The Stack and Introduction to Procedures

Course Code: CSC 2106

Course Title: Computer Organization and Architecture

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:		**Week No:**		**Semester:**	**Fall 23**
Lecturer:	*Saeeda Sharmeen Rahman*				

Lecture Outline

- The stack segment of a program is used for **temporary storage** of data and addresses.
- In this chapter we will see how the stack is **manipulated.**
- How stack is used to implement procedures.
- The **PUSH and POP** Instructions that **add and remove** words from the stack.
- Because the last word to be added to the stack Is the first to be removed(**LIFO**), A stack can be used to **reverse** a list of data

Lecture Outline

- Procedures are extremely important in all programming language.
- We will discuss the essentials of assembly language procedures.
- At the machine level, we can see exactly how a **procedure is called** and how it returns to the **calling program**.
- An example of procedure will be discussed to perform the binary multiplications and DEBUG program.

The Stack

- A stack is **one-dimensional** data structure.
- Items are added and removed from **one end** of the structure; that is, 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 familiar example is a Stack of dishes; the last dish to go on the stack is the top one, and it's the only one that can be removed easily.
- 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 IOOH

The Stack(cont'd…)

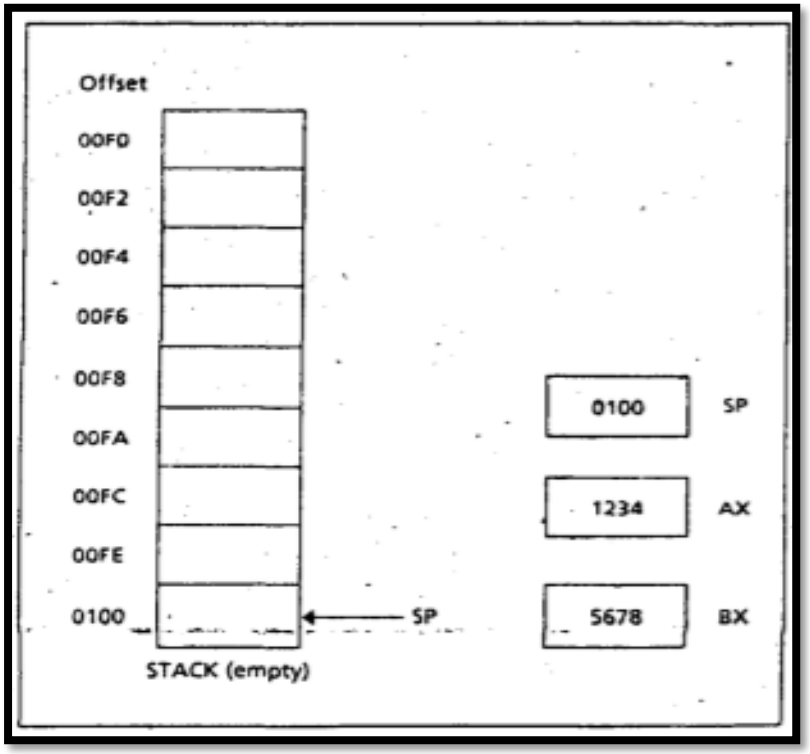
- When the program is assembled and loaded in memory , SS will contain the segment number of the stack segment.
- For the preceding Stack declaration, SP, the stack pointer, is initialized to lOOh.
- This represents empty stack position.
- When the stack is empty, SP contains the offset address of the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

