**PRACTICAL # 08**

**OBJECT:**

Working with procedures.

**THEORY:**

Procedures or functions are very important to a program. They help divide a program into logical and more manageable parts. They are the core for procedural programming paradigm.

A function call is a sort of branching instruction that breaks the execution sequence. It requires the value of **IP** to point to some other part of the code containing the function. After the called function finishes execution, it must resume execution to the last function that called it. Further, it must also return to the same line of code in calling function from where it was called. To ensure this, at the time of function call, the value of **IP** is saved on stack by **call** instruction. At the end of called function, the **ret** instruction makes the **IP** get its previous value from the stack thus gets the calling instruction address and resumes execution there.

The general syntax for procedure definition is

**procname proc {NEAR or FAR}**

**<statements>**

**procname endp**

Where NEAR or FAR is optional that defines whether the function resides in the same code segment.

Near calls and returns transfer control between procedures in the same code segment. Far calls and returns pass control between different segments, so a far call also pushes **CS**, register value on the stack and **far ret** pops back **CS** value along with the **IP** register value.

Assembler uses {NEAR or FAR} operands to determine if you're calling this procedure with a near or far call instruction, **default is NEAR** if one is not mentioned. Consider the following two procedures:

NProc proc near

mov ax, 0

ret

NProc endp

FProc proc far

mov ax, 0FFFFH

ret

FProc endp

call NPROC ; calling near proc

call FPROC ; calling far proc

The assembler automatically generates a three-byte (near) call for the first call instruction above because it knows that NPROC is a near procedure. It also generates a five-byte (far) call instruction for the second call because FPROC is a far procedure. Within the procedures, the assembler automatically converts all **ret** instructions to near or far returns depending on the type of routine.

We will use NEAR procedures here.

**Program:**

The program below defines a main function. It further defines two functions; the newlineProc that outputs a new line on the console and prntChar procedure that just prints a character.

Note that the procedures have no arguments. Argument passing requires the knowledge of stack management with respect to function calling mechanism, which will be covered later.

.model small

.stack 100h

.data

str db 'calling a procedure for new line$'

newline db 0ah, 0dh, '$'

.code

main proc

mov ax,@data

mov ds, ax

mov ah, 9

mov dx, offset str

int 21h

call newlineProc ; calls function newlineProc

call prntChar

ret ; return is must, otherwise control will fall through

main endp

prntChar proc ; procedure definition starts

mov ah, 2

mov dl, 'b'

int 21h

ret ; return to the function that called it

prntChar endp  ; procedure definition ends

newlineProc proc ; procedure definition starts

mov ah, 9

mov dx, offset newline

int 21h

ret ; procedure definition ends

newlineProc endp ; procedure definition ends

end main

The above program starts by defining two strings str 'calling a procedure for new line$' and newline 0ah, 0dh, '$'.

Next it define two other functions; the newlineProc to outputs a new line on the console and prntChar procedure to prints a character. The main function then calls both these functions as:

call newlineProc

call prntChar

When each function finishes execution, it returns the control back to the main function from where it was called.

**ACTIVITIES**

**Activity 1**

Try implementing the above program but without using the **ret** statement at the end of functions and observe the output.

**Activity 2**

Write a function that prints alphabets in reverse order, without using array.

**Activity 3**

Write a function to print a multi-digit number.

**REVIEW QUESTIONS**

1. What happens if RET statement is missed at the end of a function?
2. How does a call instruction like CALL someFunction works?
3. What is the difference between a near and far procedure.
4. How do functions help make a program manageable?