**Lab 09: Stack Operations**

**OBJECTIVE:** Learn about runtime stack in Assembly.

# Objectives:

* Stack Pointer
* Stack Segment
* PUSH Operation
* POP Operation

Stack is an area of memory for keeping temporary data.

**PUSH** and **POP** instruction are especially useful because we

don't have too much registers to operate with, so here is a trick:

* Store original value of the register in stack (using **PUSH**).
* Use the register for any purpose.
* Restore the original value of the register from stack (using
* **POP**).

**To use stack, stack segment must be defined in program.**

**.stack** size

If size is 100d, it reserves 100 bytes for stack.

If size is 100h, it reserves 256 bytes for stack.

Example: .stack 100h

Stack is used by **CALL** instruction to keep return address for procedure, **RET** instruction gets this value from the stack and returns to that offset. Quite the same thing happens when **INT** instruction calls an interrupt, it stores in stack flag register, code segment and offset. **IRET** instruction is used to return from interrupt call.

We can also use the stack to keep any other data.

The stack uses **LIFO** (Last In First Out) algorithm, this means that if we push these values one by one into the stack:

**1, 2, 3, 4, 5**

the first value that we will get on pop will be **5**, then **4**, **3**, **2**, and only then **1**.

It is very important to do equal number of **PUSH**s and **POP**s,

otherwise the stack maybe corrupted and it will be impossible to return to operating system.

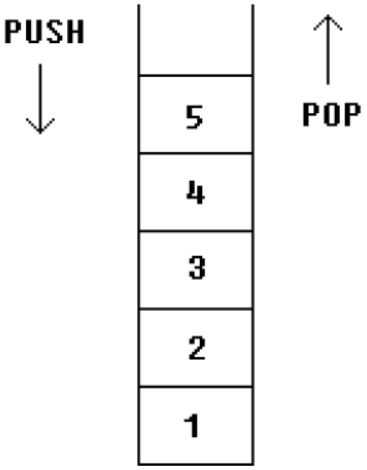
**Stack Segment (SS):**

* Specifies the segment address of Stack.
* Added to address during stack access.

**Stack Pointer (SP):**

* Stack Pointer Register (SP) points the top of Stack.
* Suppose the Stack size is set to 256bytes.
  + Stack Pointer will be Initialized with Value 256.
  + Depending on Architecture, PUSH and POP Operations will change Value of Stack Pointer.

|  |  |  |
| --- | --- | --- |
| **Pop** | REG  SREG  memory | **Get 16-bit value from the stack.**  **Algorithm:** Adds 2 to SP register.  l operand = SS:[SP] (top of the stack)  l SP = SP + 2  **Example:**  MOV AX, 1234h  PUSH AX  POP DX ; DX = 1234h  RET |
| **Push** | REG  SREG  memory  immediate | **Store 16-bit value in the stack.**  **Algorithm:** Subtracts 2 from SP register.  l SP = SP - 2  l SS:[SP] (top of the stack) = operand  **Example:**  MOV AX, 1234h  PUSH AX  POP DX ; DX = 1234h  RET |



**Example:** Push and Pop values from stack word sized and byte sized.

MOV AX, 1234h

PUSH AX ; store value of AX in stack.

MOV AX, 5678h ; modify the AX value.

POP AX ; restore the original value of AX.

**Example:** Another use of stack is for exchanging the values.

MOV AX, 1212h ; store 1212h in AX.

MOV BX, 3434h ; store 3434h in BX

PUSH AX ; store value of AX in stack.

PUSH BX ; store value of BX in stack.

POP AX ; set AX to original value of BX.

POP BX ; set BX to original value of AX.

**Activities**:

* Initialize an Array of 5 Indexes.
  + Declare main procedure
  + Initializer list having 0,2,4,8,10 on respective indexes
  + Push values of Array in a Stack
  + Pop & Print each value on Console.
  + Beware to deal with n-digit number while printing.
* Write a Procedure which when called, prints time on Console. (Review Lab # 02, Slide # 25)
* Write a Procedure which takes 5 numbers as argument. Calculate sum of numbers and return the sum. Prints the sum in main procedure. You have to use Stack for Passing and Retrieving Arguments.