LAB 4

The Stack and Introduction to Procedures – **PUSH, PUSHF and POP**

The Stack

A stack is one-dimensional data structure. It is processed in a "last-in, first-out" manner. The most recent addition to the stack is called the top of the stack. A program must set aside a block of memory to hold the stack. We have been doing this by declaring a stack segment;

for example,

.STACK 100H

When the program is assembled and loaded in memory, SS will contain the segment number of the stack segment. For the preceding stack declaration, SP, the stack pointer, is initialized to 100h. This represent the empty stack position. When the stack is non-empty, SP contains the offset address of the top of the stack.

PUSH and PUSHF Instruction

PUSHF: Push the Flag Register value to the stack. First the upper byte is pushed and then the lower byte)
To add a new word to the stack we PUSH it on. The instruction PUSHF, which has no operands, pushes the contents of the FLAGS register onto the stack. Initially, SP contains the offset address of the memory location immediately following the stack segment; the first PUSH decreases SP by 2, making it point to the last word in the stack segment. Because each PUSH decreases SP, the stack grows toward the beginning of memory.

Syntax: PUSH source; where source is a 16-bit register or memory word.

Example: PUSH AX

Execution of PUSH cause the following to happen:

1. SP is decreased by 2.

2. A copy of the source content is moved to the address specified by SS: SP. The

source is unchanged.

Example: The sequence of operation for the instruction PUSH AX is as follows:-

- Current stack top is already occupied so decrement SP(Stack Pointer) by one and store AH into the address pointed to by SP.

- Further, decrement SP by one and store AL into the location pointed to by SP.

- Examples of instructions are:-

➢ PUSH AX

➢ PUSH DS

➢ PUSH [5000H] (Content of 5000H and 5001H in DS are pushed on to the stack)

again,

The push instructions move data onto the 80x86 hardware stack and the pop instructions move data from the stack to memory or to a register. The following is an algorithmic description of each instruction:

push instructions (16 bits):

SP := SP - 2

[SS:SP] := 16 bit operand (store result at location SS:SP.)

pop instructions (16 bits):

16-bit operand := [SS:SP]

SP := SP + 2

push instructions (32 bits):

SP := SP - 4

[SS:SP] := 32 bit operand

pop instructions (32 bits):

32 bit operand := [SS:SP]

SP := SP + 4

POP and POPF Instruction

To remove the top item from the stack we POP it. The instruction POPF pops the top of the stack into the FLAGS register

Syntax: POP destination; where destination is a 16-bit register or memory word.

Example: POP BX

Executing POP causes this to happen:

1. The content of SS: SP (the top of the stack) is moved to the destination.

2. SP is increased by 2.

Example:

POP Instruction contd…

• Examples of instructions are:-

- POP AX

- POP DS

- POP [5000H]

• Effectively SP is incremented by 2 and points to the next stack top.

Ex: Let AX = 5522H

• => AH=55H and AL=22H

• When PUSH AX is executed, AH=55H is first pushed on to the stack and then AL=22H is pushed.

• Similarly, when POP AX is encountered, first

22H(AL) is deleted from stack and then 55H(AH).

Stack Instructions

There is no effect of PUSH, PUSHF, POP, POPF on the flags. Note that PUSH and POP are word operations, so a byte instruction such as PUSH DL is illegal. A push of immediate data, such as PUSH 2 is also illegal. In addition to the user's program, the operating system uses the stack for its own purposes. For example, to implement

the INT 21h functions, DOS saves any registers it uses on the stack and restores them when the interrupt routine is completed. This does not cause a problem for the user because any values DOS pushes onto the stack are popped off by DOS before it returns control to the user's program.

Program : Implement push, pop.

Array

Arrays:

It is necessary to treat a collection of values as a group. The advantage of using an array to store the data is that a single name can be given to the whole structure and an element can be accessed by providing an index. In Assembly language, an array is a contiguous block of memory containing elements of the same data type (usually bytes or words). Unlike high-level languages, there's no built-in array type — we manually declare a sequence of bytes or words and access them via offsets and registers.

Term	Explanation
.data	Segment where we define arrays (data)
db	Define Byte (8-bit values)
dw	Define Word (16-bit values)
SI/DI/BX	Index registers used to access array elements
MOV	Used to load/store data

Byte Array declaration: MSG DB 'abcde’

Word Array declaration: Word DW 10, 20, 30, 40, 50, 60

Program1 : Take 5 numbers in an array and print the sum. For sum the numbers use loop.

Program2 : Reverse a string using Stack.