Batch: A3 **Roll No.:** 16010121051

Experiment / assignment / tutorial No. 9

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:

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:

(DIV or IDIV)		**(Divisor)**	
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 8-bit numbers:

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can't take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then add both of them using ADD AL, BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type "code ends" to end the execution of the code.

Algorithm for subtracting the two 8 bit numbers:

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can't take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then subtract both of them using SUB AL, BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type "code ends" to end the execution of the code.

Algorithm for multiplying the two 8 bit numbers:

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can't take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then multiply both of them using MUL BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type "code ends" to end the execution of the code.

Algorithm for dividing the two 8-bit numbers:

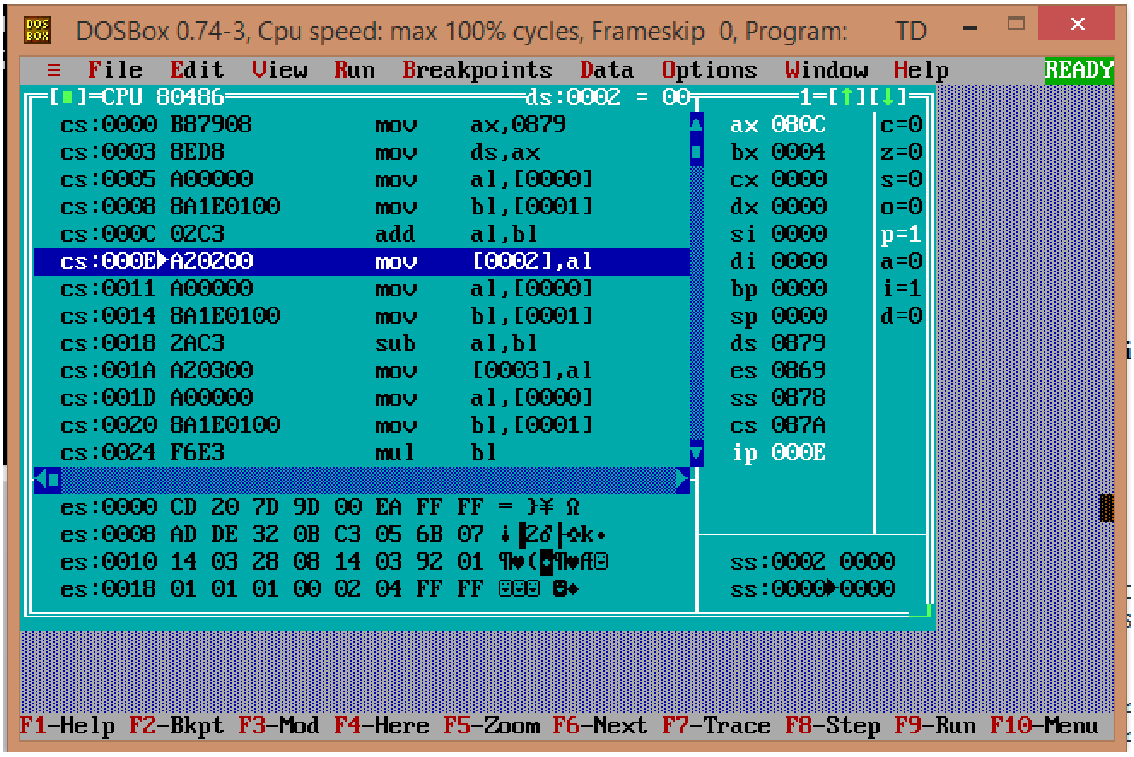
1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can't take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then divide both of them using DIV BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type "code ends" to end the execution of the code.

