NAME: GINI CHACKO

SEMESTER: IV

CLASS: SE COMPS B

BATCH: B

ROLL: 8942

TOPIC: MP EXPERIMENT 7:

A.] WALP to calculate factorial of a given number using macro

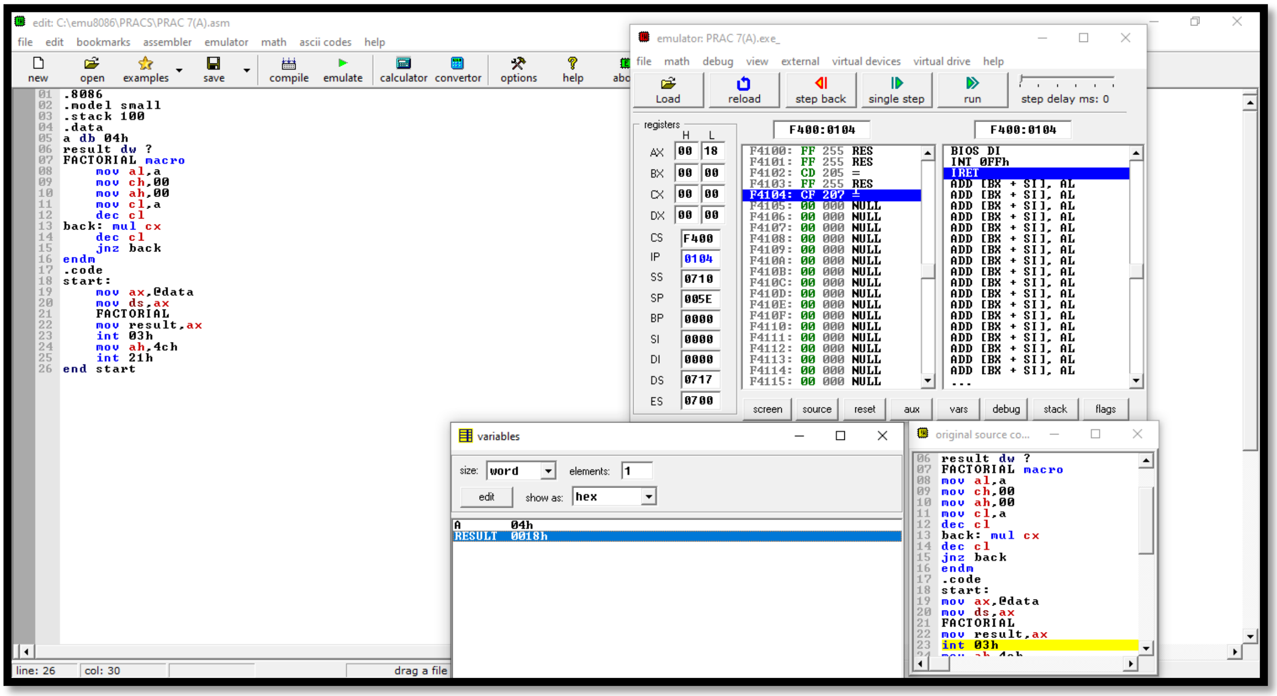
B.] WALP to find area of rectangle and area of square using Procedure (use call and ret instructions)

A.] WALP to calculate factorial of a given number using macro

CODE:

```
.8086
.model small
.stack 100
.data
a db 04h
result dw ?
FACTORIAL macro
    mov al,a
    mov ch,00
    mov ah,00
    mov cl,a
    dec cl
back: mul cx
    dec cl
    jnz back
endm
.code
start:
    mov ax,@data
    mov ds,ax
    FACTORIAL
    mov result,ax
    int 03h
    mov ah,4ch
    int 21h
end start
```

OUTPUT:


edit: C:\emu8086\PRACS\PRAC 7(A).asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help abo
emulator: PRAC 7(A).exe_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
variables
size: word elements: 1
edit show as: hex
original source co...
line: 26 col: 30 drag a file

```
.8086
.model small
.stack 100
.data
a db 04h
result dw ?
FACTORIAL macro
    mov al,a
    mov ch,00
    mov ah,00
    mov cl,a
    dec cl
back: mul cx
    dec cl
    jnz back
endm
.code
start:
    mov ax,@data
    mov ds,ax
    FACTORIAL
    mov result,ax
    int 03h
    mov ah,4ch
    int 21h
end start
```

registers

	H	L
AX	00	18
BX	00	00
CX	00	00
DX	00	00
CS	F400	
IP	0104	
SS	0710	
SP	005E	
BP	0000	
SI	0000	
DI	0000	
DS	0717	
ES	0700	

```
F400:0104
F4100: FF 255 RES
F4101: FF 255 RES
F4102: CD 205 =
F4103: FF 255 RES
F4104: CF 207 ±
F4105: 00 000 NULL
F4106: 00 000 NULL
F4107: 00 000 NULL
F4108: 00 000 NULL
F4109: 00 000 NULL
F410A: 00 000 NULL
F410B: 00 000 NULL
F410C: 00 000 NULL
F410D: 00 000 NULL
F410E: 00 000 NULL
F410F: 00 000 NULL
F4110: 00 000 NULL
F4111: 00 000 NULL
F4112: 00 000 NULL
F4113: 00 000 NULL
F4114: 00 000 NULL
F4115: 00 000 NULL
```

```
F400:0104
BIOS DI
INT 0FFh
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

