Batch: B2	**Roll No.:** 110
Experiment / assignment / tutorial No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implement simple addition, subtraction, multiplication and division instructions using TASM.

AIM: Implement simple addition, subtraction, multiplication and division instructions using TASM.

Expected OUTCOME of Experiment: (Mentions the CO/CO's attained)

Understand the Central processing unit with addressing modes and working of control unit in depth.

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:

ASUME 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:

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

Syntax: ADD destination,source

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>

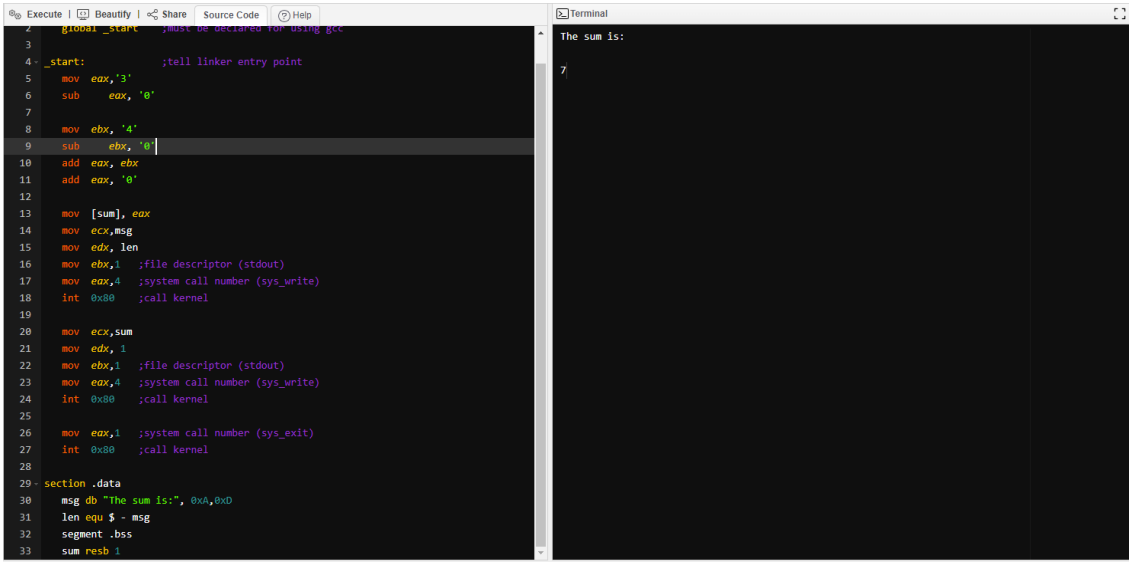
This command results in <*Filename.exe*>which can be executed in front of the command prompt.

