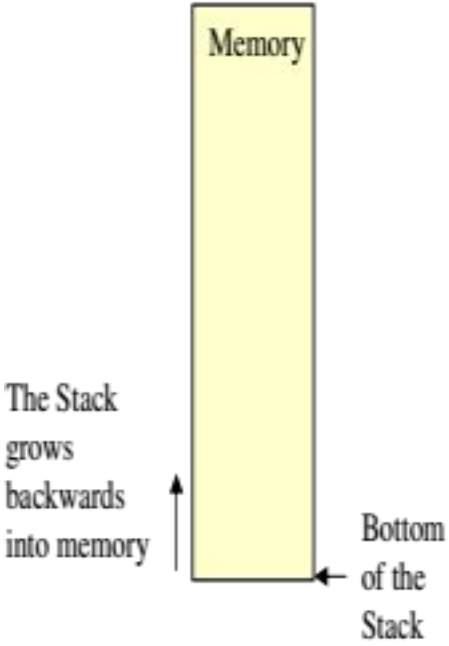
Unit 3
Stack and Subroutines

Stack

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



Stack ..

In 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)

- The size of the stack is limited only by the available memory.
- The 8085 provides two instructions : PUSH and POP for storing information on the stack and retrieving it back. Both work with register pairs only.
- Data bytes in the register pairs of the microprocessor can be stored on the stack (two at a time) in reverse order (decreasing memory address) by using the instruction PUSH.
- Data bytes can be transferred from the stack to respective registers using the instruction POP.
- Because two data bytes are being stored at a time, the 16-bit memory address in the stack pointer register is decremented by two; when data bytes are retrieved, the address is incremented by two.

PUSH instruction

The stack is used by both programmer and microprocessor. The programmer can use register pair to push and pop data to and from stack. The microprocessor uses stack during subroutines to store the content of program counter.

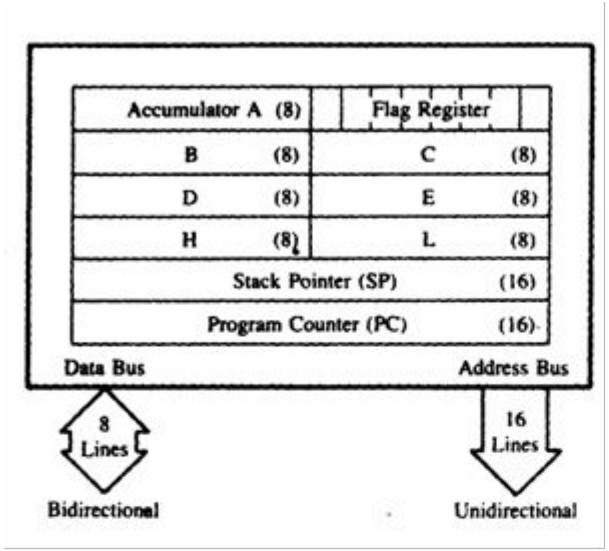
Instructions

Opcode	Operand
LXI	SP, 16-bit
PUSH	Rp

PUSH B
PUSH D
PUSH H
PUSH PSW

POP Rp

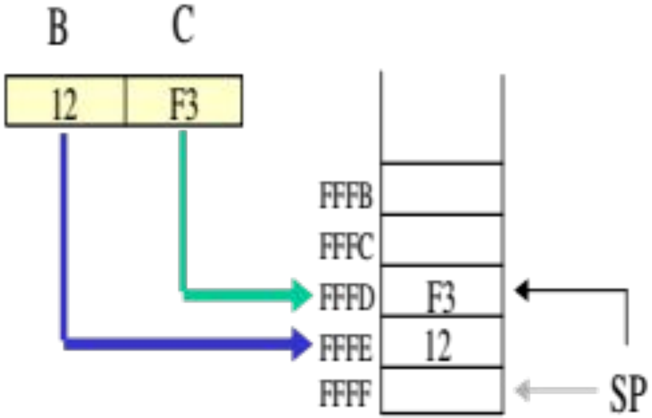
POP B
POP D
POP H
POP PSW



PUSH instruction

PUSH B (1 byte instruction)

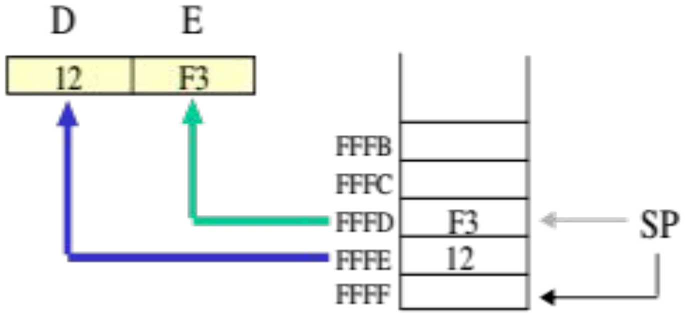
- Decrement SP
- Copy the contents of register B to the memory location pointed to by SP
- Decrement SP
- Copy the contents of register C to the memory location pointed to by SP



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 memory location 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 memory location” 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”

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.

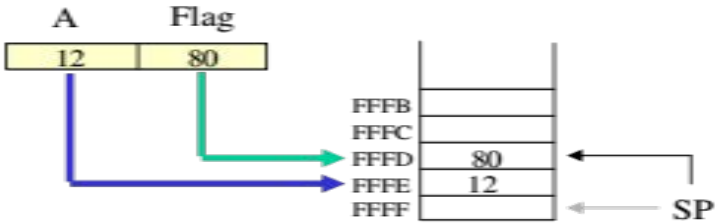
- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.