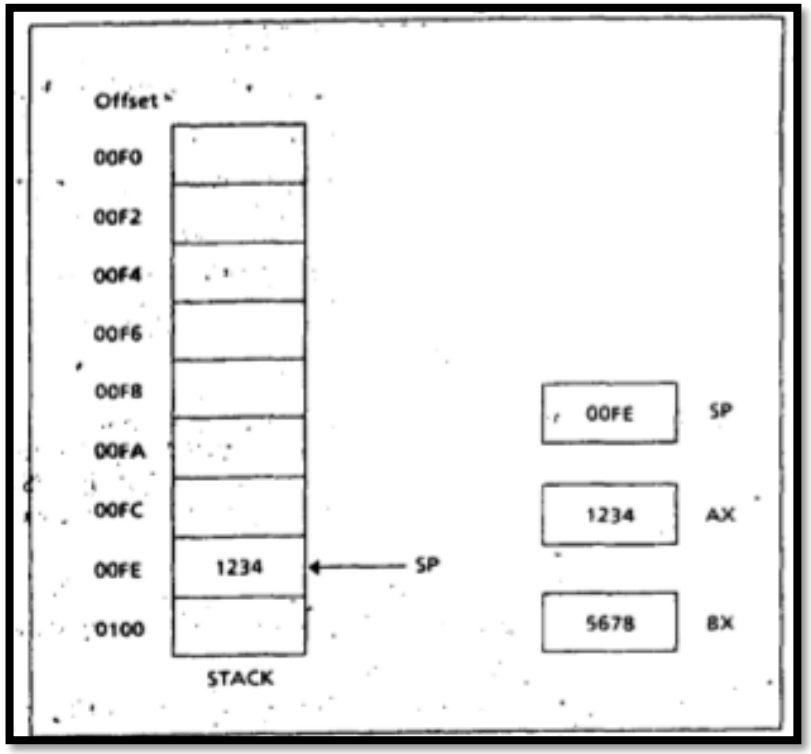
PUSH AND PUSHF

- PUSH is used to **add new word** to the stack.
- The syntax is:

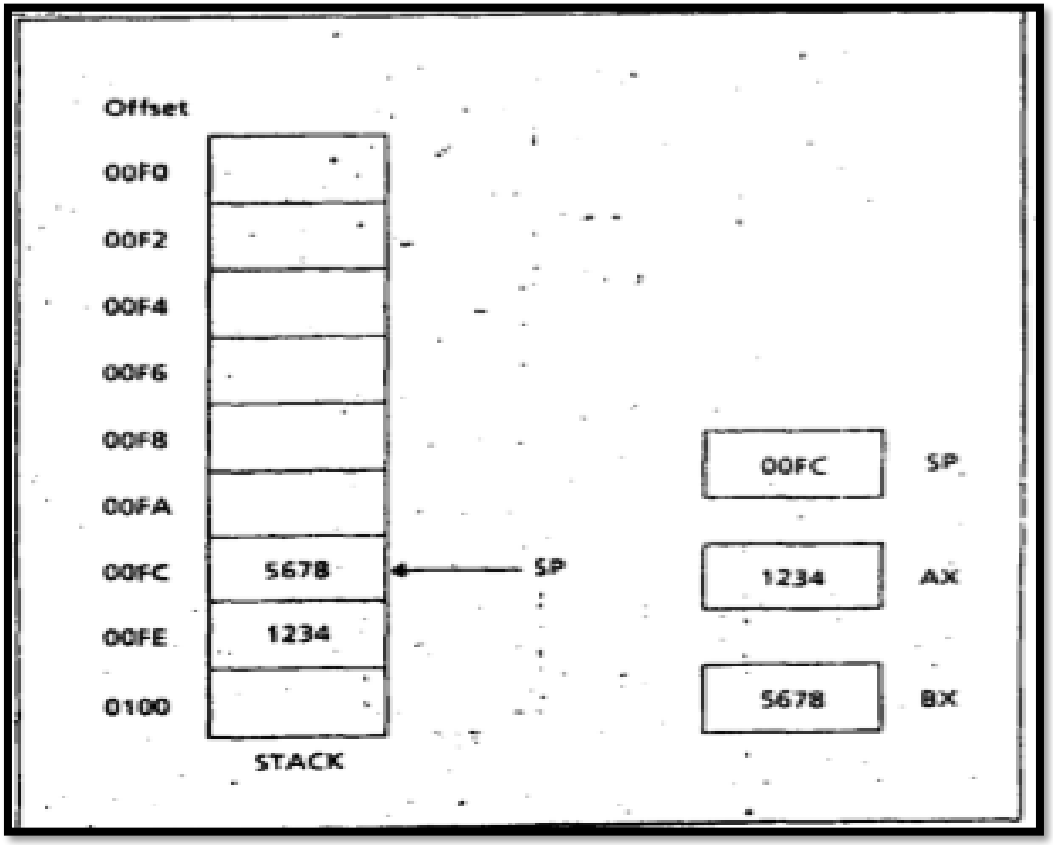
  PUSH Source (i.e. PUSH AX)
- **SP is decreased by 2**
- A copy of source content is moved to the address specified by SS:SP.
- Initially, SP contains the offset address of memory location.
- The first PUSH decreases SP by 2 and point to the LAST WORD in the STACK segment.
- **PUSHF** has no operands and pushes the contents of the flag register to the stack.

