

# Microprocessors

Dr Mridul Gupta

Assistant Professor

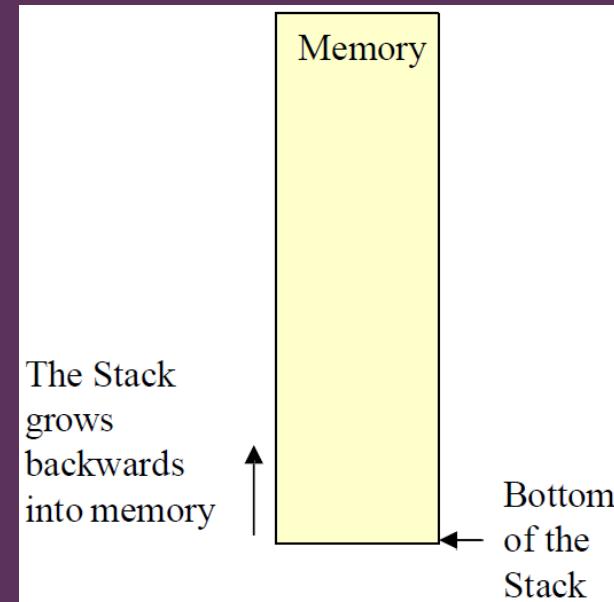
Dept. of ECE

**UNIT-2**

**Stack and Subroutine**

# The Stack

- The stack is an **area of memory** identified by the programmer for temporary storage of information.
- **The stack is a LIFO structure.**
  - Last In First Out.
- The stack normally grows backwards into memory.
  - In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.



# The Stack

- In the 8085, the stack is defined by setting the **SP** (Stack Pointer) **register**.

e.g.

LXI SP, FFFFH

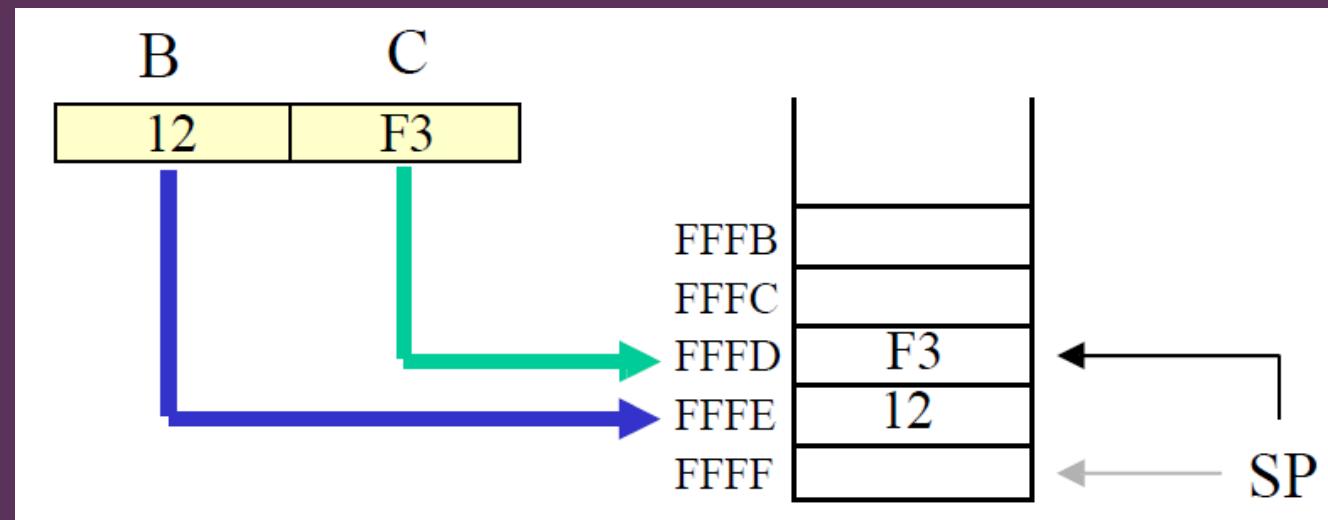
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).
- The Size of the stack is limited only by the available memory.