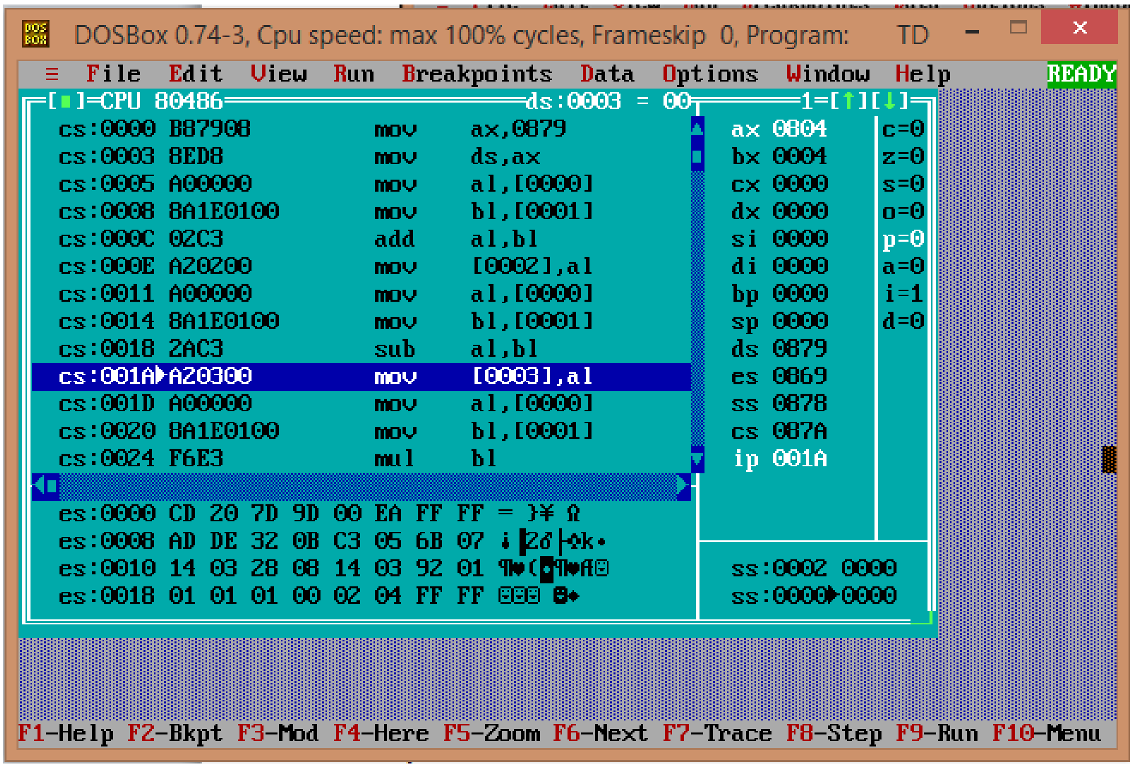
Code:

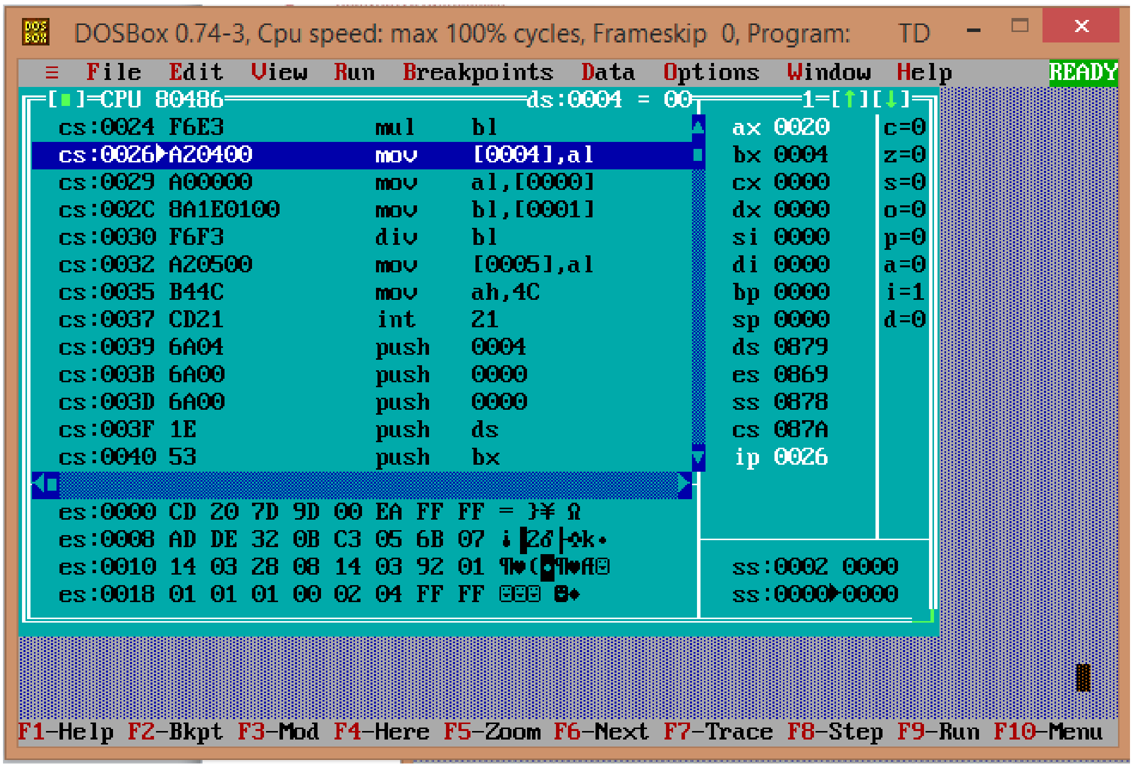
```
data segment
a db 08h
b db 04h
```

```
c db ?
d db ?
e db ?
f db ?
data ends
code segment
assume cs:code, ds:data
start:
MOV AX, data
MOV DS, AX
MOV AL, a
MOV BL, b
ADD AL, BL
MOV c, AL
MOV AL, a
MOV BL, b
SUB AL, BL
MOV d, AL
MOV AL, a
MOV BL, b
MUL BL
MOV e, AL
MOV AL, a
MOV BL, b
DIV BL
MOV f, AL
MOV ah, 4ch
int 21h
code ends
end start
end
```

