `main ENDP`

The **exit** statement (indirectly) calls a predefined MS-Windows function that halts the program.The ENDP directive marks the end of the **main** procedure. Note that **exit** is not a MASM keyword; instead, it's a macro command defined in the *Irvine32.inc* include file that provides a simple way to end a program.

- `END main`

The END directive marks the last line of the program to be assembled. It identifies the name of the program's *startup* procedure (the procedure that starts the program execution).

2. Protected Mode

a. Linear address space is 4 GBytes, using addresses 0 to FFFFFFFF hexadecimal.
b. The flat segmentation model is appropriate for protected mode programming.

2.1 Flat segmentation

The flat segmentation model is appropriate for protected mode programming. A typical protected-mode program has three segments: code, data, and stack, using the CS, DS, and SS.

3. Basic Instructions Set:

Instruction	Explanation	Syntax	Example
MOV	*Dest*←Operandor data stored at operand address	`MOV`Dest, Src	`MOV`AX,BX

MOV instruction is used to transfer data between registers, between a register and a memory location, or to move a number directly into a register or a memory location.

	Destination Operand			
Source Operand	**General Register**	**Segment Register**	**Memory location**	**Constant**
General Register	Yes	Yes	Yes	No
Segment Register	Yes	No	Yes	No
Memory location	Yes	Yes	No	No
Constant	Yes	No	Yes	No

Table 3.2: Legal Combination of Operands for MOV

- **XCHG**

Instruction	Explanation	Syntax	Example
XCHG	*Operand 1 ↔Operand 2* or data stored at operands address	XCHGDest, Src	XCHGAX,BX

This instruction swaps the contents of two operands, like in the above example data of AX and BX is being swapped.

	Destination Operand	
Source Operand	**General Register**	**Memory location**
General Register	Yes	Yes
Memory location	Yes	No

Table 3.3: Legal combination of Operands for XCHG

- **ADD and SUB**

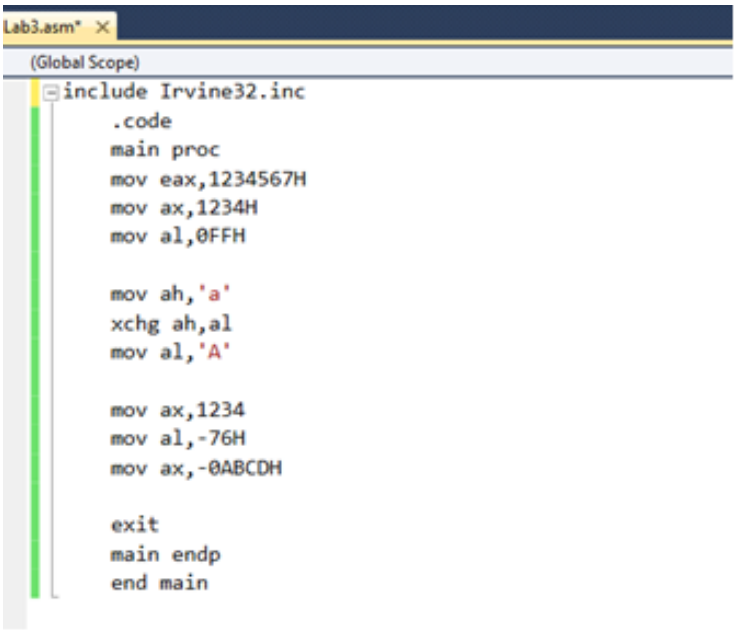
Instruction	Explanation	Syntax	Example
ADD , SUB	ADD:*Dest←Dest + source* SUB: *Dest←Dest–source*	ADDDest, source SUBDest, source	ADDAX, BX SUBVAR1, AX

The ADD and SUB instructions are used to add or subtract the contents of two registers, a register and a memory location, or to add (or subtract) a number to (or from) a register or memory location.

	Destination Operand	
Source Operand	**General Register**	**Memory location**
General Register	Yes	Yes
Memory location	Yes	No
Constant	Yes	Yes

Table 3.4: Legal combination of Operands for ADD,SUB

Example 3.1: To check the effect of basic instruction set on different registers. **Estimated completion time:20 mins**



```
Lab3.asm*
(Global Scope)
include Irvine32.inc
    .code
    main proc
    mov eax,1234567H
    mov ax,1234H
    mov al,0FFH

    mov ah,'a'
    xchg ah,al
    mov al,'A'

    mov ax,1234
    mov al,-76H
    mov ax,-0ABCDH

    exit
    main endp
    end main
```

After execution the above program step by step fill the following table (fill only required cell):

REGISTERS	**EAX**				**EIP**	**Linear Address (CS:EIP)**
	MSB 16 bits		**AX**			
INSTRUCTIONS	**U 8 bits**	**L 8 bits**	**AH**	**AL**		
MOV EAX,1234567H	01	23	45	67	00B71010	
MOV AX, 1234H	01	23	12	34		
MOV AL, 0FFH	01	23	12	FF		
MOV AH, 'a'	01	23	61	FF		
XCHG AH, AL	01	23	FF	61		
MOV AL, 'A'	01	23	FF	41		
MOV AX, 1234	01	23	04	D2		
MOV AL, -76H	01	23	04	8A		
MOV AX, -0ABCDH	01	23	54	33		

Problem(s) / Assignment(s)

Discussion & Practice **Estimated completion time:50 mins**

Problem 3.1: Write a program that adds two numbers stored in registers AX and BH as shown below, **Estimated completion time:15 mins**

AX= your roll no. , BH= FBH.

Store the result in EDX and swap the higher and lower bytes of result stored in DX.