  – 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 execute

PUSH PSW register pair

PUSH PSW (1 Byte Instruction)

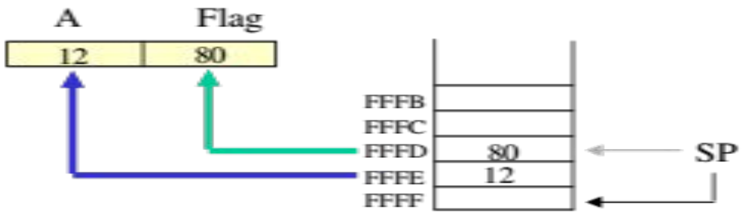
- Decrement SP
- Copy the contents of register A to the memory location pointed to by SP
- Decrement SP
- Copy the contents of Flag register to the memory location pointed to by SP



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



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

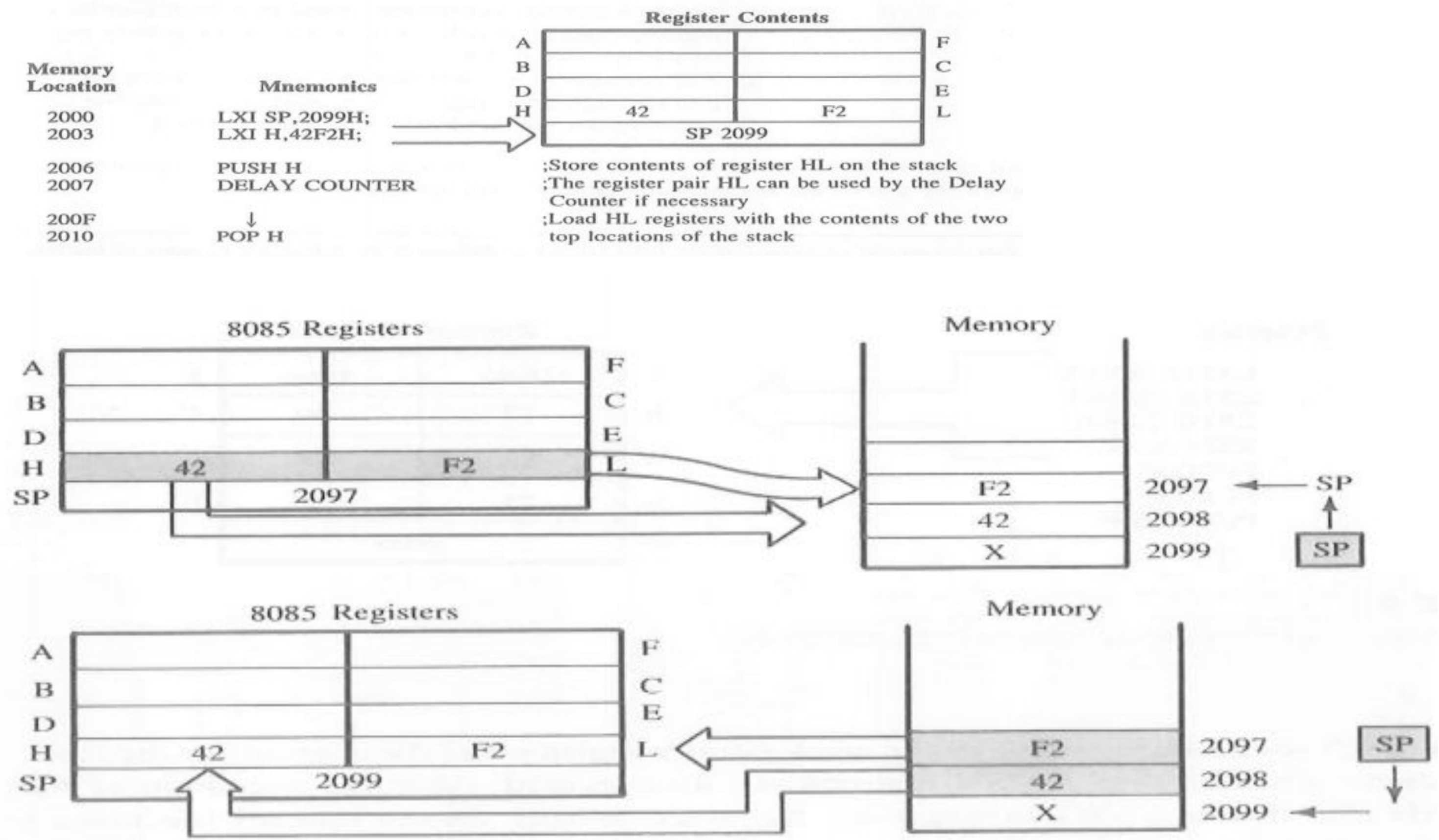
;

