Lab Session 05

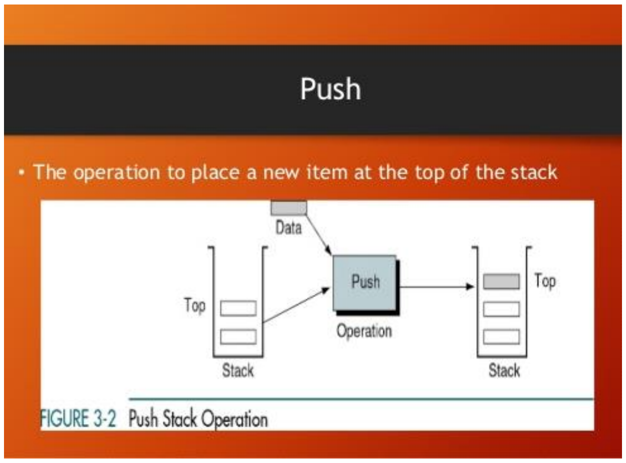
Objectives

Arrays & Loop

a. Stack Basics
b. JMP instruction
c. LOOP instruction

---

a. Stack Basics

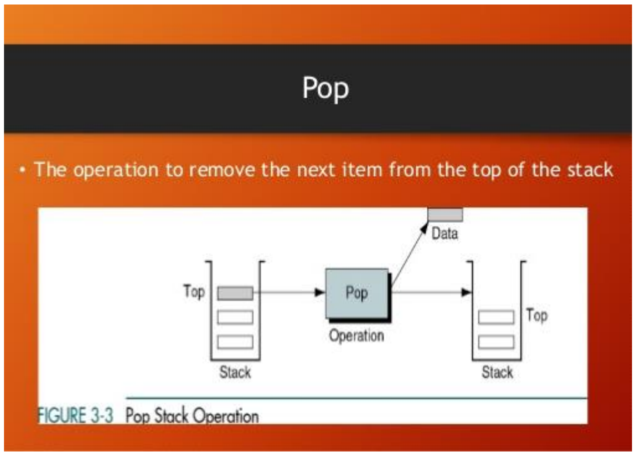


Push

• The operation to place a new item at the top of the stack

Data
Top
Push
Operation
Stack
Top
Stack

FIGURE 3-2 Push Stack Operation



Pop

• The operation to remove the next item from the top of the stack

Data
Top
Pop
Operation
Stack
Top
Stack

FIGURE 3-3 Pop Stack Operation

A stack is an array-like data structure in the memory in which data can be stored and removed from a location called the 'top' of the stack. The data that needs to be stored is 'pushed' into the stack and data to be retrieved is 'popped' out from the stack. Stack is a LIFO data structure, i.e., the data stored first is retrieved last.

Assembly language provides two instructions for stack operations: PUSH and POP. These instructions have syntaxes like −

```
PUSH    operand
POP     address/register
```

The memory space reserved in the stack segment is used for implementing stack. The registers SS and ESP (or SP) are used for implementing the stack.

The top of the stack, which points to the last data item inserted into the stack is pointed to by the SS:ESP register

SS register points to the beginning of the stack segment and the SP (or ESP) gives the offset into the stack segment.

The stack implementation has the following characteristics −

- Only **words** or **doublewords** could be saved into the stack, not a byte.
- The stack grows in the reverse direction, i.e., toward the lower memory address
- The top of the stack points to the last item inserted in the stack; it points to the lower byte of the last word inserted.

As we discussed about storing the values of the registers in the stack before using them for some use; it can be done in following way

```
; Save the AX and BX registers in the stack
PUSH    AX
PUSH    BX

; Use the registers for other purpose
MOV    AX, VALUE1
MOV    BX, VALUE2
...
MOV    VALUE1, AX
MOV    VALUE2, BX

; Restore the original values
POP    AX
POP    BX
```

b. JMP instruction

SYNTAX:
JMP destination

EXAMPLE:

```
.code
top:
        ; any statements
jmp top
```

c. LOOP instruction

SYNTAX:
LOOP destination

EXAMPLE # 01:

```
.data
intArray WORD 100, 200, 300, 400, 500
.code
main PROC
mov esi, 0
mov ax, 0
mov ecx, LENGTHOF intArray
L1:
        mov ax, intArray [esi]
        add esi, TYPE intArray
loop L1
```

EXAMPLE # 02:

```
mov eax, 0
mov ebx, 0
mov ecx, 5
L1:
        inc eax
        mov edx, ecx
        mov ecx, 10
        L2:
                inc ebx
        loop L2
        mov ecx, edx
loop L1
```

ACTIVITIES:

1. Use a loop with direct or indirect addressing to reverse the elements of an integer array in place. Do not copy elements to any other array. Use SIZEOF, TYPE and LENGTHOF operators to make program flexible.

2. Write a program that uses a loop to calculate the first ten numbers of Fibonacci sequence.

3. Write a program to sort the following array using Bubble Sort algorithm:

```
myArray          BYTE        15, 10, 25, 20, 30
```