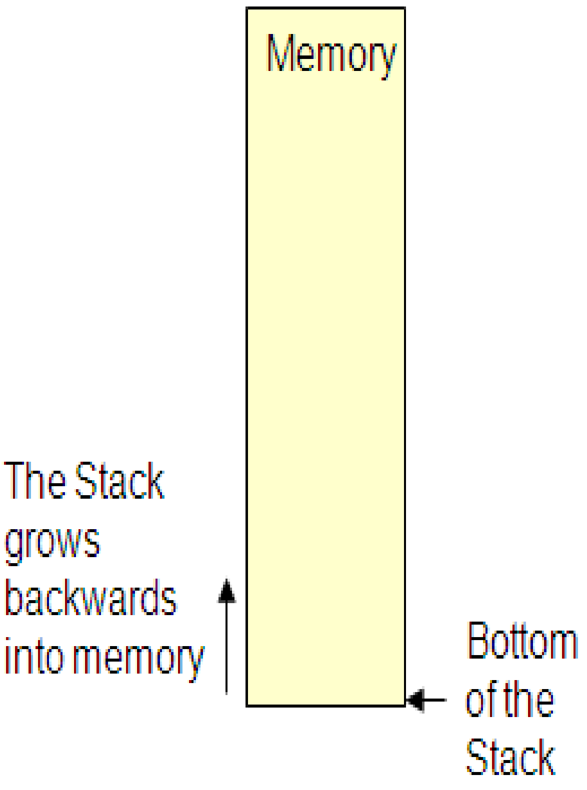
Chapter 9

Stack & Subroutines

The Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO (Last In First Out.) structure.
- 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

- 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 Rp (1 byte instruction)

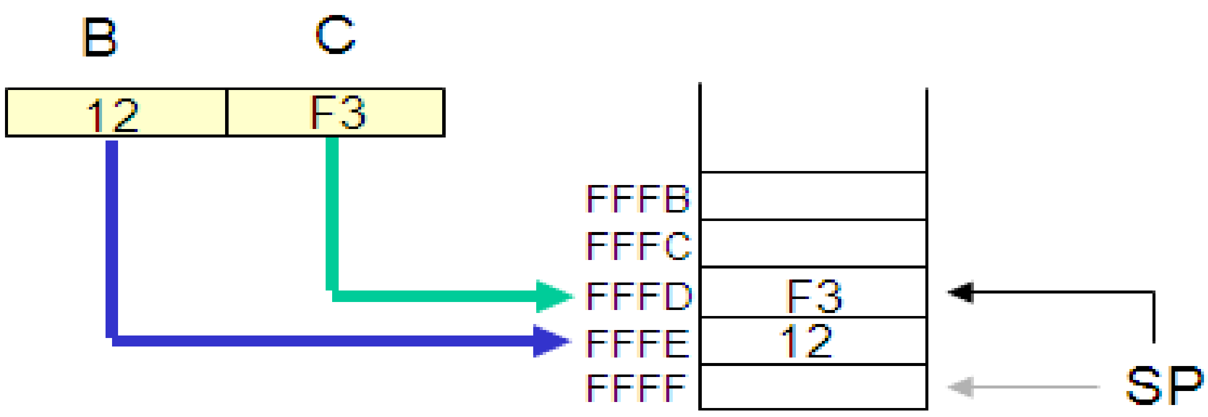
- It copies the contents of the specified register pair on the stack.

- The stack pointer (SP) register is decremented, and the contents of the high-order register are copied in the location shown by the SP.

- The stack pointer (SP) register is again decremented, and the contents of the low-order register are copied in the location shown by the SP.

The PUSH Instruction

- PUSH B/D/H/PSW
  - 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 Rp (1 byte instruction)

➢ It copies the contents of the top two memory locations of the stack into the specified register pair.

➢ The contents of the memory location indicated by the SP register are copied into the low-order register, and then the SP register is incremented by 1.

➢ The contents of the next memory location are copied into the high-order register, and the SP register is again incremented by 1.