POP D

POP B

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

Example 1 - PUSH

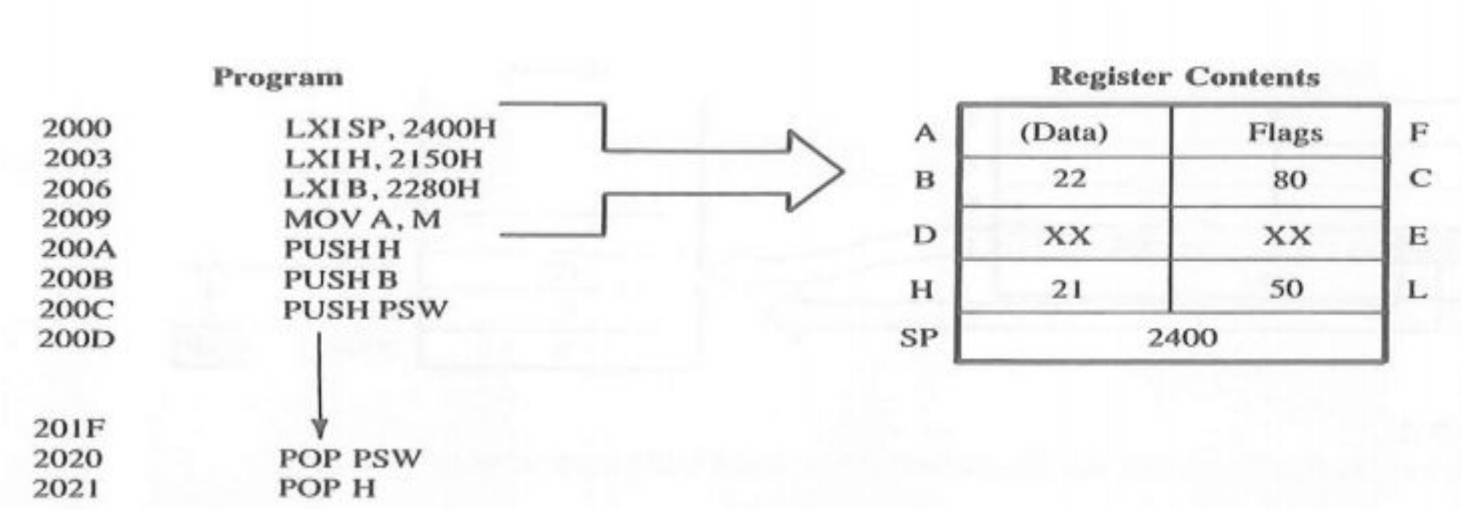


```
Memory
Location      Mnemonics

 2000         LXI SP,2099H;
 2003         LXI H,42F2H;

 2006         PUSH H               ;Store contents of register HL on the stack
 2007         DELAY COUNTER        ;The register pair HL can be used by the Delay
                                    Counter if necessary
 200F              ↓               ;Load HL registers with the contents of the two
 2010         POP H                 top locations of the stack
```

Register Contents

A			F
B			C
D			E
H	42	F2	L
	SP 2099		

Example 2

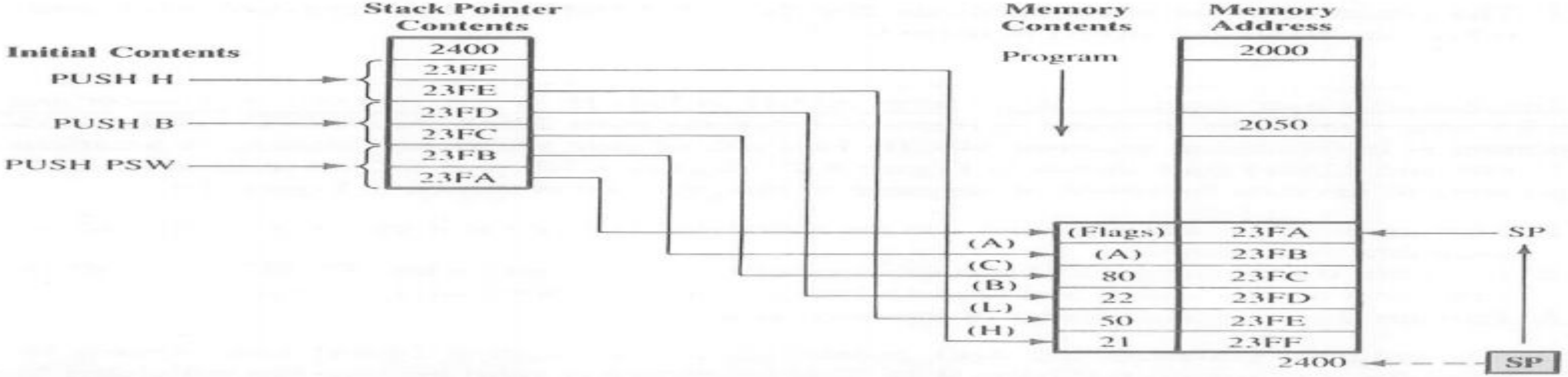


Program

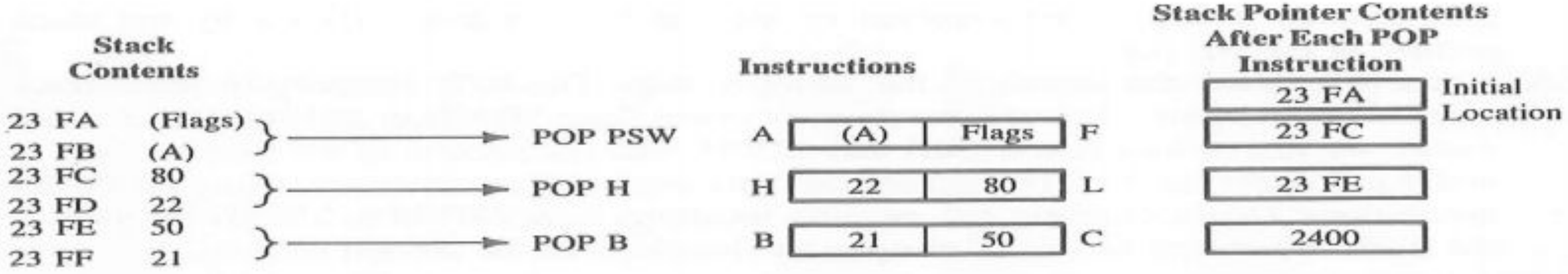
```
2000    LXI SP, 2400H
2003    LXI H, 2150H
2006    LXI B, 2280H
2009    MOV A, M
200A    PUSH H
200B    PUSH B
200C    PUSH PSW
200D

201F
2020    POP PSW
2021    POP H
```