PUSH OPERATION





PUSH OPERATION (cont'd…)

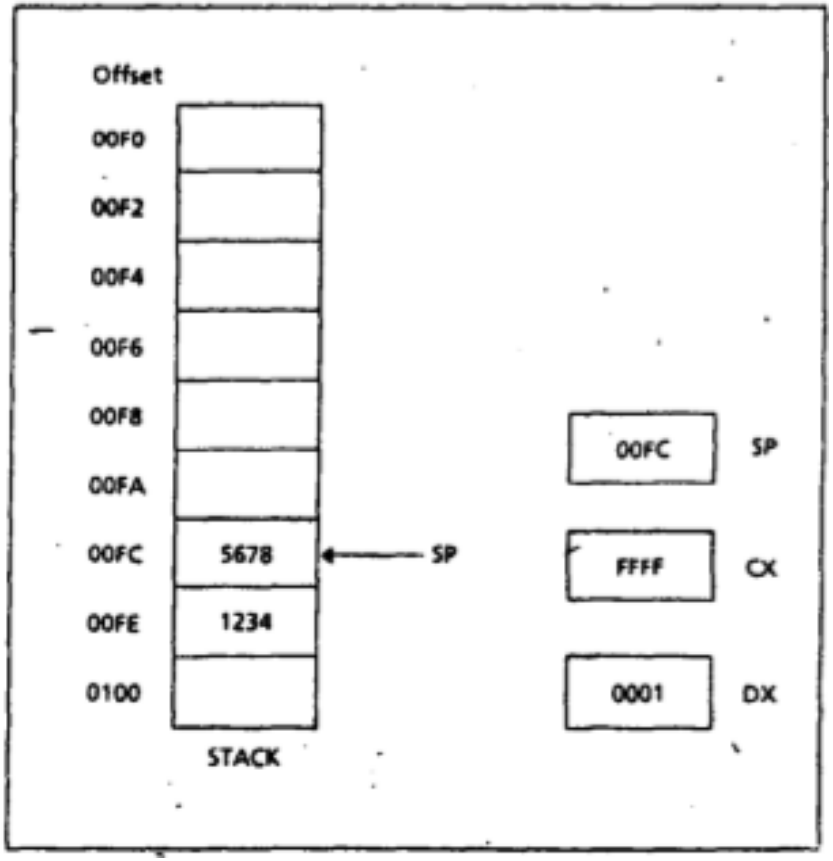


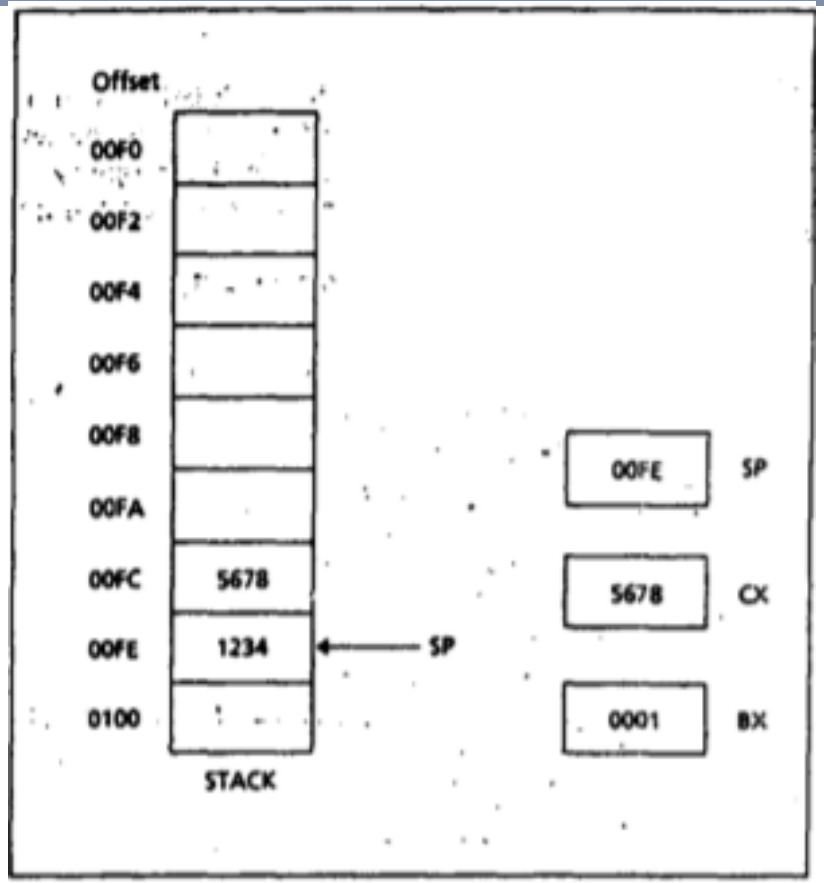
POP AND POPF

- POP is used to **remove an item from** the stack.
- The syntax is:

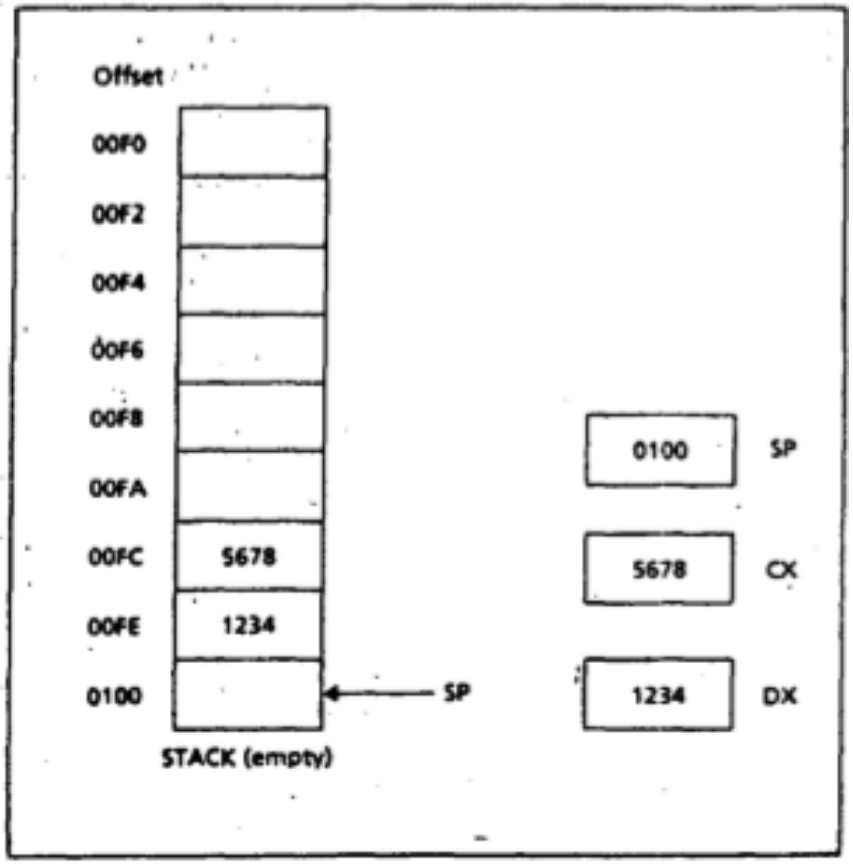
  POP destination (i.e. POP AX)
- The content of SS:SP (the top of the stack) Is moved to the destination.
- **SP is Increased by 2.**
- The Instruction **POPF, pops the top** of the stack into the FLAGS register.
- **There is no effect of PUSH, PUSHF. POP, POPF on the flags.**
- Note that PUSH and POP are **word operations**, so a byte Instruction(i.e. PUSH DL or PUSH 2) is illegal.
- For INT 21h DOS saves instructions in STACK before execution.