# Saving Information on the Stack

- The 8085 provides two instructions: **PUSH** and **POP** for storing information on the stack and retrieving it back.
  - Information is saved on the stack by **PUSHing** it on.
  - It is retrieved from the stack by **POPing** it off.
- Both **PUSH** and **POP** work with register pairs **ONLY**.

# The PUSH Instruction

- PUSH B (1 Byte Instruction)
  - Decrement SP
  - Copy the contents of register B to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of register C to the memory location pointed to by SP



# The POP Instruction

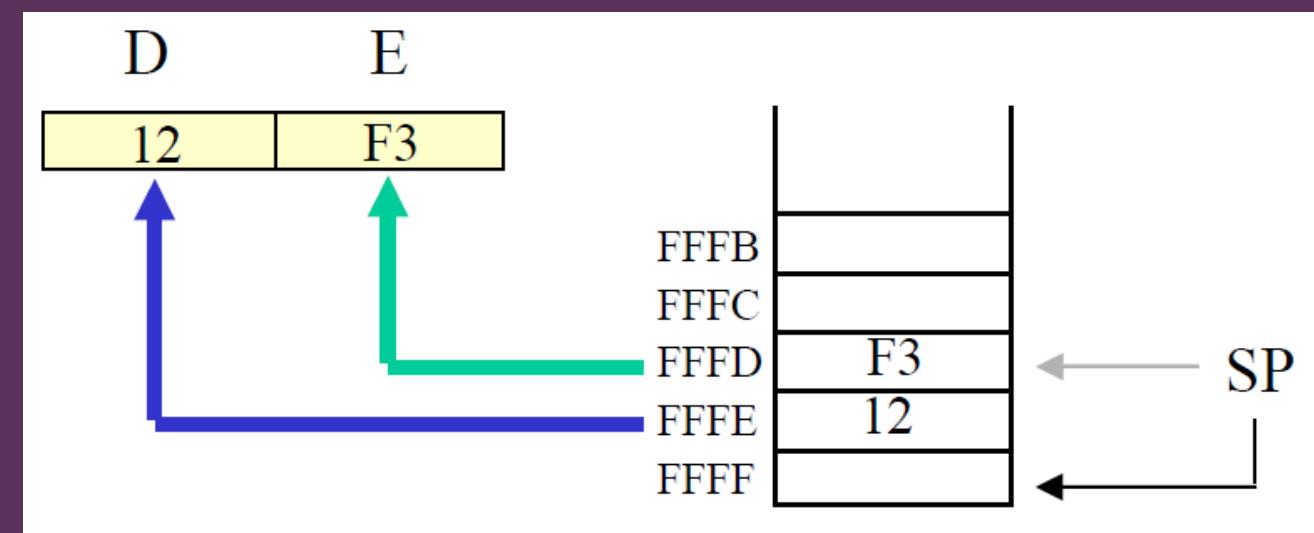
- POP D (1 Byte Instruction)

- Copy the contents of the memory location pointed to by the SP to register E

- Increment SP

- Copy the contents of the memory location pointed to by the SP to register D

- Increment SP



# Operation of the Stack

- During **pushing**, the stack operates in a “**decrement then store**” style.
  - The stack pointer is decremented first, then the information is placed on the stack.
- During **poping**, the stack operates in a “**use then increment**” style.
  - The information is retrieved from the top of the the stack and then the pointer is incremented.
- **The SP pointer always points to “the top of the stack”.**

# LIFO

- The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

PUSH B

PUSH D

...