The POP Instruction

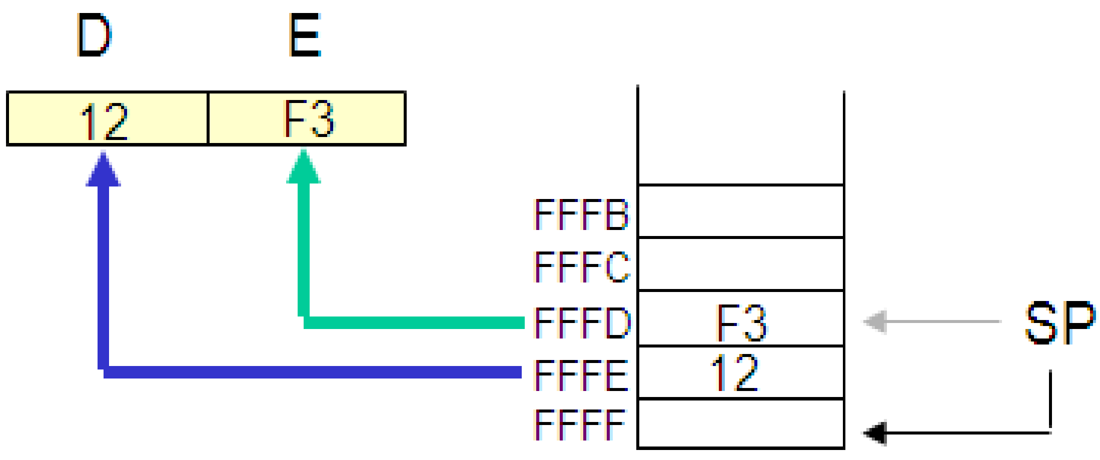
- POP B/D/H/PSW

➢ 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
```

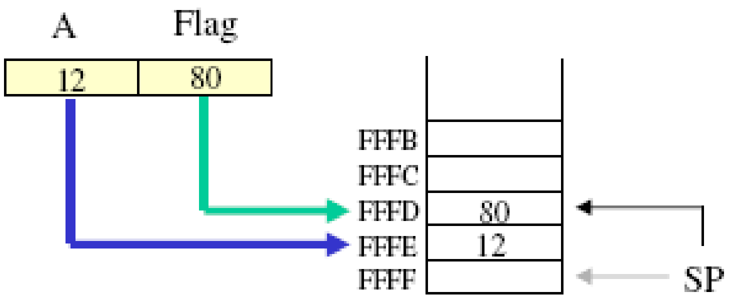
- **Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE.**

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.

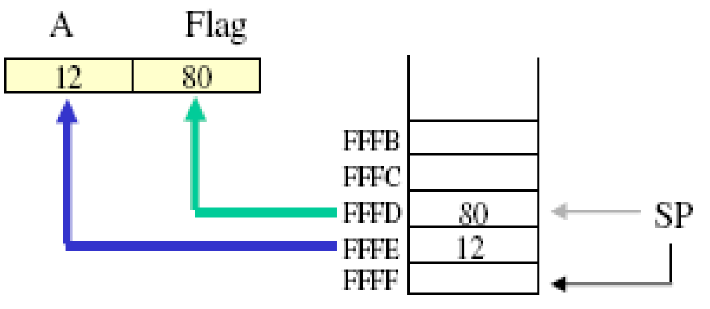
PUSH PSW Register Pair

- 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



Example 9.2

- The available user memory ranges from 2000H to 23FFH. A program of data transfer and arithmetic operations is stored in memory locations from 2000H to 2050H and the stack pointer is initialized at location 2400H. Two sets of data are stored, starting at locations 2150H and 2280H. Register HL and BC are used as memory pointers to the data locations.
  1. Explain how the stack pointer can be initialized at one memory location beyond the available user memory.
  2. Illustrate the contents of the stack memory and register when PUSH and POP instructions are executed, and explain how memory pointers are exchanged.
  3. Explain the various contents of the user memory.

Ans: See book page no – 299.

Problem -1

Write a program to perform the following functions:

- Clear all Flags.
- Load 00H in the accumulator, and demonstrate that the zero flag is not affected by data transfer instruction.
- Logically OR the accumulator with itself to set the Zero flag, and display the flag at PORT1 or store all flags on the stack.

Program to Reset and display Flags

• XX00 LXI SP, XX99H	Initialize the stack
• 03 MVI L, 00H	Clear L
• 05 PUSH H	Place (L) on stack
• 06 POP PSW	Clear Flags
• 07 MVI A, 00H	Load 00H
• 09 PUSH PSW	Save Flags on stack
• 0A POP H	Retrieve flags in L
• 0B MOV A, L	
• 0C OUT PORT0	Display Flags (00H)
• 0E MVI A, 00H	Load 00H Again

Program to Reset and display Flags

• XX10 ORA A	Set Flags and reset CY, AC
• 11 PUSH PSW	Save Flags on Stack
• 12 POP H	Retrieve Flags in L
• 13 MOV A, L	
• 14 ANI 40H	Mask all Flags except Z
• 16 OUT PORT1	Displays 40H
• 18 HLT	End of Program

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