```
A        04h
RESULT   0018h
```

```
result dw ?
FACTORIAL macro
mov al,a
mov ch,00
mov ah,00
mov cl,a
dec cl
back: mul cx
dec cl
jnz back
endm
.code
start:
mov ax,@data
mov ds,ax
FACTORIAL
mov result,ax
int 03h
```

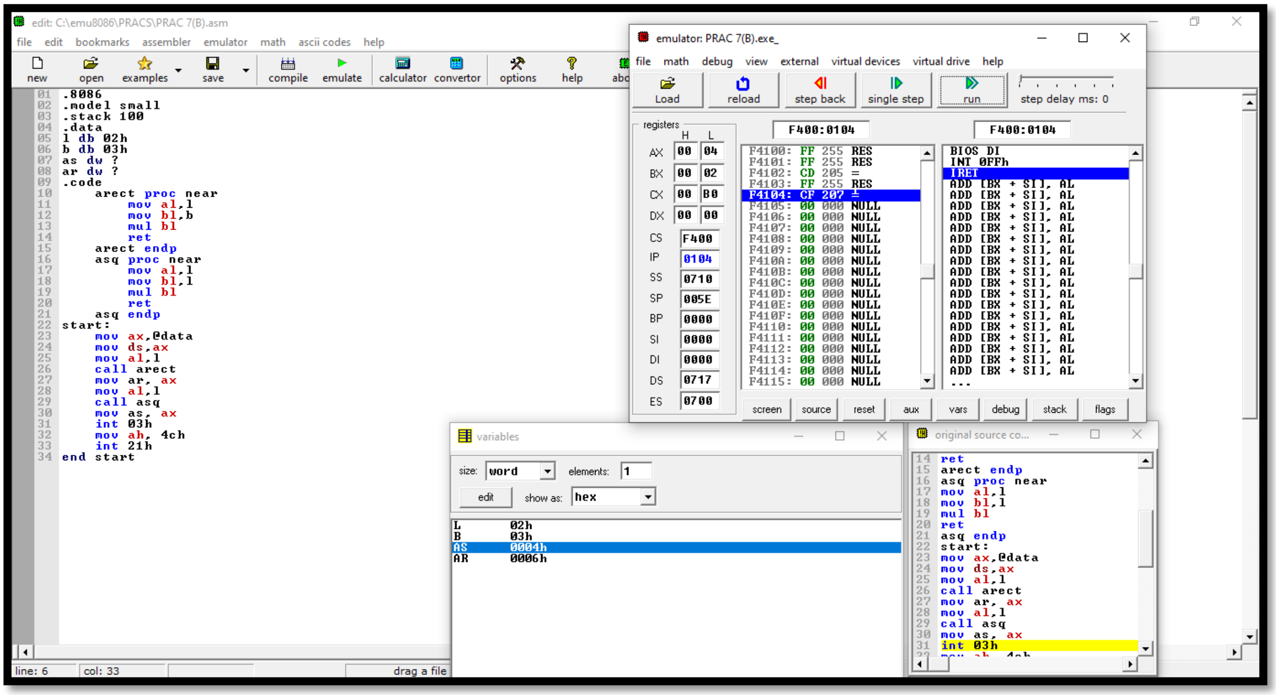
B.] WALP to find area of rectangle and area of square using Procedure (use call and ret instructions)

CODE:

```
.8086

.model small

.stack 100

.data

l db 02h

b db 03h

as dw ?

ar dw ?


.code

   arect proc near

      mov al,l

      mov bl,b

      mul bl

      ret

   arect endp

   asq proc near

      mov al,l
```

```
        mov bl,l
        mul bl
        ret
    asq endp

start:
    mov ax,@data
    mov ds,ax
    mov al,l
    call arect
    mov ar, ax
    mov al,l
    call asq
    mov as, ax
    int 03h
    mov ah, 4ch
    int 21h
end start
```

OUTPUT:



edit: C:\emu8086\PRACS\PRAC 7(B).asm

```
.8086
.model small
.stack 100
.data
l db 02h
b db 03h
as dw ?
ar dw ?
.code
    arect proc near
        mov al,l
        mov bl,b
        mul bl
        ret
    arect endp
    asq proc near
        mov al,l
        mov bl,l
        mul bl
        ret
    asq endp
start:
    mov ax,@data
    mov ds,ax
    mov al,l
    call arect
    mov ar, ax
    mov al,l
    call asq
    mov as, ax
    int 03h
    mov ah, 4ch
    int 21h
end start
```

line: 6   col: 33

emulator: PRAC 7(B).exe_

registers

