EXPERIMENT 7

8940 WARREN FERNANDES

OBJECTIVE: To understand implementation of procedure and macro, and control transfer instructions

Aim:

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**)**

Theory: Macros

Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. If you declared a macro and never used it in your code, compiler will simply ignore it.

```
Macro definition:

name    MACRO [parameters,...]

        <instructions>

ENDM
```

Unlike procedures, macros should be defined above the code that uses it, for example:

```
MyMacro    MACRO  p1, p2, p3
```

```
    MOV AX, p1
    MOV BX, p2
    MOV CX, p3

ENDM

ORG 100h

MyMacro 1, 2, 3

MyMacro 4, 5, DX

RET
```

The above code is expanded into:

```
MOV AX, 00001h

MOV BX, 00002h

MOV CX, 00003h

MOV AX, 00004h

MOV BX, 00005h

MOV CX, DX
```

Procedure:

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called.

The syntax for procedure declaration:

name PROC

```
; here goes the code

; of the procedure ...
```

RET
name ENDP

name - is the procedure name, the same name should be in the top and the bottom, this is used to check correct closing of procedures.

Probably, you already know that RET instruction is used to return to operating system. The same instruction is used to return from procedure (actually operating system sees your program as a special procedure).

PROC and ENDP are compiler directives, so they are not assembled into any real machine code. Compiler just remembers the address of procedure.

CALL instruction is used to call a procedure.

Here is an example:

```
ORG   100h

CALL  m1

MOV   AX, 2

RET                ; return to operating
system.

m1    PROC
MOV   BX, 5
RET                ; return to caller.
m1    ENDP

END
```

The above example calls procedure m1, does MOV BX, 5, and returns to the next instruction after CALL: MOV AX, 2.

There are several ways to pass parameters to procedure, the easiest way to pass parameters is by using registers, here is another example of a procedure that receives two parameters in AL and BL registers, multiplies these parameters and returns the result in AX register:

```
ORG   100h
```

```
MOV    AL, 1
MOV    BL, 2

CALL   m2
CALL   m2
CALL   m2
CALL   m2

RET                ; return to operating
system.

m2     PROC
MUL    BL          ; AX = AL * BL.
RET                ; return to caller.
m2     ENDP

END
```


In the above example value of AL register is update every time the procedure is called, BL register stays unchanged, so this algorithm calculates 2 in power of 4, so final result in AX register is 16 (or 10h).

Some important facts about macros and procedures:

- When you want to use a procedure you should use CALL instruction, for example:

CALL MyProc

- When you want to use a macro, you can just type its name. For example:

MyMacro

- Procedure is located at some specific address in memory, and if you use the same procedure 100 times, the CPU will transfer control to this part of the memory. The control will be returned back to the program by RET instruction. The stack is used to keep the return address. The CALL instruction takes about 3 bytes, so the size of the output executable file grows very insignificantly, no matter how many time the procedure is used.
- Macro is expanded directly in program's code. So if you use the same macro 100 times, the compiler expands the macro 100 times, making the output executable file larger and larger, each time all instructions of a macro are inserted.
- You should use stack or any general purpose registers to pass parameters to procedure.
- To pass parameters to macro, you can just type them after the macro name. For example:

MyMacro 1, 2, 3

- To mark the end of the macro ENDM directive is enough.
- To mark the end of the procedure, you should type the name of the procedure before the ENDP directive.

