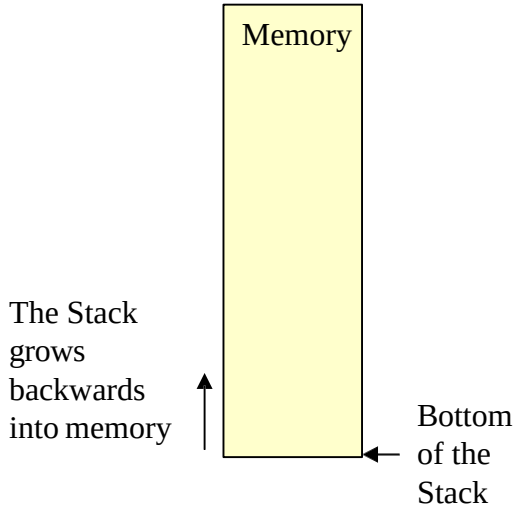
UNIT 3
Stack and Subroutines

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

- 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).
- The Size of the stack is limited only by the available memory

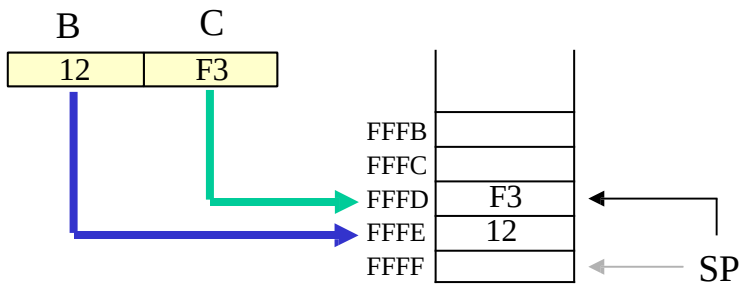
Saving Information on the Stack

- 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.

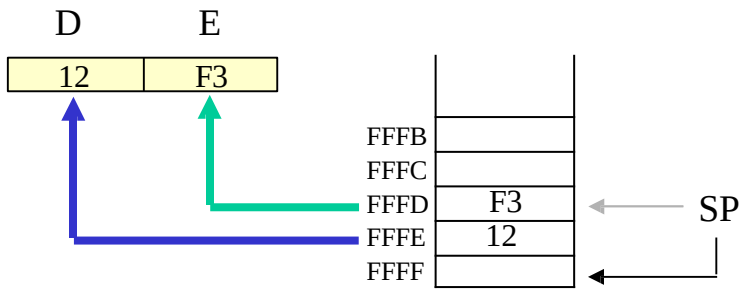
The PUSH Instruction

- PUSH B (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.

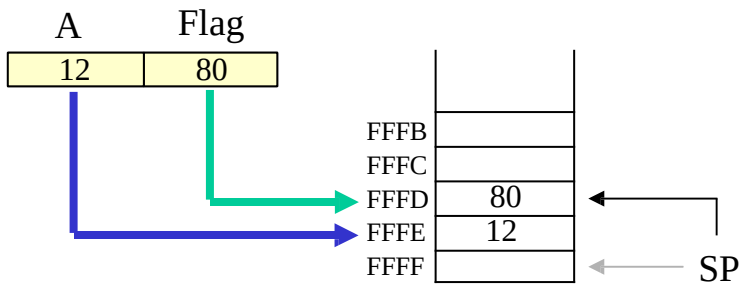
The PSW Register Pair

- 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.

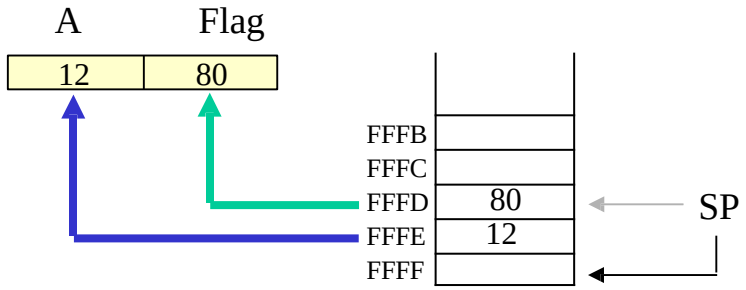
PUSH PSW Register Pair

- 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
- 7 6 5 4 3 2 1 0
- 8085 Flag :

S	Z	x	AC	x	P	x	Cy

- 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.

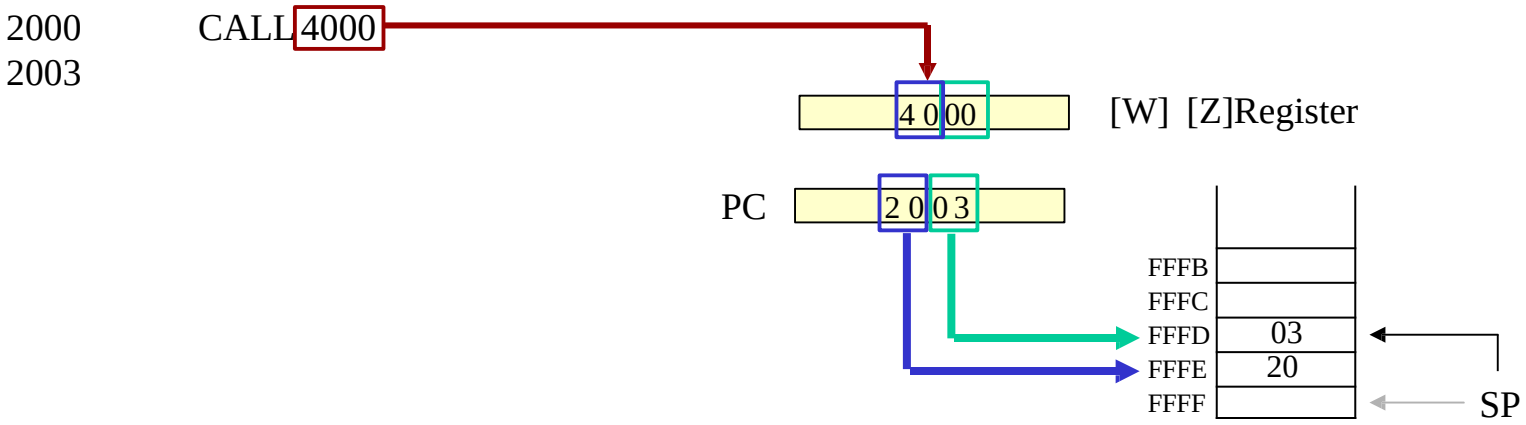
- In Assembly language, a subroutine can exist anywhere in the code.
  - However, it is customary to place subroutines separately from the main program.