POP D

POP B

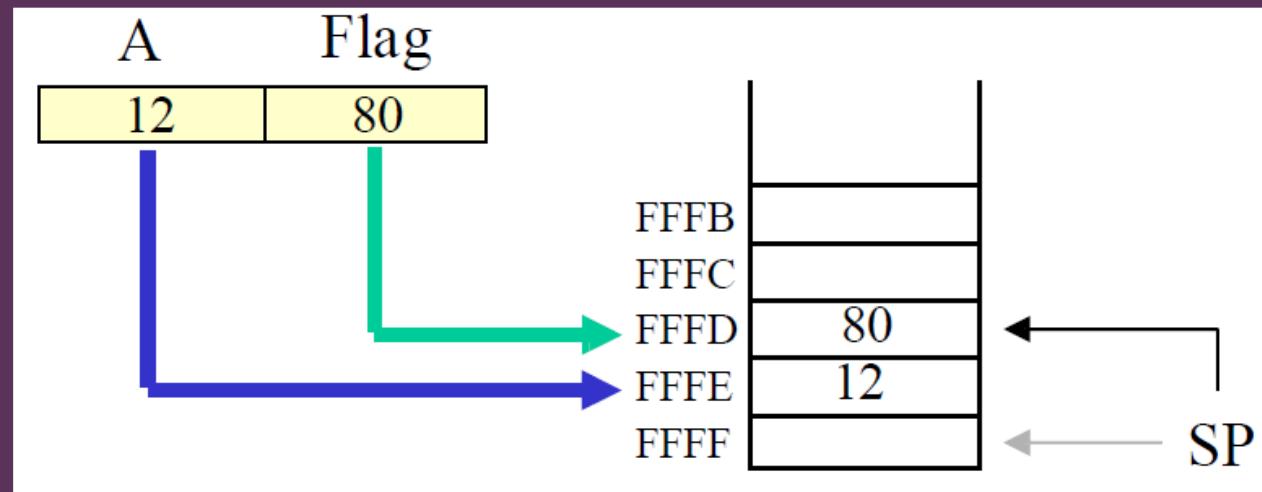
- Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE.

# The PSW Register Pair

- The 8085 recognizes **one additional register pair** called the PSW (Program Status Word).
  - This register pair is made up of the Accumulator and the Flags registers.
- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.
  - The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

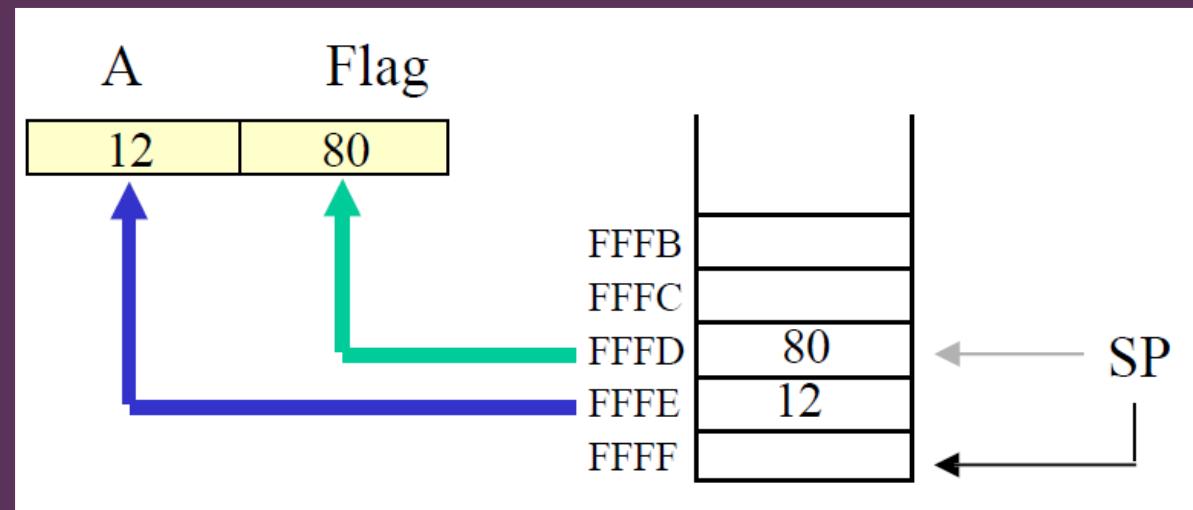
# PUSH PSW Register Pair

- PUSH PSW (1 Byte Instruction)
  - Decrement SP
  - Copy the contents of register A to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of Flag register to the memory location pointed to by SP



