1. Stack

- Stack is a group of memory location in the R/W memory that is used for temporary storage of binary information during execution of a program.
- The starting memory location of the stack is defined in program and space is reserved usually at the high end of memory map.
- The beginning of the stack is defined in the program by using instruction **LXI SP, 16-bit memory address**. Which loads a 16-bit memory address in stack pointer register of microprocessor.
- Once stack location is defined storing of data bytes begins at the memory address that is one less then address in stack pointer register. LXI SP, 2099h the storing of data bytes begins at 2098H and continues in reversed numerical order.

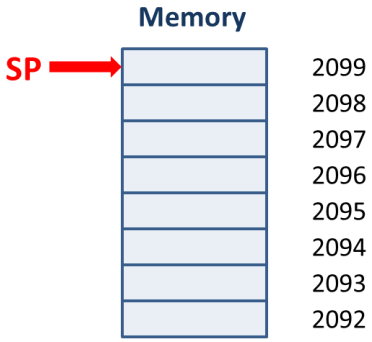

Fig. Stack

- Data bytes in register pair of microprocessor can be stored on the stack in reverse order by using the PUSH instruction.
- PUSH B instruction sore data of register pair BC on sack.

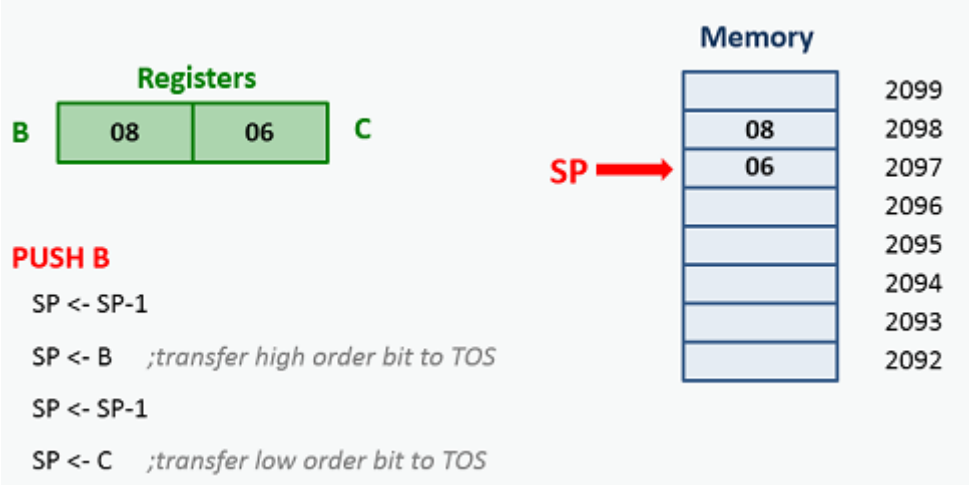

Fig. PUSH operation on stack

- Data bytes can be transferred from the stack to respective registers by using instruction POP.

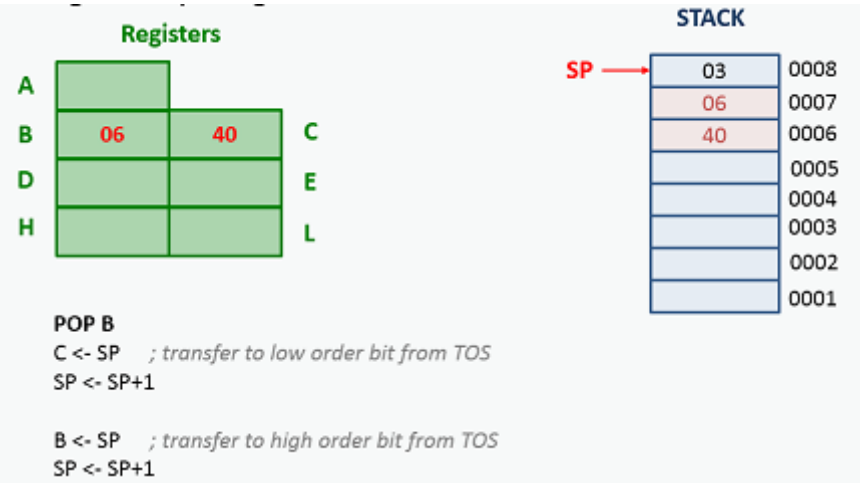

Fig. POP operation on stack

Instruction necessary for stack in 8085

LXI SP, 2095	Load the stack pointer register with a 16-bit address.
PUSH B/D/H	It copies contents of B-C/D-E/H-L register pair on the stack.
PUSH PSW	Operand PSW represents Program status word meaning contents of accumulator and flags.
POP B/D/H	It copies content of top two memory locations of the stack in to specified register pair.
POP PSW	It copies content of top two memory locations of the stack in to B-C accumulator and flags respectively.

2. Subroutine

- A subroutine is a group of instruction that performs a subtask of repeated occurrence.
- A subroutine can be used repeatedly in different locations of the program.

Advantage of using Subroutine

- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

Where to write Subroutine?

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

Instructions for dealing with subroutines in 8085.

- The **CALL** instruction is used to redirect program execution to the subroutine.
  - When CALL instruction is fetched, the Microprocessor knows that the next two **new** Memory location contains 16bit subroutine address.
  - Microprocessor 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.
  - Push the **Older** address of the instruction immediately following the CALL onto the stack [Return address]
  - Loads the program counter (**PC**) with the **new** 16-bit address supplied with the CALL instruction from **WZ** register.
- The **RET** instruction is used to return.

- 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.

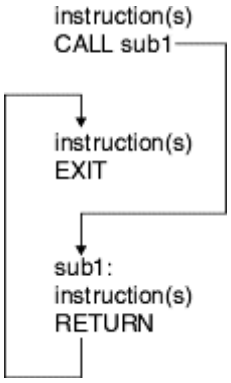

Fig. Subroutine

- Example: write ALP to add two numbers using call and subroutine.

```
LXI H 2000 ; Load memory address of operand
MOV B M ; Store first operand in register B
INX H ;Increment H-L pair
MOV A M ; Store second operand in register A
CALL ADDITION ; Call subroutine ADDITION
STA 3000 ; Store answer
HLT

ADDITION: ADD B ; Add A and B
RET ; Return
```

Conditional call and return instruction available in 8085

CC 16-bit address	Call on Carry, Flag Status: CY=1
CNC 16-bit address	Call on no Carry, Flag Status: CY=0
CP 16-bit address	Call on positive, Flag Status: S=0
CM 16-bit address	Call on minus, Flag Status: S=1
CZ 16-bit address	Call on zero, Flag Status: Z=1
CNZ 16-bit address	Call on no zero, Flag Status: Z=0
CPE 16-bit address	Call on parity even, Flag Status: P=1
CPO 16-bit address	Call on parity odd, Flag Status: P=0
RC	Return on Carry, Flag Status: CY=1
RNC	Return on no Carry, Flag Status: CY=0
RP	Return on positive, Flag Status: S=0
RM	Return on minus, Flag Status: S=1
RZ	Return on zero, Flag Status: Z=1
RNZ	Return on no zero, Flag Status: Z=0
RPE	Return on parity even, Flag Status: P=1
RPO	Return on parity odd, Flag Status: P=0