**PRACTICAL # 09**

**OBJECT:**

Procedures arguments, local data and return values.

**THEORY:**

Procedures help divide a program into logical and more manageable parts. The procedures accept arguments to work on and they can also return values. Function parameters use stack for passing arguments. The local data of a function is also stored on the stack.

In a C function call, arguments are passed to the function by pushing one argument at a time onto the stack in reverse of the order as listed in the function declaration. Near functions are of the following form (or an equivalent form):

MyFunc proc

push bp ; (1) save bp

mov bp, sp ; (2) set bp for referencing stack

sub sp, {local data size} ; (3) allocate space for local variables

...

... ; function body

...

mov {return reg}, {return value} ; (4) set return value

mov sp, bp ; (5) free space used by local variables

pop bp ; (6) restore bp

ret

MyFunc endp

The first thing a C function does is save **BP** register.

Then put a copy of **SP** register into **BP**. Storage space for local variables is reserved on the stack. These first three steps set up the **stack frame** for the function. Since the initial value of **SP** is saved in register **BP.** **BP** register is used as a reference to access the arguments that were pushed onto the stack when the function was called as well as the local variables declared within the function.

At the end, if the function returns a value, it is placed in the appropriate register. To clean up the stack frame, the stack is restored to its initial value, freeing the local data space that was allocated, and **BP** is restored, thus removing the stack frame. **BP** value should not be modified in the function body.

**Note:** If there are no local variables then there is no need to reserve space on the stack.

If there are no function arguments or local variables then it may not be necessary to set up and restore the stack frame.

**Use stack to save and restore registers that are used in the function**

Example: To use the register BX in a function, execute **push bx** at the beginning of the function and **pop bx** at the end.

**Program:**

.model small

.stack 100d

.data

.code

main proc

mov bx, 5

mov ax, 'a'

push bx

push ax

call printArg

mov ah, 2

mov dl, 'z'

int 21h

mov ah, 4ch

int 21h

ret

main endp

printArg proc

push bp

mov bp, sp ; store sp current value for relative addressing

sub sp, 2 ; reserve 2 bytes/1 words on stack for local variables

; save the registers values on stack before use

push ax

push bx

push dx

mov [bp-1], 2 ; initialize local variable

mov [bp-2], 3

mov dx, [bp+6] ; access first argument

add dx, 48

mov ah, 2

int 21h

mov dx, [bp+4] ; access second argument

int 21h

mov al, [bp-1] ; access local vaiable1

mov bl, [bp-2] ; access local vaiable1

mul bl

mov dl, al

add dl, 48

mov ah, 2

int 21h

;restore the registers original values

pop dx

pop bx

pop ax

add sp, 2 ; unallocate reserved bytes on stack

pop bp ; restore bp original value

ret ; restore ip to caller function

printArg endp

end main

**ACTIVITIES**

**Activity 1**

Write a simple function that gets two numbers as arguments and returns their sum in a register.

**Activity 2**

Find out a way to print the string in reverse order and write a program to that.

**Activity 3**

Write a program that finds sum of elements of an array of size three.

**REVIEW QUESTIONS**

1. What is the purpose of SP register and SS register in stack management?
2. In which order does the stack grow as data is pushed into it?
3. What will be the stack overflow and underflow conditions?
4. How are the parameters to a function accessed?

1. How are local variables created and accessed in a function?
2. What are the function **calling conventions**?

1. How is the current register data saved before using the register in some called function?