# Pop PSW Register Pair

- POP PSW (1 Byte Instruction)
  - Copy the contents of the memory location pointed to by the SP to Flag register
  - Increment SP
  - Copy the contents of the memory location pointed to by the SP to register A
  - Increment SP



# Modify Flag Content using PUSH/POP

- Let, We want to Reset the Zero Flag

8085 Flag

7	6	5	4	3	2	1	0
S	Z	x	AC	X	P	X	Cy

# Modify Flag Content using PUSH/POP

- Let, We want to Reset the Zero Flag

- ANI BFH

(BF H = 1011 1111) \* Masking

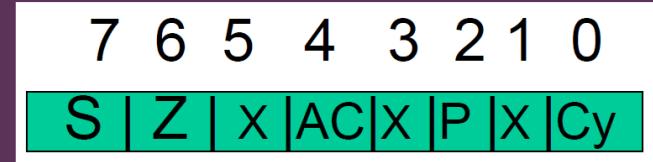
8085 Flag

7	6	5	4	3	2	1	0
S	Z	X	AC	X	P	X	Cy

# Modify Flag Content using PUSH/POP

- Let, We want to Reset the Zero Flag

# 8085 Flag



## Program:

- LXI SP FFFF
  - PUSH PSW
  - POP H
  - MOV A L
  - ANI BFH (BF H = 1011 1111) \* Masking
  - MOV L A
  - PUSH H
  - POP PSW

# Subroutines

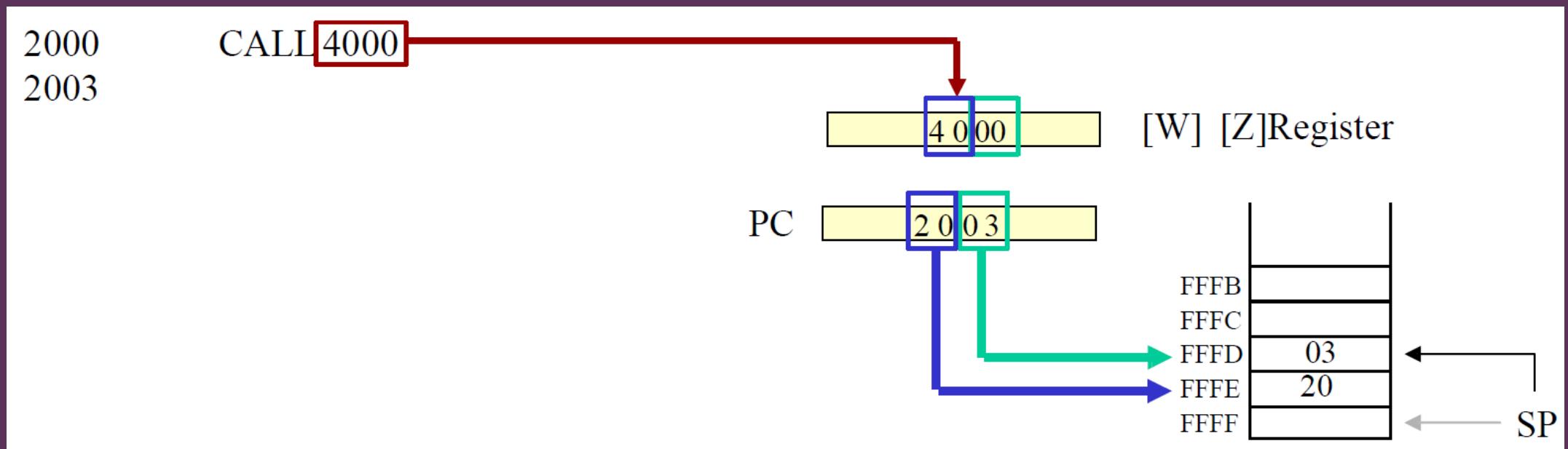
- A subroutine is a **group of instructions** that will be used repeatedly in different locations of the program
  - Rather than repeat the same instructions several times, they can be grouped into a subroutine that is **called from the different locations.**
- In Assembly language, a subroutine can exist anywhere in the code
  - However, it is customary to place subroutines separately from the main program.