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

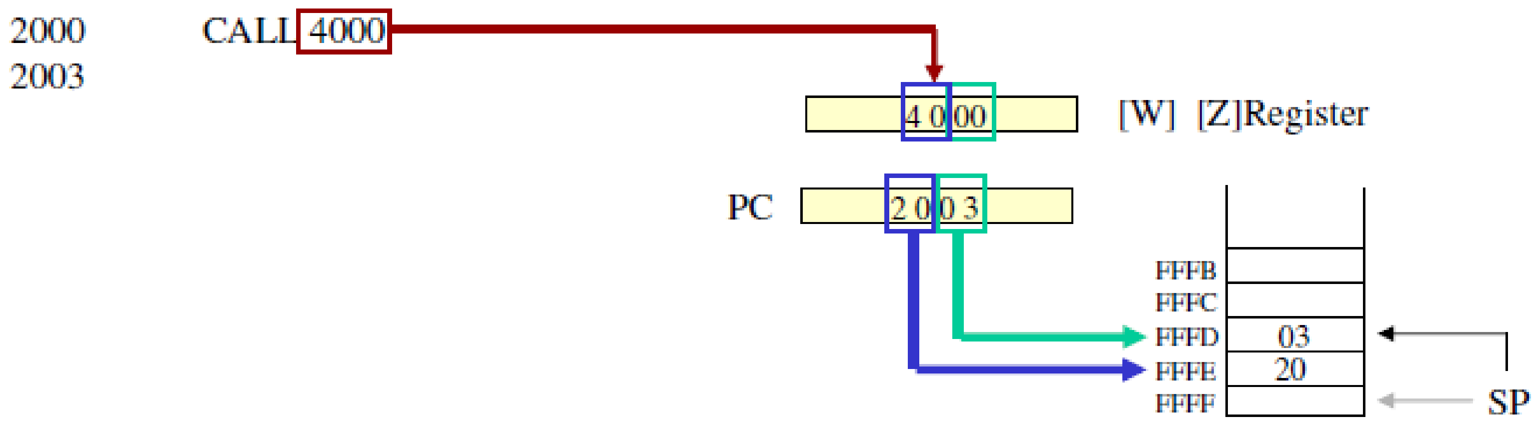
- The 8085 has two instructions for dealing with subroutines.

- The **CALL** instruction is used to redirect program execution to the subroutine.

- The **RTE** instruction is used to return the execution to the calling routine.

The CALL Instruction

- **CALL 4000H**
  - 3-byte instruction.
  - Push the address of the instruction immediately following the CALL onto the stack and decrement the stack pointer register by two.
  - Load the program counter with the 16-bit address supplied with the CALL instruction.
  - Jump Unconditionally to memory location.

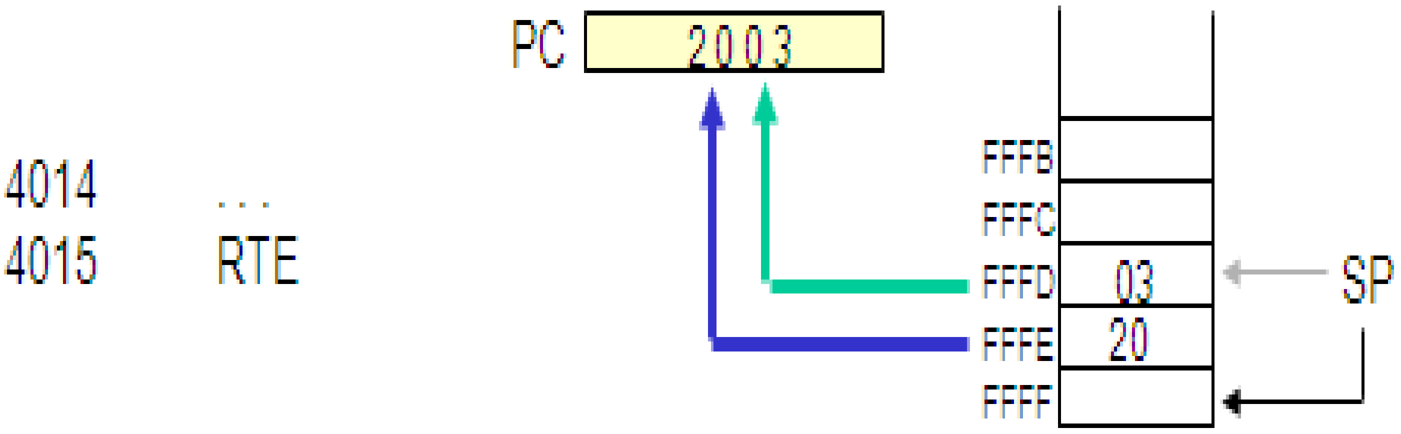


The CALL Instruction

- 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
- Push 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 RTE Instruction

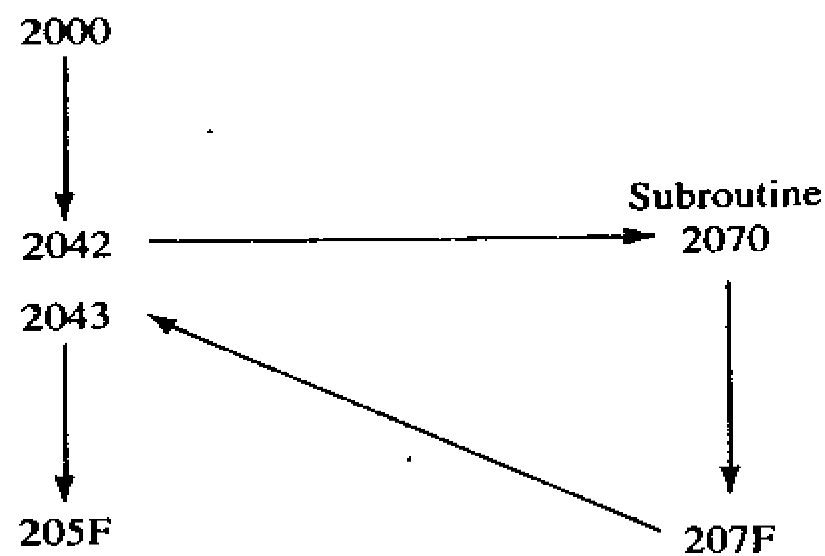
- **RTE**
  - 1-byte instruction
  - Retrieve the return address from the top of the stack and increments stack pointer register by two.
  - Load the program counter with the return address.
  - Unconditionally returns from a subroutine.



Illustrates the exchange of information between stack and Program Counter

Memory Address	
2000	LXI SP,2400H
↓	↓
2040	CALL 2070H
2041	
2042	
2043	NEXT INSTRUCTION
↓	↓
205F	HLT
2070	First Subroutine Instruction
↓	↓
207F	RET
2080	↓
↓	Other Subroutines
	↓
2398	Empty Space
23FF	↓
2400	

Program Execution



Memory Address	Machine Code	Mnemonics	Comments
2040	CD	CALL 2070H	;Call subroutine located at the memory
2041	70		; location 2070H
2042	20		
2043	NEXT	INSTRUCTION	

CALL Execution

- **Instruction requires five machine cycles and eighteen T-states: Call instruction is fetched, 16-bit address is read during M2 and M3 and stored temporarily in W/Z registers. In next two cycles content of program counter are stored on the stack (address from where microprocessor continue it execution of program after co**

Instruction: CALL 2070H

Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand	70
M_3 Memory Read	23FF	2042	20 43	20 Operand	20