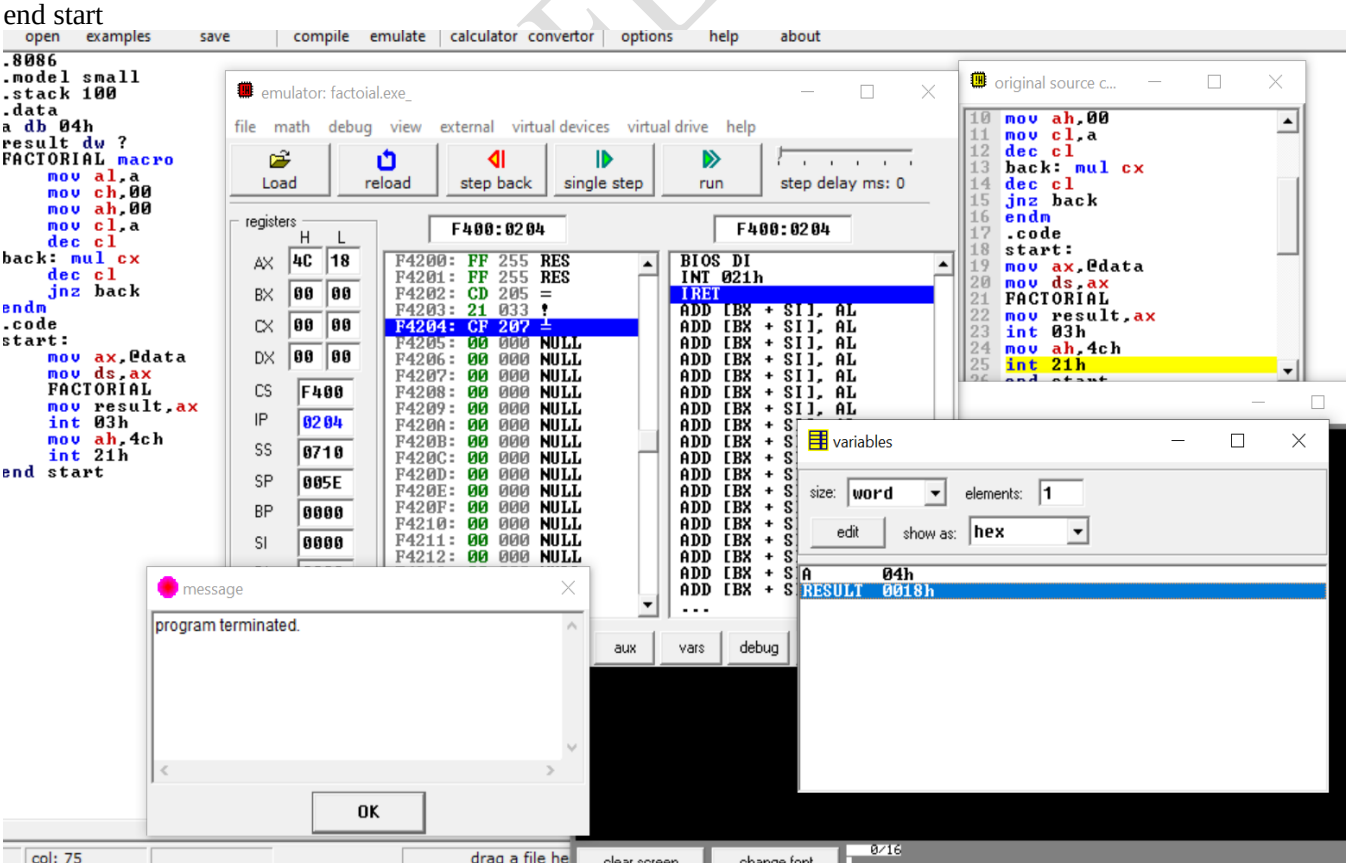
Algorithm:

1. Start.
2. Read a number from user.
3. Pass it to the macro.
4. Factorial macro:
   a) Store the number in any register.
   b) Find factorial by multiplying a number with a decremented value.
   c) Repeat step (b) till decremented value of number is not one.
   d) Store the factorial in any memory location.

