**Lab 05 : Stack and Procedures**

**Introduction:**

In this lab, you will be introduced to the Stack and Procedures in assembly language. You will also develop the understanding of Procedure calls and returns of 8086 family of processors.

**Objectives:**

After this lab, you will be able to:

• Use PUSH and POP instructions in assembly language

• Use CALL and RET instructions in assembly language

• Define your own procedures

**Stack:**

The stack is a special segment in memory used to facilitate subroutine handling. The stack always starts at a high address and grows towards the beginning of the stack segment at a lower address. When a program starts, the stack is empty, and its size is zero. The microprocessor stores data on the stack as needed, and uses the SP register to point to the last item stored on the stack. The stack size dynamically changes as data is stored or retrieved from the stack.

The PUSH instruction is used to store the content of a 16-bit register, or memory location, on the stack. It first decreases the content of SP by two and then stores the data into the two bytes on the top of the stack. The high order byte of the data goes to the high addressed byte in the stack.

All pushes and pops are 16 operations. There is no (easy) way to push a single eight bit value onto the stack. To push an eight bit value you would need to load it into the H.O. byte of a 16 bit register, push that register, and then add one to the stack pointer. Therefore, most programs push or pop 16 bits, even when dealing with eight bit values.

|  |  |
| --- | --- |
| **Instruction** | **Example** |
| PUSH | PUSH AX |
| POP | POP AX |

**Task 1:**

What do POPF and PUSHF instructions do? Do they affect flags in any way?

**Procedures**

A procedure is a set of instructions that compute some value or take some action (such as printing or reading a character value).Most procedural programming languages implement procedures using the call/return mechanism. That is, some code calls a procedure, the procedure does its thing, and then the procedure returns to the caller. The calling code calls a procedure with the **CALL** instruction, the procedure returns to the caller with the **ret** instruction.

A simple procedure may consist of nothing more than a sequence of instructions ending with a ret instruction. For example the following “procedure” zeros out the 256 bytes

starting at the address in the BX register:

You should use **proc** and **endp** assembler directives. The ZeroBytes routine, using the proc and endp directives is

ZeroBytes proc

XOR AX, AX MOV CX, 128

ZeroLoop:

MOV [BX], AX ADD BX, 2

LOOP ZeroLoop

RET

ZeroBytes ENDP

The PROC directive can be followed by the type of the procedure: NEAR (intra-segment) or

FAR (inter- segment).

**The CALL Instruction:**

The CALL instruction transfers the flow of the program to the procedure. The CALL instruction differs from the jump instruction in the sense that a CALL saves a return address on the stack. The RET instruction return control to the instruction that

immediately follows the CALL. There exist two types of calls: FAR and NEAR. .

**Near CALL:**

The near call instruction does the following:

a) It pushes the 16 bit offset of the next instruction following the call onto the stack.

b) It copies the 16 bit effective address into the IP register.

c) Execution continues at the first instruction of the subroutine.

This first instruction is the opcode at the target address computed in the previous step.

**Far CALL:**

The FAR CALL can call a procedure anywhere in the system memory. It is a five-

byte instruction that contains an opcode followed by the next value for the IP and CS

registers. Bytes 2 and 3 contain the new contents of IP, while bytes 4 and 5 contain the new contents for CS.

The far call instruction does the following:

a) It pushes the CS register onto the stack.

b) It pushes the 16 bit offset of the next instruction following the call onto the stack.

c) It copies the 32 bit effective address into the CS:IP registers.

d) Execution continues at the first instruction of the subroutine. This

first instruction is the opcode at the target address computed in the previous step.

**Task 2:**

Prompt the user for entering a word terminated by enter. Display the characters of the word in reverse order using stack. Properly comment the code. Make the console intuitive. Attach the code and screenshot of console.

; You may customize this and other start-up templates;

; The location of this template is c:\emu8086\inc\0\_com\_template.txt

org 100h

.data

str db 13,10, 'Enter your name:$'

buffer DB 13,10, 08, 09 DUP(?) 13,0ah

space DB 20h

.code

main proc

mov AX, @data

mov ds,ax

mov dx,offset str

mov ah,9

int 21h

MOV AH, 0AH

MOV DX, OFFSET buffer

INT 21H

mov SI,OFFSET buffer+2

mov cx,8

L1:

mov bx,[SI]

push bx

inc SI

Loop L1

mov cx,8

L2:

pop dx

mov ah,2

int 21h

loop L2

mov ah,4ch

int 21h

main endp

ret

**Task 3:**

Write an assembly code for calculating the factorial of a number input by user. Attach code and screenshot of console.