Register Contents

A	(Data)	Flags	F
B	22	80	C
D	XX	XX	E
H	21	50	L
SP	2400		



Stack contents after execution of PUSH instructions

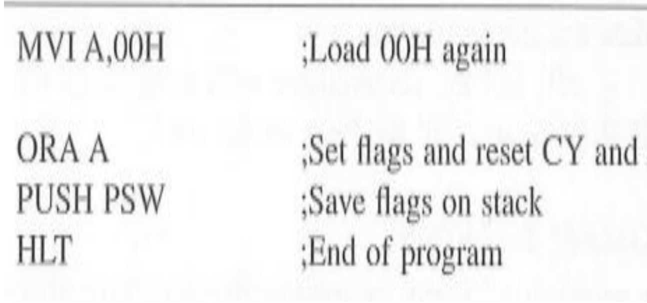


Stack contents after execution of POP instructions

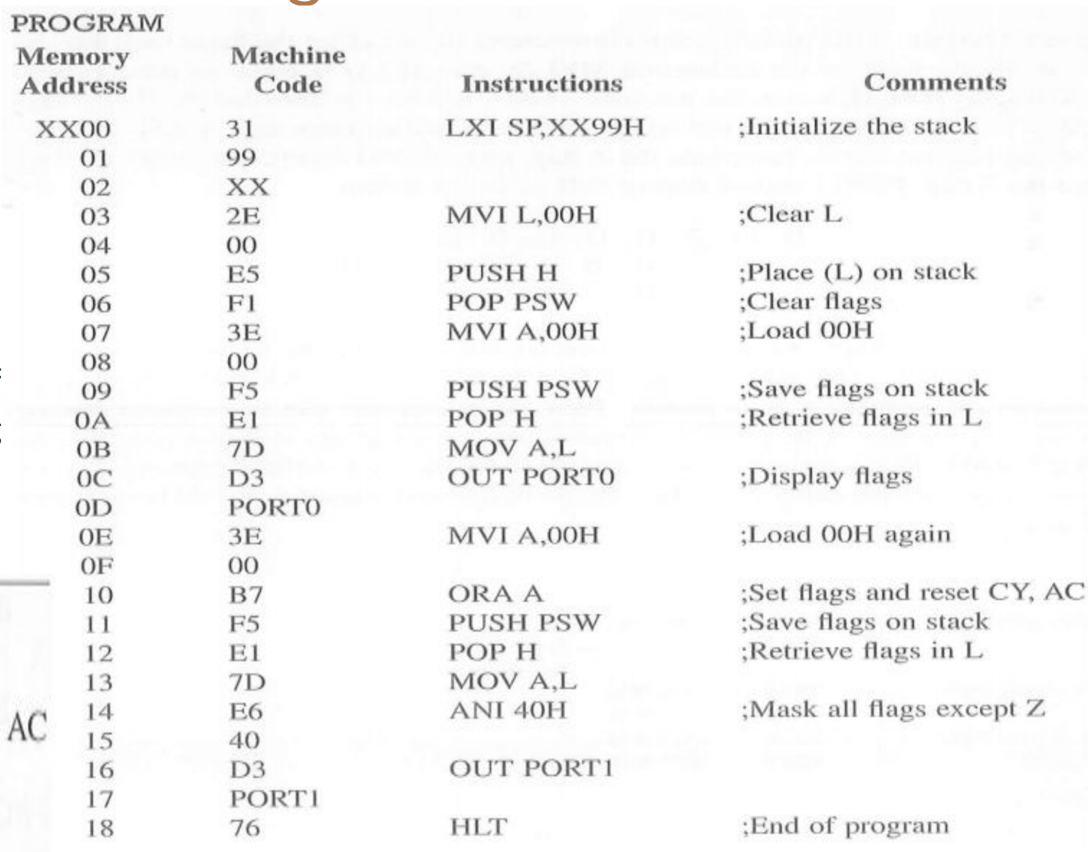
Modify Flag content using PUSH / POP

Problem 1

1. Clear all flags
2. Load 00H in the accumulator, and demonstrate that the zero flag is not affected by the data transfer instruction.
3. Logically OR the accumulator with itself to set the Zero flag, and display the flag at PORT1 or store all the flags on the stack 00000000 01000100 00 44



```
MVI A,00H        ;Load 00H again

ORA A            ;Set flags and reset CY and AC
PUSH PSW         ;Save flags on stack
HLT              ;End of program
```



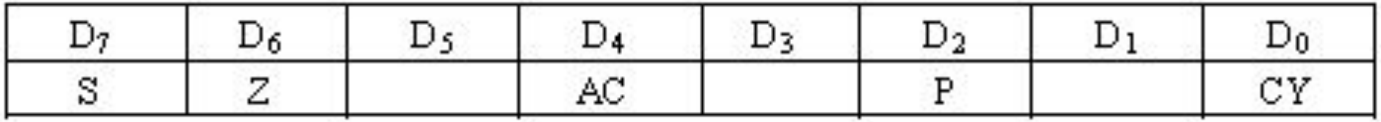
PROGRAM

Memory Address	Machine Code	Instructions	Comments
XX00	31	LXI SP,XX99H	;Initialize the stack
01	99		
02	XX		
03	2E	MVI L,00H	;Clear L
04	00		
05	E5	PUSH H	;Place (L) on stack
06	F1	POP PSW	;Clear flags
07	3E	MVI A,00H	;Load 00H
08	00		
09	F5	PUSH PSW	;Save flags on stack
0A	E1	POP H	;Retrieve flags in L
0B	7D	MOV A,L	
0C	D3	OUT PORT0	;Display flags
0D	PORT0		
0E	3E	MVI A,00H	;Load 00H again
0F	00		
10	B7	ORA A	;Set flags and reset CY, AC
11	F5	PUSH PSW	;Save flags on stack
12	E1	POP H	;Retrieve flags in L
13	7D	MOV A,L	
14	E6	ANI 40H	;Mask all flags except Z
15	40		
16	D3	OUT PORT1	
17	PORT1		
18	76	HLT	;End of program

Flag bits

0100 0100

0001 0000



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

A= 0100 0100

40H=0100 0000

0100 0000

Subroutine

A subroutine is a group of instructions separate from main program that will be called repeatedly in different locations of the program to perform a function..