M_4 Memory Write	23FE (SP–2)	23FF	20 43	20 (PCH)	
M_5 Memory Write	23FE	23FE	20 43	43 (PCL)	(20) (70)
M_1 Opcode Fetch of Next Instruction		2070 (W)(Z)	2071		(2070) (W)(Z)

Memory Address	Code (H)
2040	CD
2041	70
2042	20

Data Transfer During the Execution of the CALL Instruction

RET Execution

- Program execution sequence is transferred to the memory location 2043H location.M1 is normal fetch cycle during M2 contents of stack pointer are placed on address bus so 43H data is fetched and stored on Z register and SP is upgraded. Similarly for M3. Program sequence is transfered to2043H by placing contents of

Memory Address	Code (H)
207F	C9

Contents of Stack Memory	
23FE	43
23FF	20

Machine Cycles	Stack Pointer (23FE)	Address Bus (AB)	Program Counter	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FE	207F	2080	C9 Opcode	
M_2 Memory Read	23FF	23FE		43 (Stack)	43
M_3 Memory Read	2400	23FF		20 (Stack−1)	20
M_1 Opcode Fetch of Next Instruction		2043 (W) (Z)	2044		2043 (W) (Z)

Data Transfer During the Execution of the RET Instruction

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.

Problem Statement

Write a program to provide the given on/off time to three traffic lights (Green, Yellow, and Red) and two pedestrian signs (WALK and DON'T WALK). The signal lights and signs are turned on/off by the data bits of an output port as shown below:

Lights	Data Bits	On Time
1. Green	D_0	15 seconds
2. Yellow	D_2	5 seconds
3. Red	D_4	20 seconds
4. WALK	D_6	15 seconds
5. DON'T WALK	D_7	25 seconds

Problem Statement

Time Sequence in Seconds	DON'T WALK	WALK		Red		Yellow		Green		Hex Code
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
0										
(15) ↓ 15	0	1	0	0	0	0	0	1	=	41H
(5) ↓ 20	1	0	0	0	0	1	0	0	=	84H
(20) ↓ 40	↓ 1	0	0	1	0	0	0	0	=	90H

Problem Statement

	Mnemonics	Comments
	LXI SP,XX99	;Initialize stack pointer at location XX99H
		;High-order address (page) of user memory
START:	MVI A,41H	;Load accumulator with the bit pattern for
		; Green light and WALK sign
	OUT PORT#	;Turn on Green light and WALK sign

Problem Statement