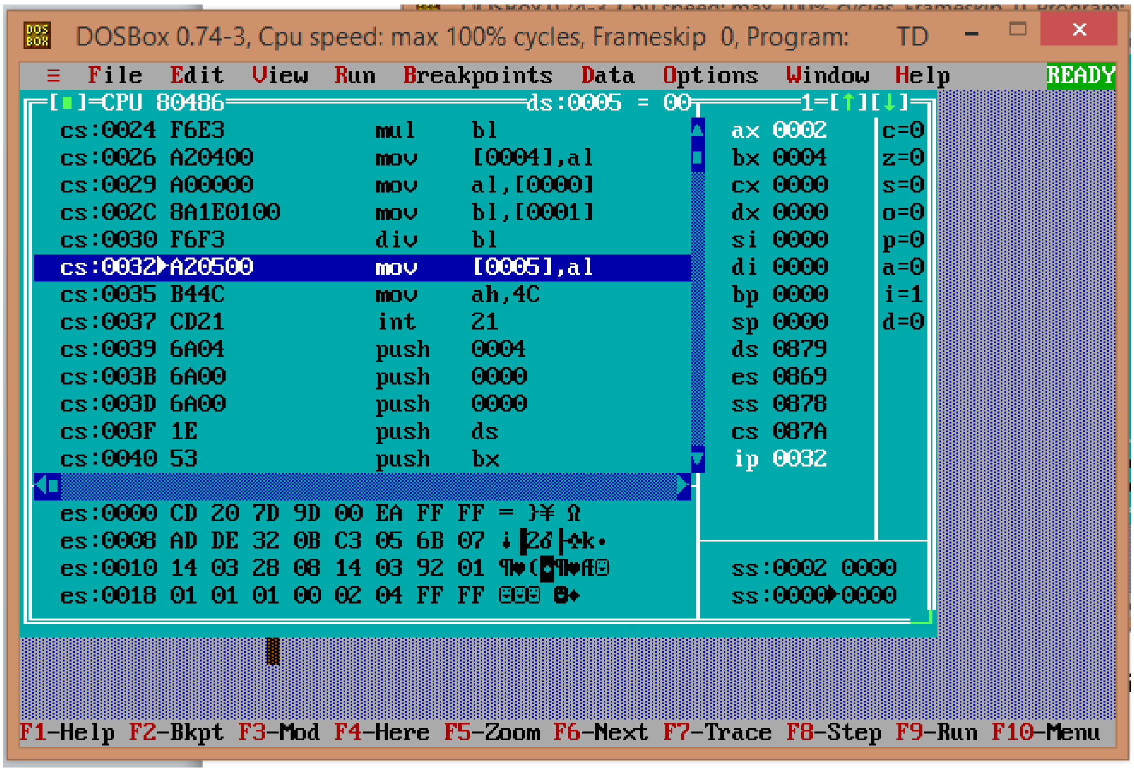
Output:


DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TD
≡ File Edit View Run Breakpoints Data Options Window Help READY
[■]=CPU 80486 ds:0002 = 00 1=[↑][↓]
cs:0000 B87908 mov ax,0879
cs:0003 8ED8 mov ds,ax
cs:0005 A00000 mov al,[0000]
cs:0008 8A1E0100 mov bl,[0001]
cs:000C 02C3 add al,bl
cs:000E A20200 mov [0002],al
cs:0011 A00000 mov al,[0000]
cs:0014 8A1E0100 mov bl,[0001]
cs:0018 2AC3 sub al,bl
cs:001A A20300 mov [0003],al
cs:001D A00000 mov al,[0000]
cs:0020 8A1E0100 mov bl,[0001]
cs:0024 F6E3 mul bl
ax 080C
bx 0004
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 0879
es 0869
ss 0878
cs 087A
ip 000E
c=0
z=0
s=0
o=0
p=1
a=0
i=1
d=0
es:0000 CD 20 7D 9D 00 EA FF FF
es:0008 AD DE 32 0B C3 05 6B 07
es:0010 14 03 28 08 14 03 92 01
es:0018 01 01 01 00 02 04 FF FF
ss:0002 0000
ss:0000 0000
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu


DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TD
≡ File Edit View Run Breakpoints Data Options Window Help READY
[■]=CPU 80486 ds:0003 = 00 1=[↑][↓]
cs:0000 B87908 mov ax,0879
cs:0003 8ED8 mov ds,ax
cs:0005 A00000 mov al,[0000]
cs:0008 8A1E0100 mov bl,[0001]
cs:000C 02C3 add al,bl
cs:000E A20200 mov [0002],al
cs:0011 A00000 mov al,[0000]
cs:0014 8A1E0100 mov bl,[0001]
cs:0018 2AC3 sub al,bl
cs:001A A20300 mov [0003],al
cs:001D A00000 mov al,[0000]
cs:0020 8A1E0100 mov bl,[0001]
cs:0024 F6E3 mul bl
ax 0804
bx 0004
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 0879
es 0869
ss 0878
cs 087A
ip 001A
c=0
z=0
s=0
o=0
p=0
a=0
i=1
d=0
es:0000 CD 20 7D 9D 00 EA FF FF
es:0008 AD DE 32 0B C3 05 6B 07
es:0010 14 03 28 08 14 03 92 01
es:0018 01 01 01 00 02 04 FF FF
ss:0002 0000
ss:0000 0000
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu



1.



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

Explain instructions ADC and SBB with example

Answer:

Adding and Subtracting in Multiple Registers:

Numbers larger than the register size on your processor can be added and subtracted with the ADC (Add with Carry) and SBB (Subtract with Borrow) instructions.

```
These instructions work as follows:
ADC Dest, Source      ; Dest = Dest + Source + Carry Flag
SBB Dest, Source      ; Dest = Dest - Source - Carry Flag
```

If the operations prior to an ADC or SBB instruction do not set the carry flag, these instructions are identical to ADD and SUB. While operating on large values in more than one register, ADD and SUB are used for the least significant part of the number and ADC or SBB for the most significant part.

Use of ADC and SBB Instructionson the 8086 Processor

```
        .DATA
mem32   DWORD   316423
mem32a  DWORD   316423
mem32b  DWORD   156739
        .CODE
        .
        .
        .
; Addition
        mov     ax, 43981                ; Load immediate     43981
        sub     dx, dx                   ;   into DX:AX
        add     ax, WORD PTR mem32[0]    ; Add to both      + 316423
        adc     dx, WORD PTR mem32[2]    ;   memory words     ------
                                         ; Result in DX:AX    360404

; Subtraction
        mov     ax, WORD PTR mem32a[0]   ; Load mem32         316423
        mov     dx, WORD PTR mem32a[2]   ;   into DX:AX
        sub     ax, WORD PTR mem32b[0]   ; Subtract low     - 156739
        sbb     dx, WORD PTR mem32b[2]   ;   then high        ------
                                         ; Result in DX:AX    159684
```

Date: 21/11/2022 **Signature of faculty in-charge**