**Lab # 08**

**Procedures in x86 Assembly Language**

The 80x86 architecture enables implementation of procedures that are similar to those in high-level language. These procedures can be called from high-level language program or can call high-level language procedures. There are three main concepts involved: (1) transfer of control from calling program (procedure) to the called procedure and back, (2) passing parameters from calling program (procedure) to the called procedure and results back to the calling program, and

(3) development of procedure code that is independent of the calling program. A procedure is a subprogram that is essentially a self-contained unit. Main program or another subprogram calls a procedure. A procedure may simply do a task or it may return a value. Value-returning procedure is sometimes called a function.

Procedures are valuable in assembly language for the same reasons as in a HLL. However, sometimes assembly language can be used to write more efficient code than is produced by a HLL compiler and this code can be put in a procedure called by a HLL program that does tasks that don't need to be as efficient.

**Procedure Definition** In a code segment following *.CODE* directive, the procedure body is bracketed by PROC and ENDP directives giving procedure name as the label.

.CODE procName PROC

procedure body

procName ENDP

➢

# Call Instruction

Transferring Control to a Procedure In the “main” program or calling procedure, control is transferred to the called procedure using CALL instruction. Call causes the procedure named in the operand to be executed. When the called procedure completes, execution flow resumes at the instruction following the call instruction . return address is (IP) is pushed to stack. **Syntax :**

# *Call procedure\_name*

The next instruction executed will be the first one in the procedure.

➢

# Return from Procedure (ret)

The ret instruction transfers control to the return address located on the stack. This address is usually placed on the stack by a call instruction. Issue the ret instruction within the called procedure to resume execution flow at the instruction following the call. Return address (IP) is removed from the stack. **Syntax:**

## RET Or RET imm

**Example 1:**

.MODEL small

.Stack 100h

.CODE ; start of main program code main proc mov cl, 3 ; move data into registers mov bl, 2

***call add2nums;*** push address of next instruction onto the stack, and pass control mov ah,4ch ; exit code Output int 21h



5

main ENDp ; end of source code

***add2nums proc*** add cl, bl ; Add two numbers mov dl,cl add dl,30h mov ah,2h int 21h

***ret*** ; Pop the address from stack ***add2nums endp***

end main

# Calling a Procedure within a Procedure

.data

**Msg** db ? ,"$"

.code

Main proc

Mov ax,@data

Mov ds,ax

; Calling Three Procedures

**Call input ;PUSH** address of next instruction onto the stack, and pass control **Call save**

Mov ah,4ch

Int 21h

Main endp

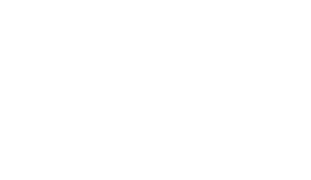
# Input proc Mov ah,1

Int 21h

|  |  |
| --- | --- |
| **ret**  **Input endp**  **Save proc**  Mov msg,al | **; POP the address from the stack** |
| **Call Display ret**  **Save endp**    **Display proc** | **;Call “Display” with in “Save”** |
| Mov dl,0ah | **Output:** |

