K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 1

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To To Add two 16 bit numbers, to multiply two 8 bit & to divide two 8 bit numbers

Aim: To expose the students to the basic concepts of Microprocessor

Expected Outcome of Experiment:

CO 1: Explain the process of Compilation from Assembly language to machine language

Books/ Journals/ Websites referred:

1) Microprocessor architecture and applications with 8085: By Ramesh Gaonkar (Penram International Publication).

2) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).

Pre Lab/ Prior Concepts:

Assembler directives: These are statements that direct the assembler to do something

Definition:

Types of Assembler Directives:

ASSUME Directive - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

Example:

ASSUME CS: CODE; This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

ASUME DS: DATA ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA

Start:
It is entry point of the program. Without this program won't run.

END - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

ENDS - This ENDS directive is used with name ofthe segment to indicate the end of that logic segment.

Example:

CODE SEGMENT;
Hear it Start the logic
;segment containing code
; Some instructions statements to perform the logical
;operation
CODE ENDS ; End of segment named as; CODE

Arithmetic instruction set:
ADD instruction:

Syntax: ADD destination, source

Mnemonic	**Meaning**	**Format**	**Operation**	**Flags Affected**
ADD	Addition	ADD D, S	(S) + (D)→(D) Carry→(CF)	All
ADC	Add with carry	ADC D, S	(S) + (D) +(CF) → (D) Carry→(CF)	All

SUB instruction:

Mnemonic	**Meaning**	**Format**	**Operation**	**Flags Affected**
SUB	Subtract	SUB D, S	(D) - (S)→(D) Borrow→(CF)	All
SBB	Subtract with borrow	SBB D, S	(D) - (S) –(CF)→(D)	All

MUL instruction:
Syntax: MUL source

Multiplication (MUL or IMUL)	**Multiplicand**	**Operand (Multiplier)**	**Result**
Byte * Byte	AL	Register or Memory	AX
Word * Word	AX	Register or memory	DX :AX

DIV instruction:

Division (DIV or IDIV)	**Dividend**	**Operand (Divisor)**	**Quotient : Remainder**
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX

The steps to execute a program in TASM are

ASSEMBLING AND EXECUTING THE ROGRAM

1) Writing an Assembly Language Program

Assembly level programs generally abbreviated as ALP are written in text editor EDIT.

Type *EDIT* in front of the command prompt **(C:\TASM\BIN)** to open an untitled text file.

EDIT<file name>

After typing the program save the file with appropriate file name with an extension *.ASM*

Ex: Add.ASM

2) Assembling an Assembly Language Program

To assumble an ALP we needed executable file called MASM.EXE. Only if this file is in current working directory we can assemble the program. The command is

TASM<filename.ASM>

If the program is free from all syntactical errors, this command will give the **OBJEC**T file. In case of errors it list out the number of errors, warnings and kind of error.

Note: No object file is created until all errors are rectified.

3) Linking

After successful assembling of the program we have to link it to get **Executable file.**

The command is

TLINK<File name.OBJ>

This command results in <*Filename.exe*>which can be executed in front of the command prompt.

4) Executing the Program

Open the program in debugger by the command(note only exe files can be open)by the command.

<Filename.exe>

This will open the program in debugger screen where in you can view the assemble code with the CS and IP values at the left most side and the machine code. Register content,memory content also be viewed using ***TD***option of the debugger & to execute the program in single steps(F7)

Algorithm for adding the two numbers:

```
DATA SEGMENT
A DW 08H
B DW 03H
C  DW ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE ,DS:DATA
START:
MOV AX,DATA
MOV DS,AX

MOV AX,A
MOV BX,B
ADD AX,BX
MOV C,AX

MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Algorithm for Subtracting the two 16 bits numbers

```
DATA SEGMENT
A DW 08H
B DW 03H
C  DW ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE ,DS:DATA
START:
MOV AX,DATA
MOV DS,AX

MOV AX,A
MOV BX,B
```

```
SUB AX,BX
MOV C,AX

MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Algorithm for multiplying the two numbers:

```
DATA SEGMENT
A DB 08H
B DB 03H
C  DB ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE ,DS:DATA
START:
MOV AX,DATA
MOV DS,AX

MOV AL,A
MOV BL,B
MUL BL
MOV C,AL

MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Algorithm for dividing the two numbers:

```
DATA SEGMENT
A DB 08H
B DB 03H
C  DB ?
D DB ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE ,DS:DATA
START:
MOV AX,DATA
MOV DS,AX

MOV AL,A
MOV BL,B
MOV AH, 00H
DIV BL
MOV C,AL
MOV D, AH

MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Conclusion: The programs for adding, subtracting, multiplying and dividing two numbers were successfully written and executed in Assembly.

Post Lab Descriptive Questions (Add questions from examination point view)

Explain instructions ADC and SBB with example

ADC: Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a carry from a previous addition. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The ADC instruction is usually executed as part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

```
AL = 01 and BL = 02, and CF = 1
ADC AL,BL
```

SBB: Adds the source operand (second operand) and the carry (CF) flag, and subtracts the result from the destination operand (first operand). The result of the subtraction is stored in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a borrow from a previous subtraction.When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The SBB instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a borrow in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.The SBB instruction is usually executed as part of a multibyte or multiword subtraction in which a SUB instruction is followed by a SBB instruction. This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

```
AL = 05 and BL = 02, and CF = 1
SBB AL,BL
```

Date: 26/01/2019 **Signature of faculty in-charge**