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.

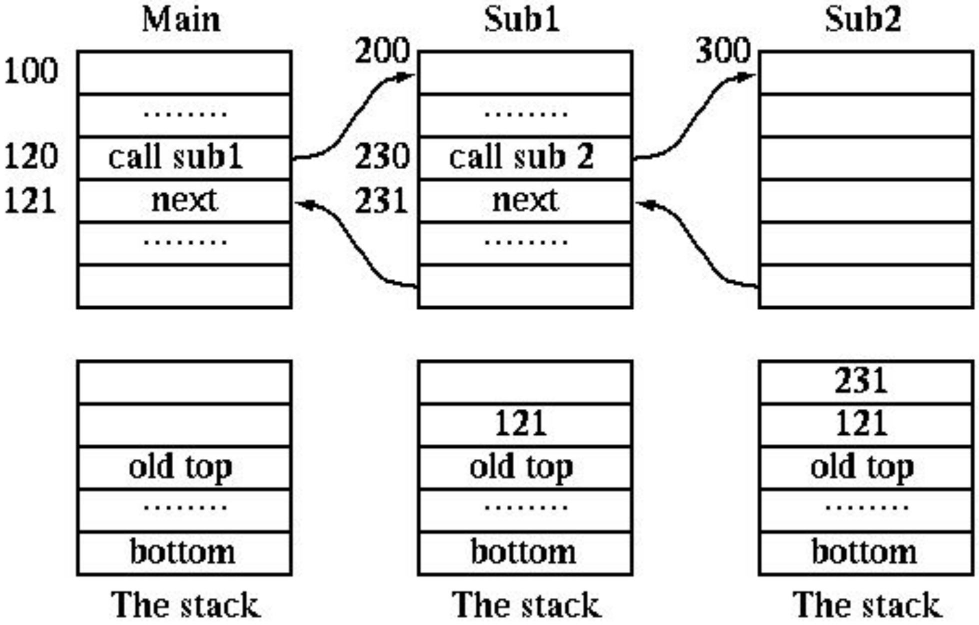
MAIN PROGRAM

78H	CALL
79H	00
80H	12
81H	
200H	CALL
201H	00
202H	12

SUBROUTINE

1200H	
	RET

Example1



(1) **Jump to function:**

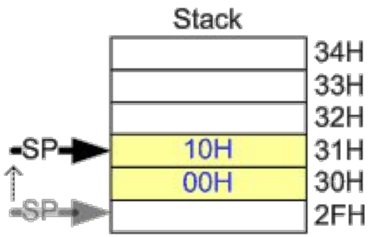
Two things need to be done when jumping from the invoking routine, say MAIN, to the invoked routine, say SUB:

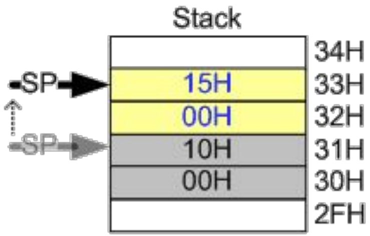
- Push updated PC content on a stack;
- Load PC with the starting address of SUB, the address of the first instruction in SUB.

(2) **Return from function:**

Pop the top item on stack, the return address, to PC.

Example2





MAIN PRG

0FFDH	CALL 14F0
1000H	

SUBROUTINE A

14FOH	
14FDH	CALL 20FAH
1500H	
	RET

SUBROUTINE B

20FAH	
	RET

Subroutines cont...

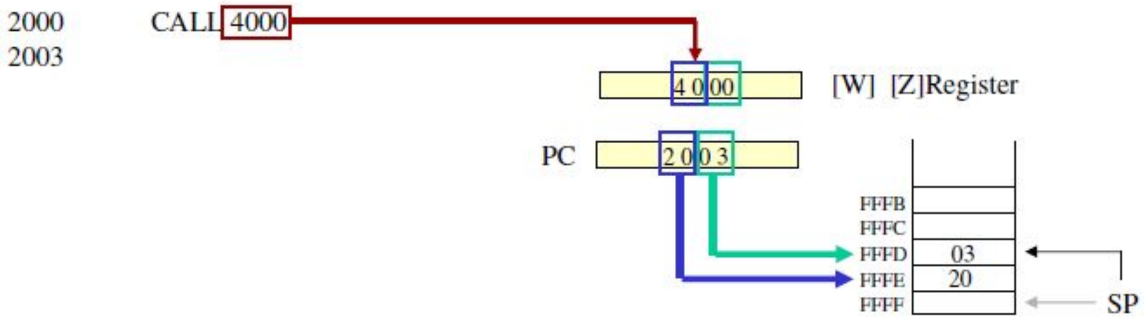
8085 microprocessor has two instruction for subroutines

CALL 16 bit address - to redirect program execution to the subroutine (3 byte instruction)

RET - to return to the calling routine (one byte instruction)

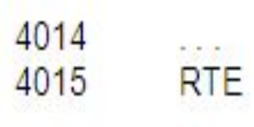
CALL instruction

- CALL 4000H
- 3 byte instruction, 5 machine cycles, 18 T states
- Push the address of the instruction immediately following the CALL onto the stack and decrement the stack pointer register by two.
- Jump Unconditionally to memory location given next to CALL.
- Load the program counter with the 16-bit address supplied with the CALL instruction.
- Microprocessor reads the subroutine address from the next two memory location and stores the higher order 8-bit of the address in the W register and stores the lower order 8-bit of the address in the Z register.

