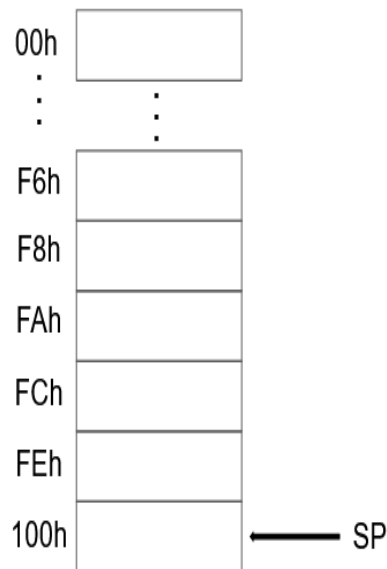


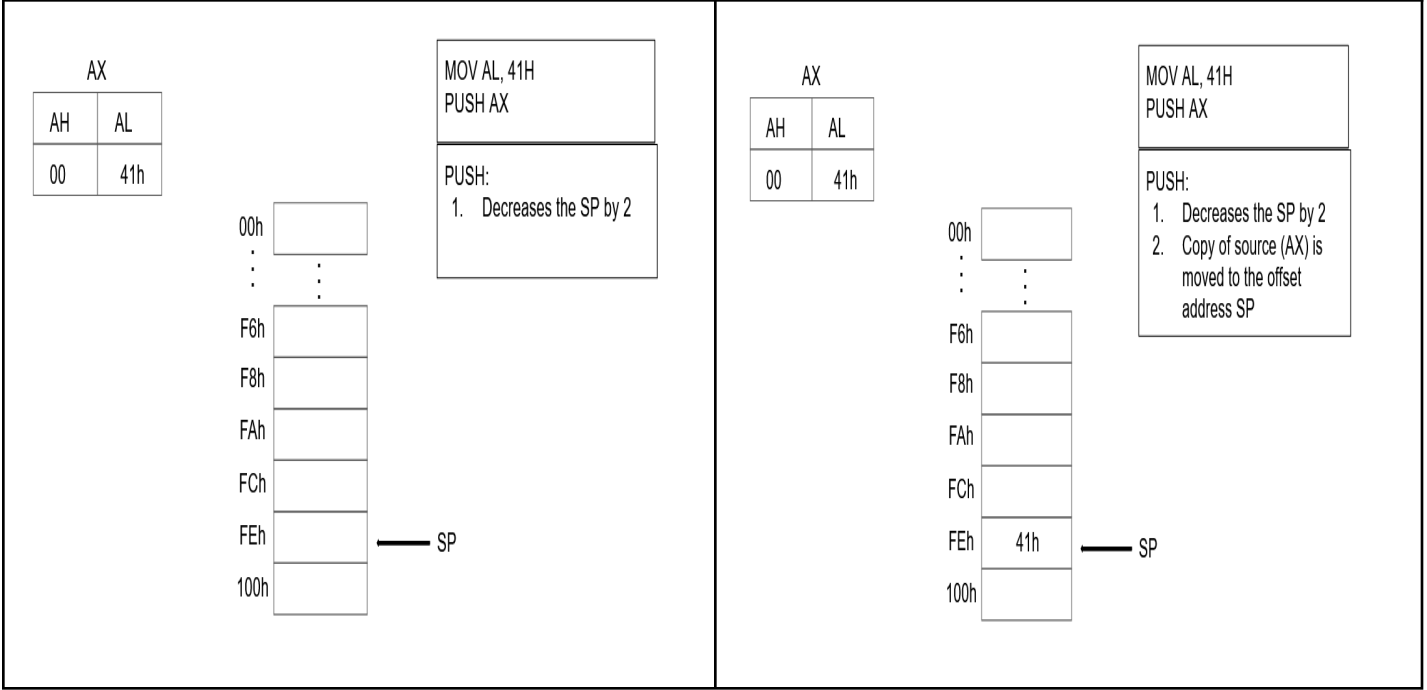


STACK:

