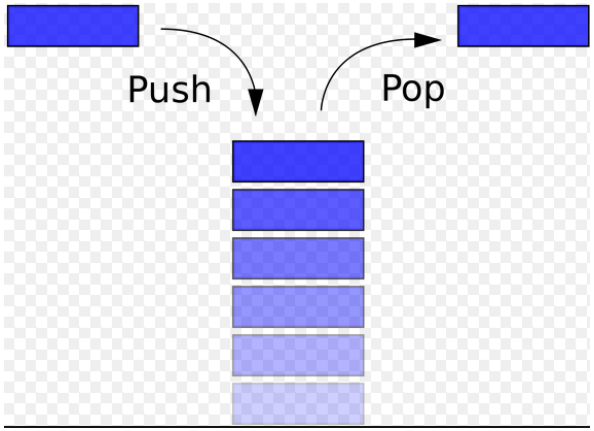
# Stack and Subroutines




## LECTURE 4

Dronacharya Group of Institutions

# The Stack

- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.

```
LXI SP, FFFFH
```

- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

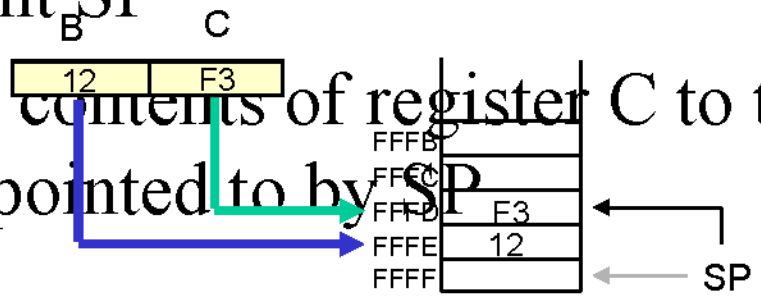
# Saving Information on the Stack

- Information is saved on the stack by PUSHing it on.
  - It is retrieved from the stack by POPing it off.

- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
  - Both PUSH and POP work with register pairs ONLY.