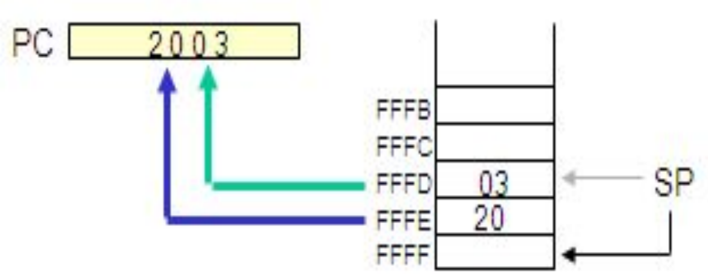


RET instruction

- 1-byte instruction, 3 machine cycle and 10 T-states
- 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.

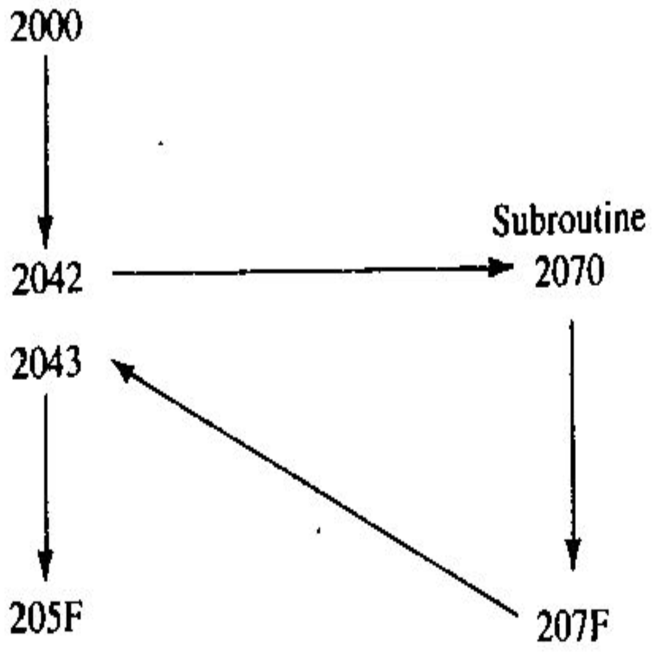


```
4014        ...
4015        RTE
```



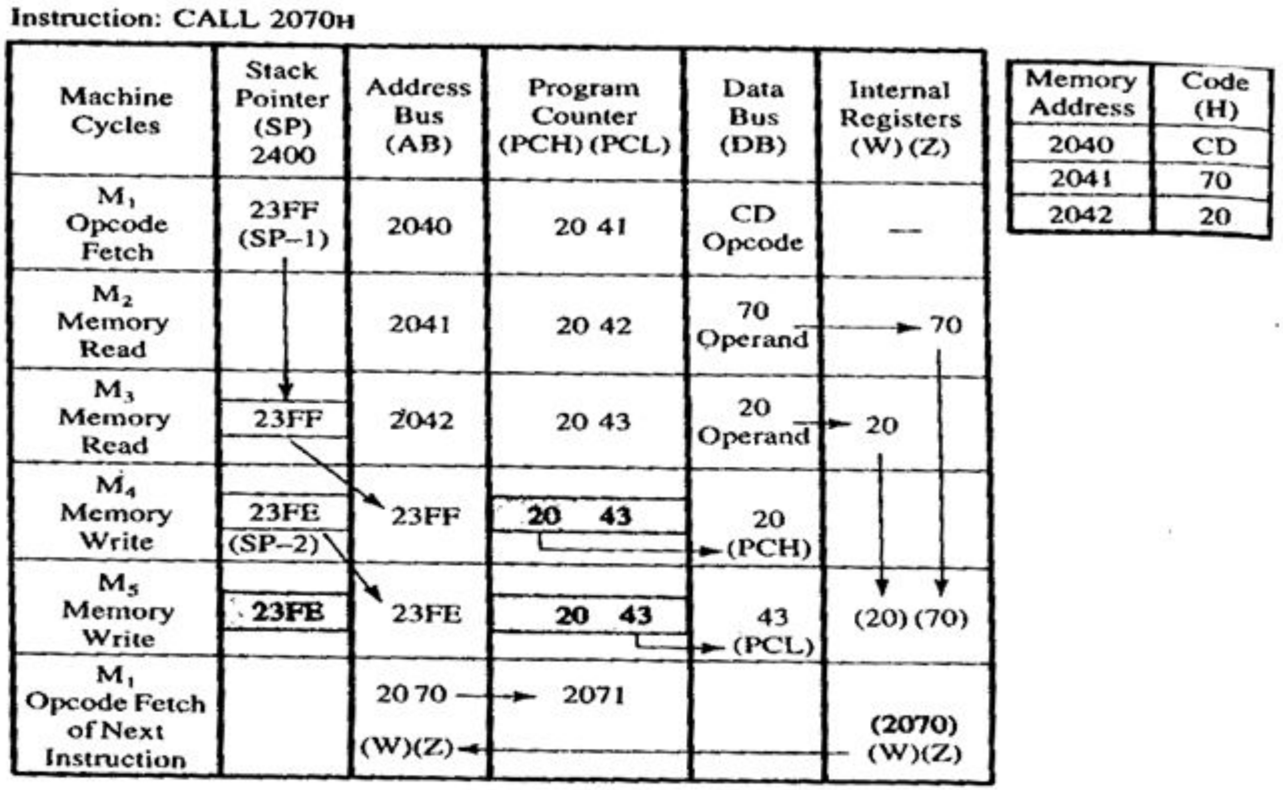
Illustrates the exchange of information between stack and Program Counter

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	



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 the execution of program after completion of the subroutine.