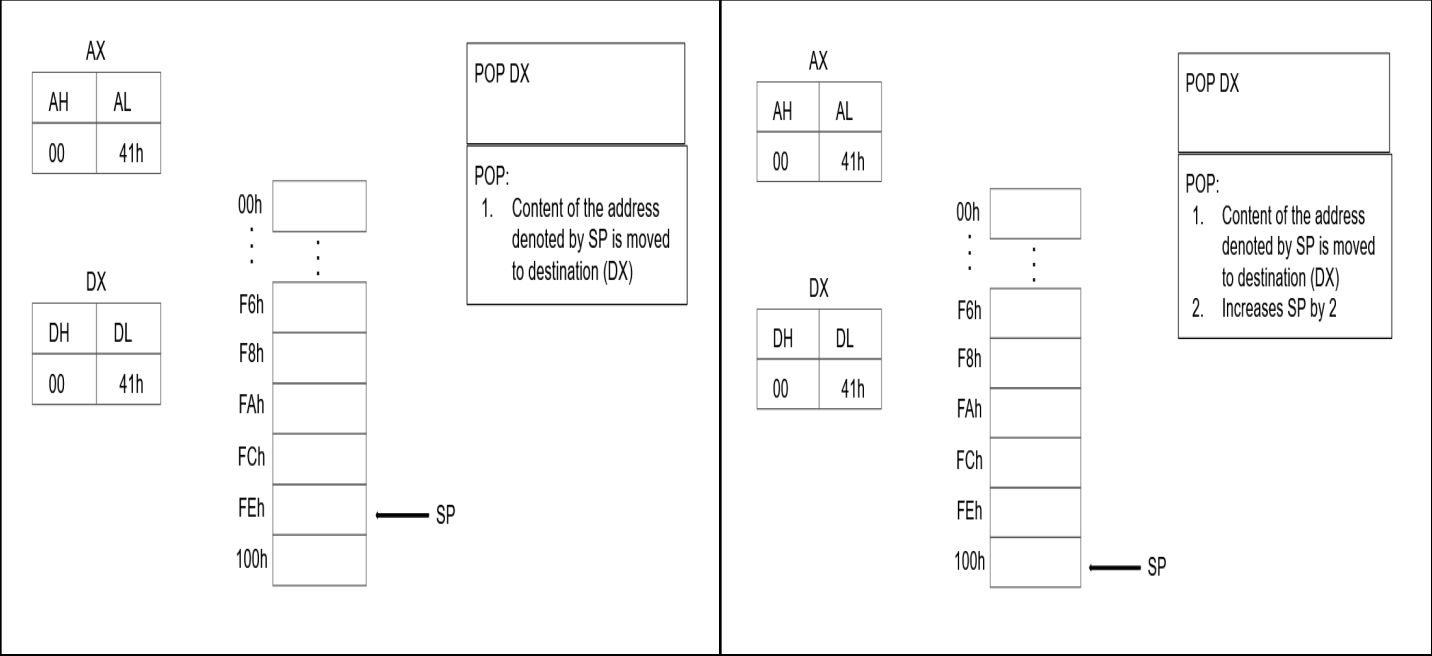
1. PUSH: **PUSH source**
source must be a **16-bit register or memory word**.
2. First **decrease SP by 2**, then copy of the source is moved to **SS:SP**.
3. SP contains the offset address of the top of the stack.
4. POP: **POP destination**
destination must be a **16-bit register or memory word**.
5. First, the content of **SS:SP** (top of the stack) is moved to destination, then SP **increases by 2**



PUSH instruction



POP instruction



Example: Write a program that takes a text input from the user until the user enters a **carriage return**. Display the text in reverse order (you must use stack instructions).

```
.MODEL SMALL
.STACK 100H
.DATA
    MSG1 DB "Enter a line of text: $"
    MSG2 DB 0AH, 0DH, "Reverse: $"
    MSG3 DB 0AH, 0DH, "The stack is empty.$"

.CODE
    MOV AX, @DATA
    MOV DS, AX

    MOV AH, 9
    LEA DX, MSG1
    INT 21H

    MOV CX, 0           ;input count set to 0

INPUT:                 ;take input
    MOV AH, 1
    INT 21H
    CMP AL, 0DH         ;check if CRET?
    JE DISPLAY          ;if CRET, exit loop & go to display

    XOR AH, AH          ;clear AH
    PUSH AX             ;push the input on stack
    INC CX              ;increment input count
    JMP INPUT           ;loop back

DISPLAY:
    CMP CX, 0H          ;check if stack is empty
    JE EMPTY_STACK     ;if empty go to empty_stack

    MOV AH, 9           ;if not not empty, display the reverse
    LEA DX, MSG2
    INT 21H

REV:
    POP DX              ;get a character from stack
    MOV AH, 2
    INT 21H            ;display the character

    LOOP REV
    JMP EXIT           ;as done with display, go to exit
```

```
EMPTY_STACK:
MOV AH, 9
LEA DX, MSG3      ;show the empty stack message
INT 21H

EXIT:
MOV AH, 4CH      ;terminate program
INT 21H
```

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make programs more structural and easier to understand. Generally the procedure returns to the same point from where it was called.

The syntax for procedure declaration:

```
name PROC
    ; here goes the code
    ; of the procedure ...
RET
name ENDP
```

name - is the procedure name, the same name should be in the top and bottom, this is used to check the correct closing of procedures.

RET instruction is used to return the control from procedure to the caller.

PROC and **ENDP** are compiler directives, so they are not assembled into any real machine code. Compiler just remembers the address of the procedure.

CALL instruction is used to call a procedure.

```
INCLUDE EMU8086.INC
```

```
.MODEL SMALL
```

```
.STACK 100H
```

```
.CODE
```

```
MAIN PROC
```

```
    CALL SCAN_NUM ;LIBRARY PROC
```

```
    PRINTN
```

```
    MOV AX, CX
```

```
    CALL SCAN_NUM ;LIBRARY PROC
```

```
    PRINTN
```

```
    CALL SUM      ;USER-DEFINED PROC
```

```
    CALL PRINT_NUM ;LIBRARY PROC
```

```
    EXIT:
```

```
    MOV AH, 4CH
```

```
    INT 21H
```

```
MAIN ENDP
```

```
SUM PROC
```

```
    ADD AX, CX
```

```
    RET
```

```
SUM ENDP
```

```
DEFINE_SCAN_NUM
```

```
DEFINE_PRINT_NUM
```

```
DEFINE_PRINT_NUM_UN$
```