Mov ah,2 Int 21h

Lea dx,msg



Mov ah,9

Int 21h

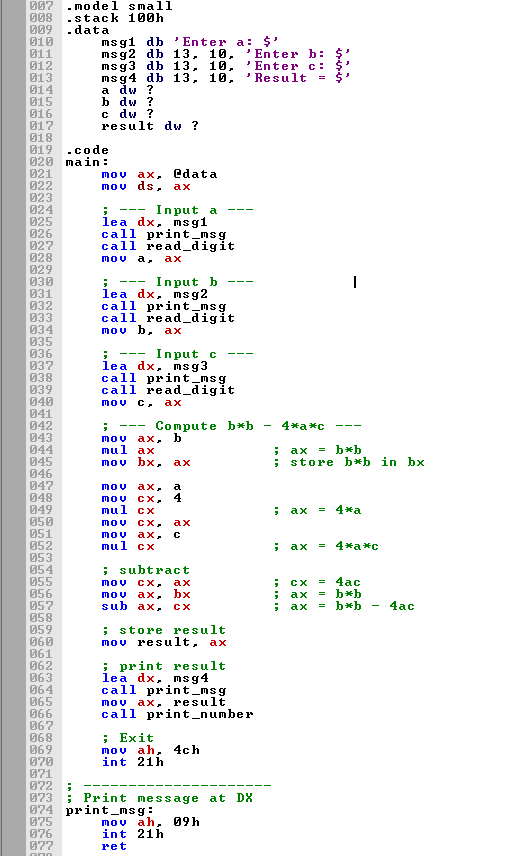
**Ret**

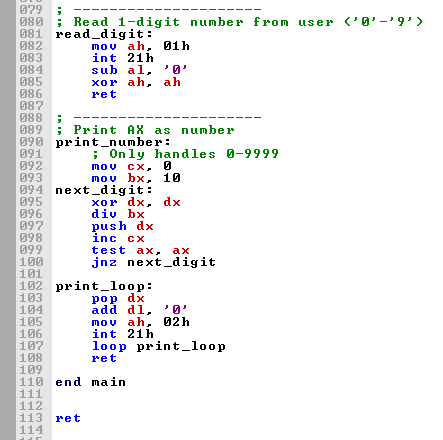
# Display Endp

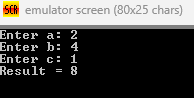
End main

**EXERCISE:**

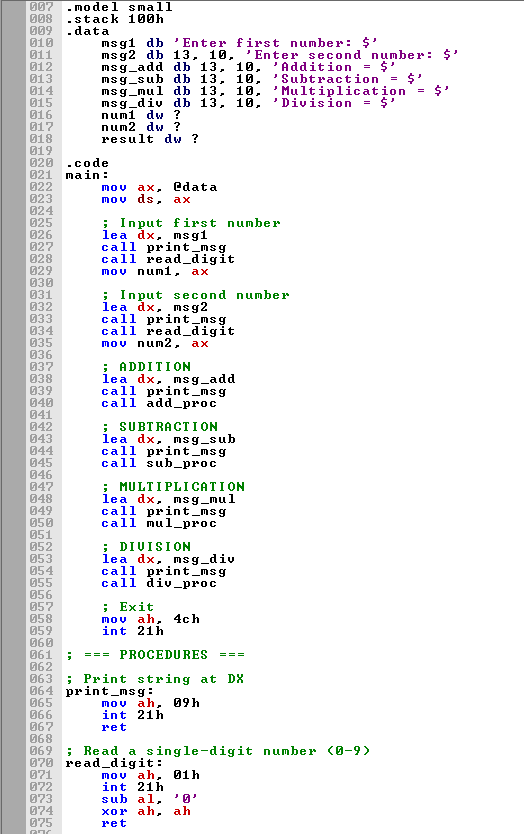
Q1. Write assembly code of a procedure that computes the value of *b\*b-4\*a\*c ,*where a,b,c are user provided inputs

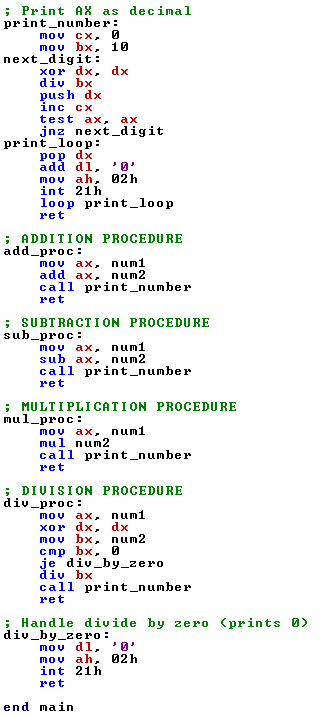


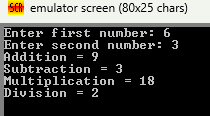




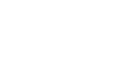
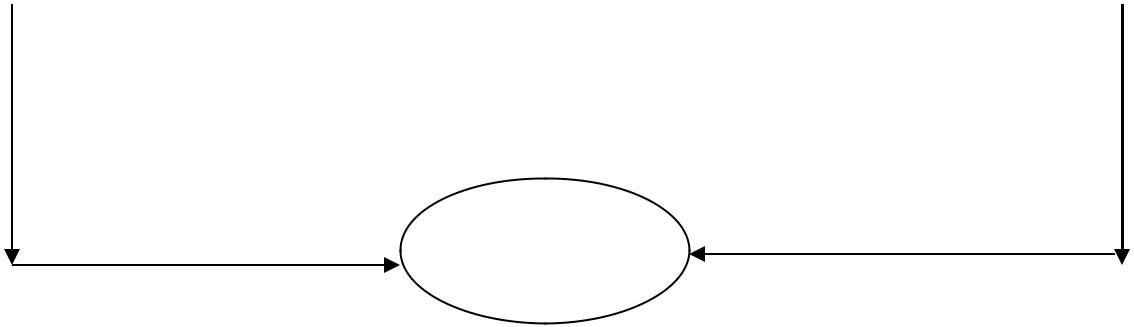
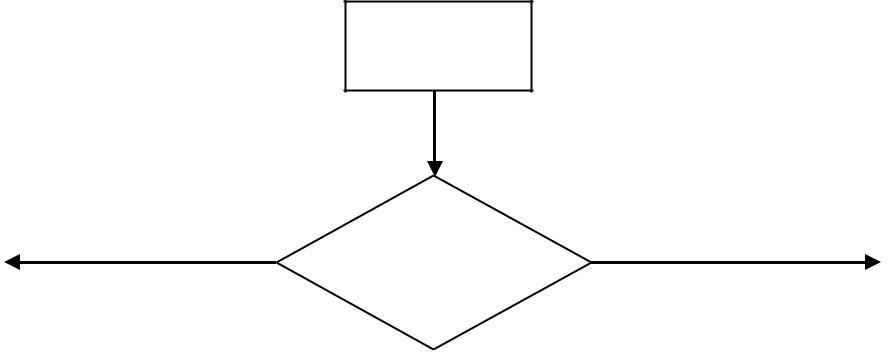
Q2. Write code of a program that perform 4 basic arithmetic operations on user provided input . code should have at least four procedures.



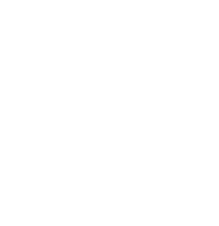
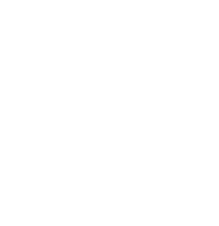




Q 3 Write down the code of following scenario. Program should have 2 procedures .



Input==1



END