```
      H    L
AX    00   04
BX    00   02
CX    00   B0
DX    00   00
CS    F400
IP    0104
SS    0710
SP    005E
BP    0000
SI    0000
DI    0000
DS    0717
ES    0700
```

```
F400:0104

F4100: FF 255 RES
F4101: FF 255 RES
F4102: CD 205 =
F4103: FF 255 RES
F4104: CF 207 ±
F4105: 00 000 NULL
F4106: 00 000 NULL
F4107: 00 000 NULL
F4108: 00 000 NULL
F4109: 00 000 NULL
F410A: 00 000 NULL
F410B: 00 000 NULL
F410C: 00 000 NULL
F410D: 00 000 NULL
F410E: 00 000 NULL
F410F: 00 000 NULL
F4110: 00 000 NULL
F4111: 00 000 NULL
F4112: 00 000 NULL
F4113: 00 000 NULL
F4114: 00 000 NULL
F4115: 00 000 NULL
```

```
F400:0104

BIOS DI
INT 0FFh
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

variables

size: word   elements: 1   show as: hex

```
L       02h
B       03h
AS      0004h
AR      0006h
```

original source co...

```
ret
arect endp
asq proc near
mov al,l
mov bl,l
mul bl
ret
asq endp
start:
mov ax,@data
mov ds,ax
mov al,l
call arect
mov ar, ax
mov al,l
call asq
mov as, ax
int 03h
```

POSTLAB QUESTIONS:

1. **Write difference between procedure and Macro**

Ans:

S.No.	MACRO	PROCEDURE
01.	Macro definition contains a set of instruction to support modular programming.	Procedure contains a set of instructions which can be called repetitively which can perform a specific task.
02.	It is used for small set of instructions mostly less than ten instructions.	It is used for large set of instructions mostly more than ten instructions.
03.	In case of macro memory requirement is high.	In case of procedure memory requirement is less.
04.	CALL and RET instruction/statements are not required in macro.	CALL and RET instruction/statements are required in procedure.
05.	Assembler directive MACRO is used to define macro and assembler directive ENDM is used to indicate the body is over.	Assembler directive PROC is used to define procedure and assembler directive ENDP is used to indicate the body is over.
06.	Execution time of macro is less than it executes faster than procedure.	Execution time of procedures is high as it executes slower than macro.
07.	Here machine code is created multiple times as each time machine code is generated when macro is called.	Here machine code is created only once, it is generated only once when the procedure is defined.
08.	In a macro parameter is passed as part of statement that calls macro.	In a procedure parameters are passed in registers and memory locations of stack.
09.	Overhead time does not take place as there is no calling and returning.	Overhead time takes place during calling procedure and returning control to calling program.

2. Write processor control transfer instructions

Ans:

The 80386 provides both conditional and unconditional control transfer instructions to direct the flow of execution. Conditional control transfers depend on the results of operations that affect the flag register. Unconditional control transfers are always executed.

- **Unconditional Transfer Instructions**

JMP, CALL, RET, INT and IRET instructions transfer control from one code segment location to another. These locations can be within the same code segment (near control transfers) or in different code segments (far control transfers). If the model of memory organization used in a particular 80386 application does not make segments visible to applications programmers, intersegment control transfers will not be used.

- **Conditional Transfer Instructions**

The conditional transfer instructions are jumps that may or may not transfer control, depending on the state of the CPU flags when the instruction executes.

Conditional jump instructions contain a displacement which is added to the EIP register if the condition is true. The displacement may be a byte, a word, or a double word. The displacement is signed; therefore, it can be used to jump forward or backward.

3. Write the role of stack in call and ret instruction

Ans:

The **CALL instruction** is used whenever we need to make a call to some procedure or a subprogram. Whenever a **CALL** is made, the following process takes place inside the microprocessor:

- The address of the next instruction that exists in the caller program (after the program CALL instruction) is stored in the stack.
- The instruction queue is emptied for accommodating the instructions of the procedure.
- Then, the contents of the instruction pointer (IP) is changed with the address of the first instruction of the procedure.
- The subsequent instructions of the procedure are stored in the instruction queue for execution.

The **RET instruction** stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program. Whenever the **RET instruction** is called, the following process takes place inside the microprocessor:

- The address of the next instruction in the mainline program which was previously stored inside the stack is now again fetched and is placed inside the instruction pointer (IP).
- The instruction queue will now again be filled with the subsequent instructions of the mainline program.

CALL instruction, RET instruction and the role of stack :
When the CALL instruction is executed, the address of the instruction below the CALL instruction is pushed onto the stack. When the execution of that subroutine is finished and RET is executed, the address of the instruction below the CALL instruction is loaded in the program counter and it is executed.