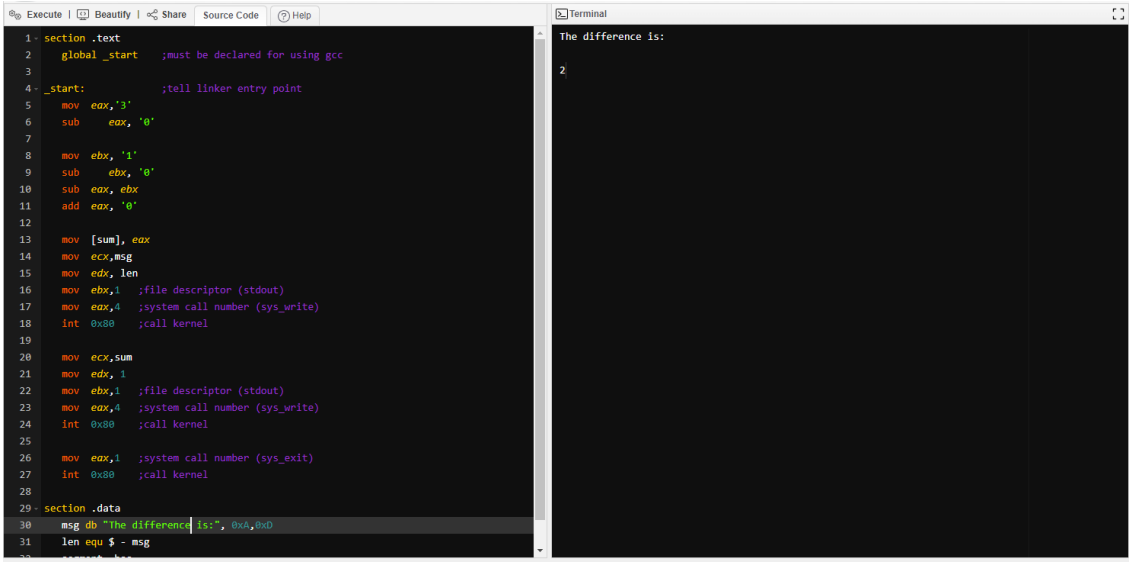
4) Executing the Program

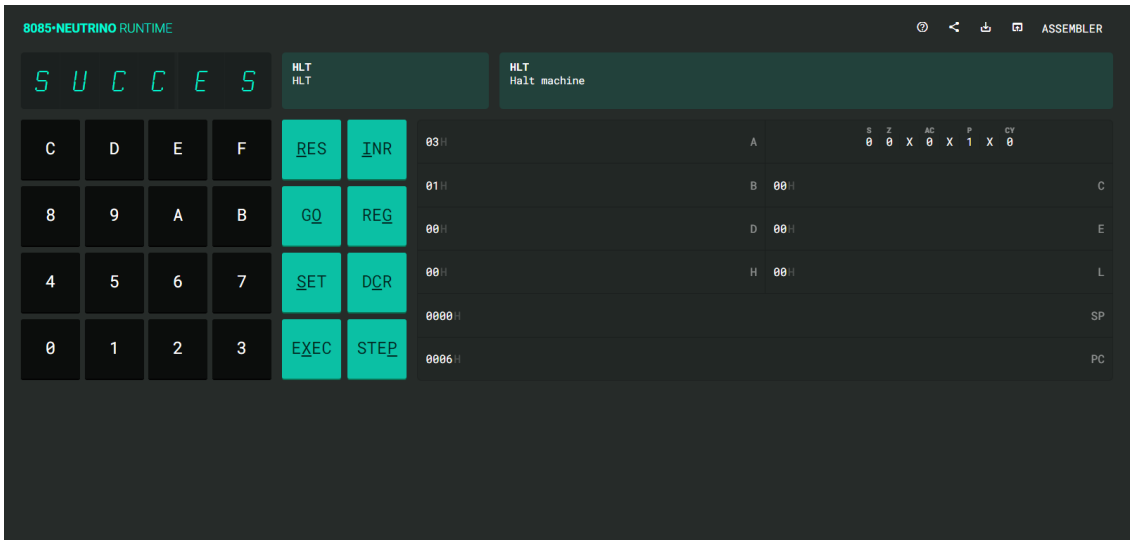
Open the program in debugger by the command(note only exe files can be open)by the command.

<*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)


Execute | Beautify | Share Source Code Help
global _start ;must be declared for using gcc
_start: ;tell linker entry point
mov eax,'3'
sub eax, '0'
mov ebx, '4'
sub ebx, '0'
add eax, ebx
add eax, '0'
mov [sum], eax
mov ecx,msg
mov edx, len
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov ecx,sum
mov edx, 1
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov eax,1 ;system call number (sys_exit)
int 0x80 ;call kernel
section .data
msg db "The sum is:", 0xA,0xD
len equ $ - msg
segment .bss
sum resb 1
Terminal
The sum is:
7


Execute | Beautify | Share Source Code Help
section .text
global _start ;must be declared for using gcc
_start: ;tell linker entry point
mov eax,'3'
sub eax, '0'
mov ebx, '1'
sub ebx, '0'
sub eax, ebx
add eax, '0'
mov [sum], eax
mov ecx,msg
mov edx, len
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov ecx,sum
mov edx, 1
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov eax,1 ;system call number (sys_exit)
int 0x80 ;call kernel
section .data
msg db "The difference is:", 0xA,0xD
len equ $ - msg
Terminal
The difference is:
2



Algorithm for adding the two 8-bit numbers:

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

Algorithm for subtracting the two 8 bit numbers:

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Sub the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

Algorithm for Subtracting

1) Take twos complement
2) add the numbers
3) if first bit is 1 take twos complement again

Algorithm for multiplying the two 8 bit numbers:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry
in memory location.
8) Terminate the program.

Algorithm for dividing the two 8-bit numbers:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register(B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry .
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

References - **https://www.mpuat.ac.in/images/editorFiles/file/E-Content/Microprocessor%20Programming%20Lab%20Manual.pdf**

Conclusion:

Thus we have understood the working of assembly language. We performed addition and subtraction of two 8 bit numbers using assembly language and understood the constraints and logic of the working.

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

Explain instructions ADC and SBB with example

ADC is add with carry and SBB is subtract with borrow.

ADC adds two numbers and adds carry if present as denoted by the carry flag.

SBB subtracts the two numbers with borrow.

Example ADC A,B or SBB A,C