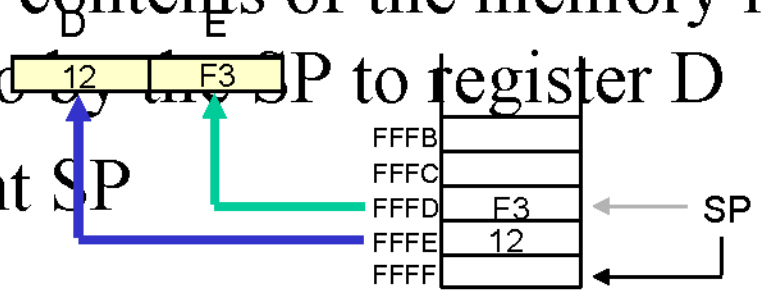
# The PUSH Instruction

- PUSH B
  - 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
  - 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
```

# 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.

# 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.

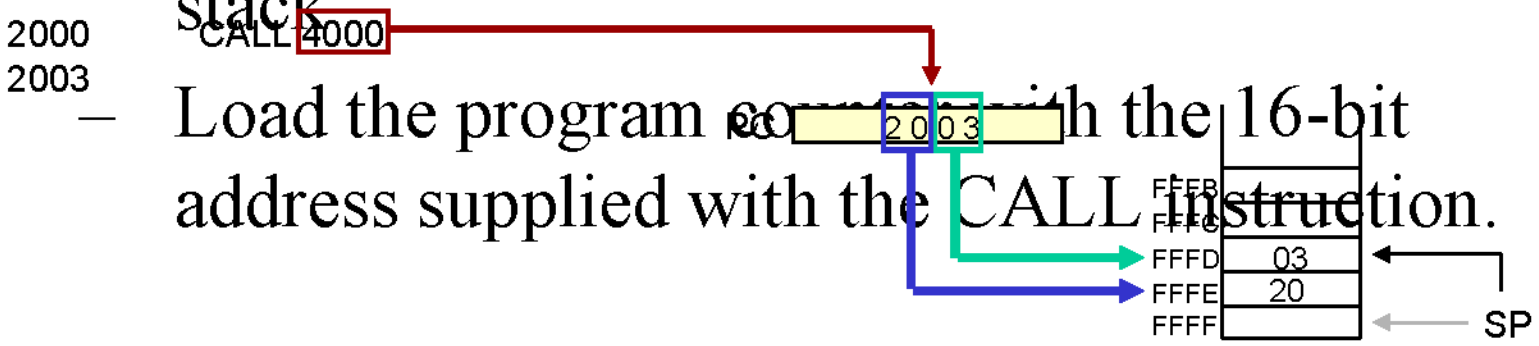
- In Assembly language, a subroutine can exist anywhere in the code.
  - However, it is customary to place subroutines separately from the main program.

# Subroutines

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

# The CALL Instruction

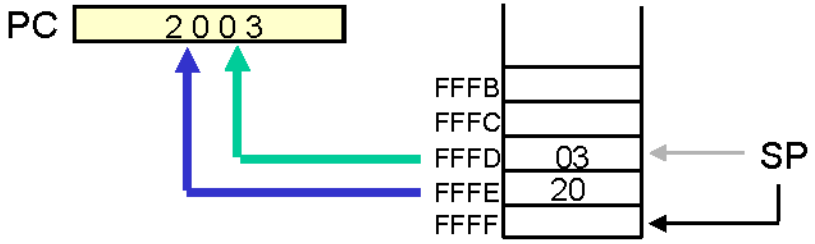
- CALL 4000H
  - Push the address of the instruction immediately following the CALL onto the stack
  - Load the program counter with the 16-bit address supplied with the CALL instruction.



# The RTE Instruction

- RTE
  - Retrieve the return address from the top of the stack
  - Load the program counter with the return address.

```
4014        . . .
4015        RTE
```



# Cautions

- 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 RTE 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 loose the return address.

# Passing Data to a Subroutine

- In Assembly Language data is passed to a subroutine through registers.
  - The data is stored in one of the registers by the calling program and the subroutine uses the value from the register.

- The other possibility is to use agreed upon memory locations.
  - The calling program stores the data in the memory location and the subroutine retrieves the data from the location and uses it.

# Call by Reference and Call by Value

- If the subroutine performs operations on the contents of the registers, then these modifications will be transferred back to the calling program upon returning from a subroutine.
  - Call by reference

- If this is not desired, the subroutine should PUSH all the registers it needs on the stack on entry and POP them on return.
  - The original values are restored before execution returns to the calling program.

# Cautions with PUSH and POP

- PUSH and POP should be used in opposite order.
- There has to be as many POP's as there are PUSH's.
  - If not, the RET statement will pick up the wrong information from the top of the stack and the program will fail.
- It is not advisable to place PUSH or POP inside a loop.

# Conditional CALL and RTE Instructions

- The 8085 supports conditional CALL and conditional RTE instructions.
  - The same conditions used with conditional JUMP instructions can be used.

  - CC, call subroutine if Carry flag is set.
  - CNC, call subroutine if Carry flag is not set
  - RC, return from subroutine if Carry flag is set
  - RNC, return from subroutine if Carry flag is not set
  - Etc.

# A Proper Subroutine

- According to Software Engineering practices, a proper subroutine:
  - Is only entered with a CALL and exited with an RTE
  - Has a single entry point
    - Do not use a CALL statement to jump into different points of the same subroutine.
  - Has a single exit point
    - There should be one return statement from any subroutine.
- Following these rules, there should not be any confusion with PUSH and POP usage.

# Advance subroutine Concepts

| | | |
|---|---|---|
| NOP | none | No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP |
| Halt and enter wait state<br>HLT | none | The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT |
| Disable interrupts<br>DI | none | The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI |
| Enable interrupts<br>EI | none | The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI |

# Advance subroutine Concepts

- A subroutine calling one subroutine and that calling another before the Return.
- For example, Main program calls subroutine B and Subroutine B calls subroutine A before return.
- Since PC always saves at top of the stack the return is always to the calling routine.