5. Stop.

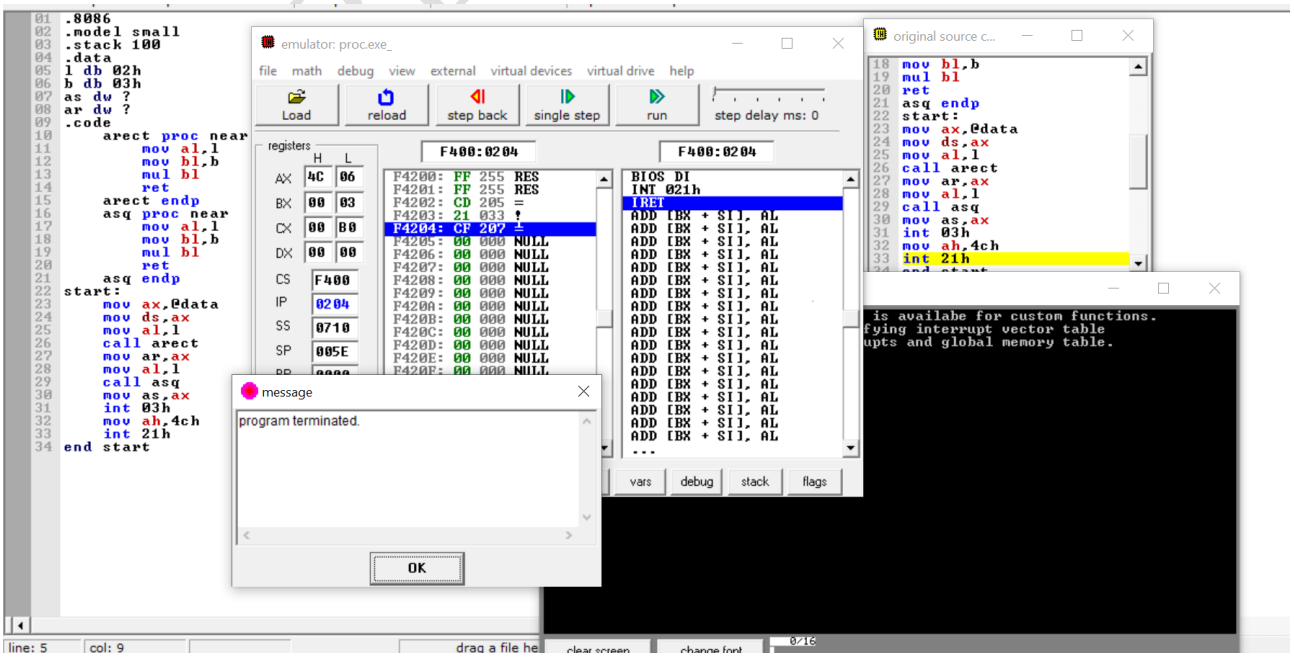
Program to find factorial

```
.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
```



Program to find Area

```
.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,b
      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
```



Post lab questions

Write difference between procedure and Macro

Write processor control transfer instructions

Write the role of stack in call and ret instructio

1)
A macro is a group of repetitive instructions in a program which are codified only once and can be used as many times as necessary.The main difference between a macro and a procedure is that in the macro the passage of parameters is possible and in the procedure it is not, this is only applicable for the MASM - there are other programming languages which do allow it. At the moment the macro is executed each parameter is substituted by the name or value specified at the time of the call. We can say then that a procedure is an extension of a determined program, while the macro is a module with specific unctions which can be used by different programs. Another difference between a macro and a procedure is the way of calling each one, to call a procedure the use of directive is required, on the other hand the call of macros is done as if it were an assembler instruction. Example of procedure: For example, if we want a routine which adds two bytes stored in AH and AL each one, and keep the addition in the BX register.

2)

- JMP – Unconditional

  1. Direct Jump

  The new branch location is specified directly in the instruction. The new address is calculated by adding the 8 or 16 bit displacement to the IP

  2. Indirect Jump

  The new branch address is specified indirectly through a register or a memory location. The value in the IP is replaced with the new value

- JCondition

  This is conditional branch instruction. If the condition is TRUE then it is similar to direct jump and if FALSe then branch does not take place and next sequential instruction is executed.

- CALL

  1. Near Call

  The new subroutine called must be in the same segment. The call address can be specified directly in the instruction or indirectly through registers or memory locations.

  2. Far Call

  The new subroutine called is in another segment. Here CS and IP both get new values.

- RET

  RET – Return instruction causes the control to return to the main program from the subroutine

3)
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