```
MVI B,0FH          ;Use B as a counter to count 15seconds.
                   ;   B is decremented in the subroutine
CALL DELAY         ;Call delay subroutine located at XX50H

                   ;High-order address (page) of user memory
MVI A,84H          ;Load accumulator with the bit pattern for
                   ;   Yellow light and DON'T WALK
OUT PORT#          ;Turn on Yellow light and DON'T WALK
                   ;   and turn off Green light and WALK
MVI B,05           ;Set up 5-second delay counter

CALL DELAY

                   ;High-order address of user memory
MVI A,90H          ;Load accumulator with the bit pattern for
                   ;   Red light and DON'T WALK
OUT PORT#          ;Turn on Red light, keep DON'T WALK on,
                   ;   and turn off Yellow light
MVI B,14H          ;Set up the counter for 20-second delay

CALL DELAY


JMP START          ;Go back to location START to repeat the
                   ;   sequence
```

Problem Statement

```
DELAY:   PUSH D          ;Save contents of DE and accumulator
         PUSH PSW
SECOND:  LXI D,COUNT     ;Load register pair DE with a count for
                         ;   1-second delay

Loop:    DCX D           ;Decrement register pair DE
         MOV A,D
         ORA E           ;OR (D) and (E) to set Zero flag
         JNZ LOOP        ;Jump to Loop if delay count is not equal to 0
         DCR B           ;End of 1 second delay; decrement the counter
         JNZ SECOND      ;Is this the end of time needed? If not, go
                         ;   back to repeat 1-second delay
                         ;High-order memory address of user memory
         POP PSW         ;Retrieve contents of saved registers
         POP D
         RET             ;Return to main program
```