Instruction: CALL 2070H

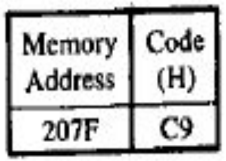
Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand	70
M_3 Memory Read	23FF	2042	20 43	20 Operand	20
M_4 Memory Write	23FE (SP–2)	23FF	20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

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

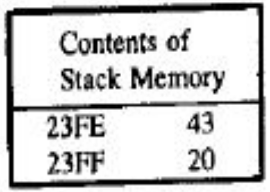
Data Transfer During the Execution of the CALL Instruction

RET Execution

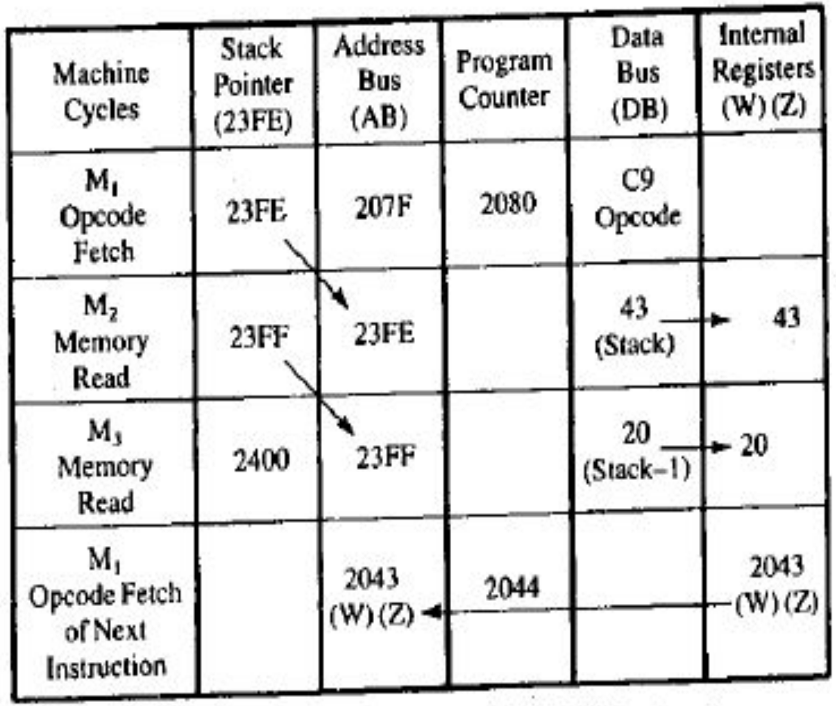
Program execution sequence is transferred to the memory location 2043H location.M1 is normal fetch cycle during M2 contents of stack pointer are placed on address bus so 43H data is fetched and stored on Z register and SP is upgraded. Similarly for M3. Program sequence is transfered to2043H by placing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



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.

Conditional call instructions

The conditional Call and Return instructions are based on four flag conditions (Carry , Zero, Sign and Parity)

In case of conditional call the program is transferred to the subroutine if condition is met.

In case of a conditional Return instruction, the sequence returns to the main program if the condition is met.

Conditional Call

CC - Call subroutine if Carry flag is set (CY=1)
CNC - Call subroutine if Carry flag is reset (CY=0)
CZ - Call subroutine if Zero flag is set (Z=1)
CNZ - Call subroutine if Zero flag is reset (Z=0)
CM - Call subroutine if sign flag is set (S=1 , negative number)
CP - Call subroutine if sign flag is reset (S=0 , positive number)
CPE - Call subroutine if parity flag is set (P=1 , even parity)
CPO - Call subroutine if parity flag is reset (P=0 , odd parity)

Conditional Return instructions

Conditional Return

RC - Return if Carry flag is set (CY=1)

RNC - Return if Carry flag is reset (CY=0)

RZ - Return if Zero flag is set (Z=1)

RNZ - Return if Zero flag is reset (Z=0)

RM - Return if Sign flag is set (S=1, negative number)

RP - Return if Sign flag is reset (S=0, positive number)

RPE - Return if parity flag is set (P=1, even parity)

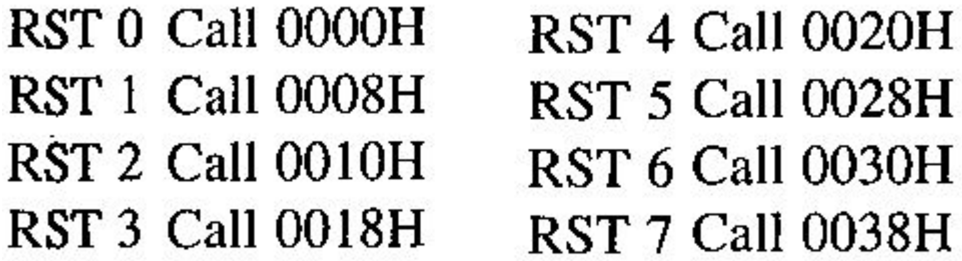
RPO - Return if parity flag is reset (P=0, odd parity)

RESTART instructions

In addition to the unconditional CALL and RET instructions, the 8085 instruction set includes eight Restart instructions and eight conditional call and Return instructions.