POP OPERATION





POP OPERATION (cont'd…)



STACK Application

- **Algorithm to Reverse Input**

```
Display a '?'
Initialize count to 0
Read a character
WHILE character is not a carriage return DO
                    push character onto the stack
                    increment count
                    read a character
END WHILE:
Go to a new 1ine

FOR count times DO

        pop a character from the stack;

        display it;

END FOR
```

Terminology of Procedures

- The idea is to take the original problem and decompose it into a series of sub problems that are easier to solve than the original.
- Like high level languages, an assembly language program can also be structured as a collection of procedures.

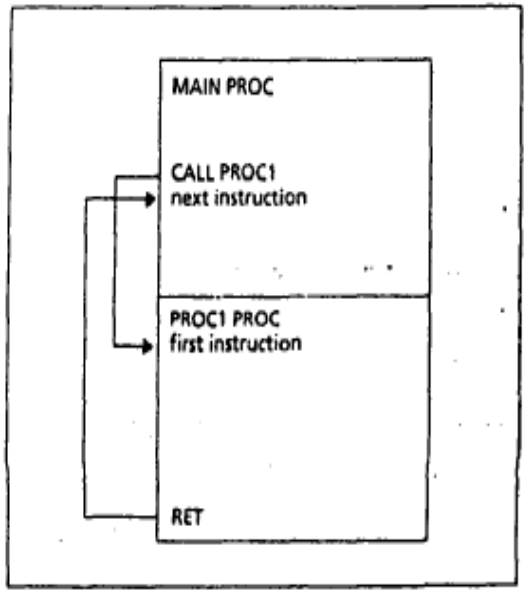
- One of the procedures is the main procedure containing the entry point to the program.

- To carry out a task the main procedure calls one of the other procedures. It is also possible for these procedures to call each other or for a procedure to call itself.
- **Procedure declaration:**

```
name PROC type
; body of the procedure
RET
name ENDP
```

Procedure Call and Return

- Name is the user defined name of the procedure.
- **Near:** It means that the statement that calls the procedure is in the same segment as the procedure it self.
- **Far:** It means that the calling statement is in the a different segment.



Procedure Call and Return (cont'd...)

- Ret : The **ret** instruction causes control to transfer back to the calling procedure.
- Every procedure should have a **ret** someplace (except main procedure)

References

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).

- https://www.csie.ntu.edu.tw/~cyy/courses/assembly/10fall/lectures/handouts/lec15_x86procedure_4up.pdf

Books

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).

- Essentials of Computer Organization and Architecture, (Third Edition), Linda Null and Julia Lobur

- W. Stallings, “Computer Organization and Architecture: Designing for performance”, 67h Edition, Prentice Hall of India, 2003, ISBN 81 – 203 – 2962 – 7

- Computer Organization and Architecture by John P. Haynes.