# Subroutines

- The 8085 has **two instructions** for dealing with subroutines.
  - The **CALL** instruction is used to redirect program execution to the subroutine.
  - The **RET** instruction is used to return the execution to the calling routine.

# The CALL Instruction

- CALL 4000H (3 byte instruction)
  - When CALL instruction is fetched, the MP knows that the next Two Memory location contains 16bit subroutine address in the memory.

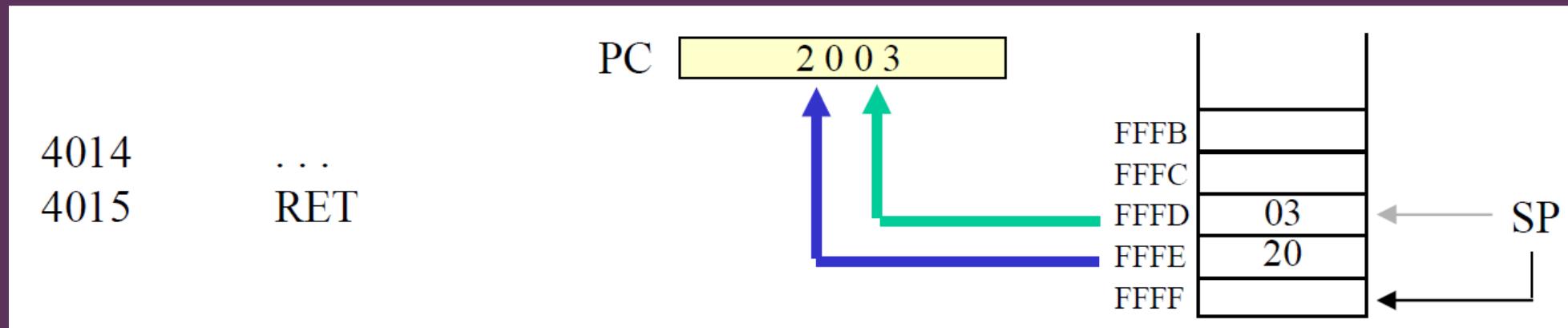


# The CALL Instruction

- MP Reads the subroutine address from the next two memory location and stores the higher order 8bit of the address in the W register and stores the lower order 8bit of the address in the Z register.
- Pushes the address of the instruction immediately following the CALL onto the stack [Return address]
- Loads the program counter with the 16 bit address supplied with the CALL instruction **from WZ register**.

# The RET Instruction

- RET (1 byte instruction)
  - Retrieve the return address from the top of the stack
  - Load the program counter with the return address.



# Things to be considered in Subroutine

- The CALL instruction places the return address at the two memory locations immediately before where the Stack Pointer is pointing.
  - You must set the SP correctly BEFORE using the CALL instruction.
- The RET instruction takes the contents of the two memory locations at the top of the stack and uses these as the return address.
  - **DO NOT MODIFY the stack pointer in a subroutine.** You will lose the return address.
  - Number of PUSH and POP instruction used in the subroutine must be same, otherwise, RET instruction will pick wrong value of the return address from the stack and program will fail.