RST instruction (3 machine cycles and , 12 T-states)

- 1 byte call instructions
- Transfer the program execution to a specific location on page 00H
- Execute the same way as CALL instructions.
- Used in conjunction with interrupt.



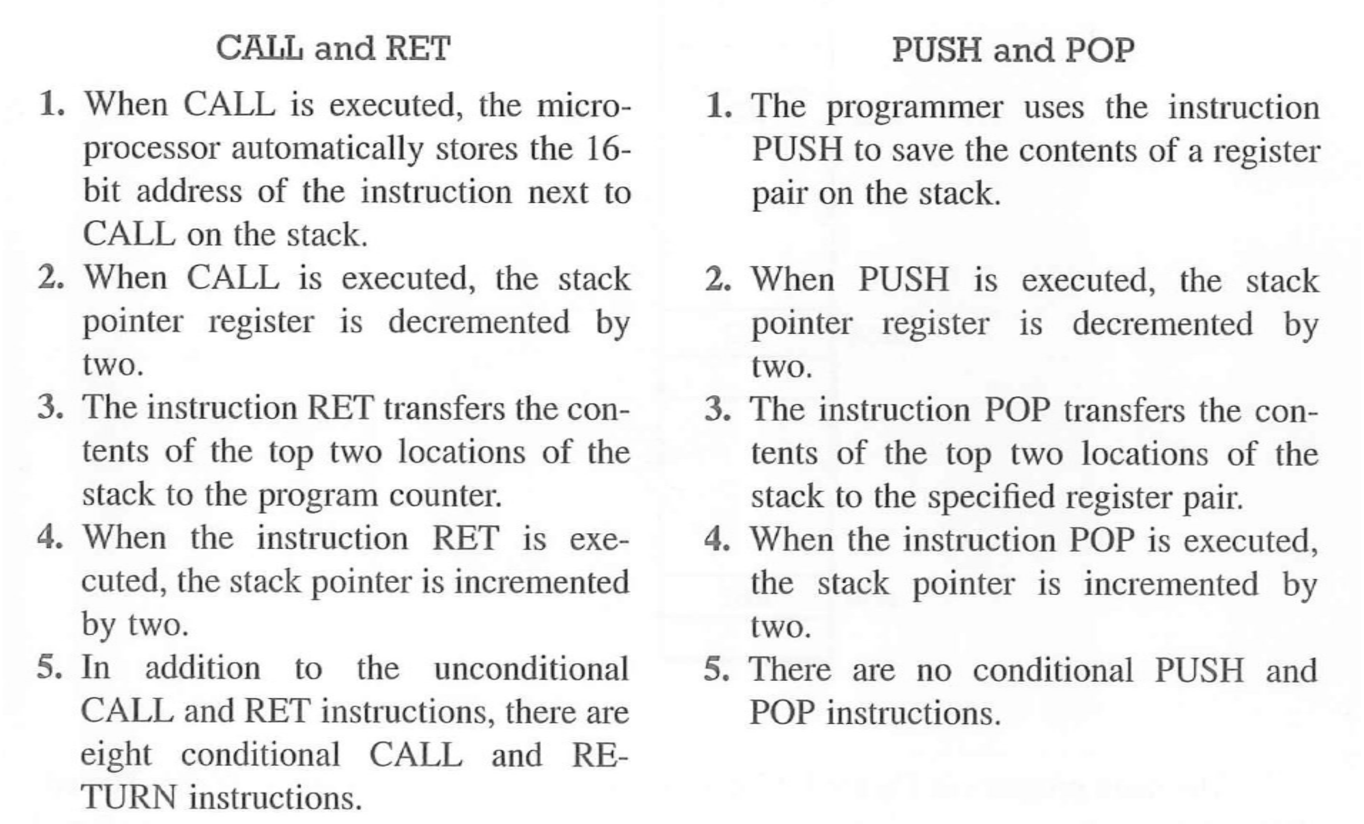
RST 0	Call 0000H	RST 4	Call 0020H
RST 1	Call 0008H	RST 5	Call 0028H
RST 2	Call 0010H	RST 6	Call 0030H
RST 3	Call 0018H	RST 7	Call 0038H

Example

Write a program that will display FF and 11 repeatedly on the seven segment display. Write a ‘delay’ subroutine and call as it necessary

```
                2000    ;  LXI SP, FFFF
                2003    :   MVI A, FF
                2005   :   OUT PORT1
                2007   :  CALL  2014
                200A   :   MVI A, 11
                200C    :   OUT PORT1
                200E    :   CALL 2014
                2011   :   JMP 2003
DELAY    :       2014    :   MVI B FF
                 2016    :   MCI C FF
                 2018   :    DCR C
                 2019   :    JNZ 2018
                 201C   :     DCR B
                 201D   :   JNZ  2016
                 2020   :    RET
```

Difference between CALL ,RET and PUSH,POP



CALL and RET	PUSH and POP
1. When CALL is executed, the microprocessor automatically stores the 16-bit address of the instruction next to CALL on the stack.	1. The programmer uses the instruction PUSH to save the contents of a register pair on the stack.
2. When CALL is executed, the stack pointer register is decremented by two.	2. When PUSH is executed, the stack pointer register is decremented by two.
3. The instruction RET transfers the contents of the top two locations of the stack to the program counter.	3. The instruction POP transfers the contents of the top two locations of the stack to the specified register pair.
4. When the instruction RET is executed, the stack pointer is incremented by two.	4. When the instruction POP is executed, the stack pointer is incremented by two.
5. In addition to the unconditional CALL and RET instructions, there are eight conditional CALL and RETURN instructions.	5. There are no conditional PUSH and POP instructions.