Subroutines

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

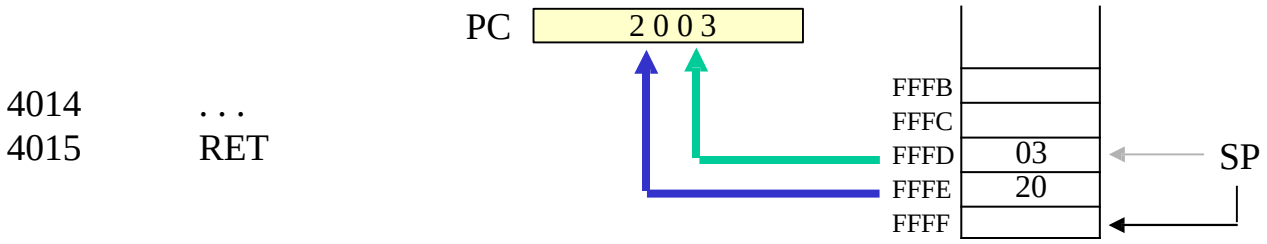


The CALL Instruction

- 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
- Pushe 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 loose the return address.

Things to be considered in Subroutine

- 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.

Passing Data to a Subroutine

- Data is passed to a subroutine through registers.
  - Call by Reference:
    - The data is stored in one of the registers by the calling program and the subroutine uses the value from the register. The register values get modified within the subroutine. Then these modifications will be transferred back to the calling program upon returning from a subroutine
  - Call by Value:
    - The data is stored in one of the registers, but the subroutine first PUSHES register values in the stack and after using the registers, it POPS the previous values of the registers from the stack while exiting the subroutine. i.e. the original values are restored before execution returns to the calling program.

Passing Data to a Subroutine

- 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.

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

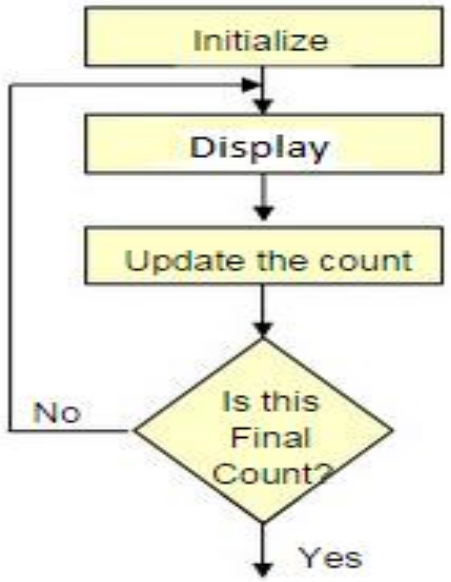
- The 8085 supports conditional CALL and conditional RTE instructions.
  - The same conditions used with conditional JUMP instructions can be used.

  - 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.

COUNTERS AND TIME DELAYS

Counter and Time Delays

•A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions.
•Loop is established to update the count.
•Each count is checked to determine whether it has reached final number ;if not, the loop is repeated.

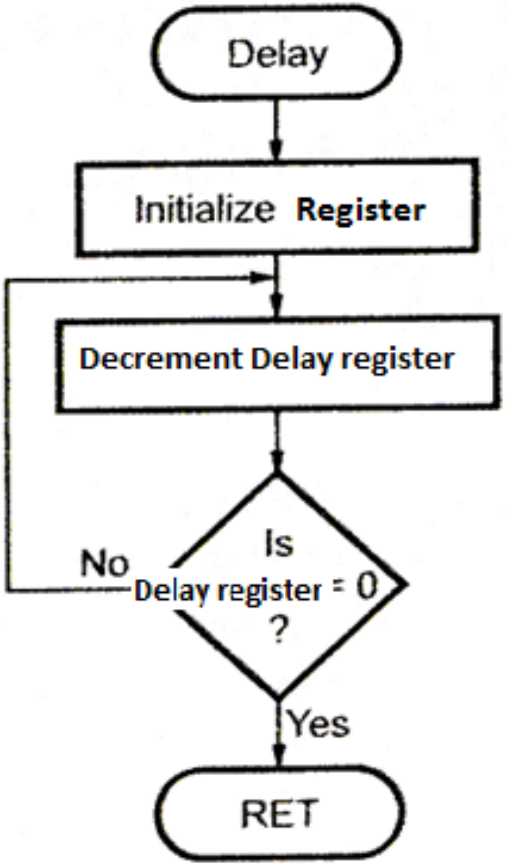


Time Delay

Procedure used to design a specific delay.
A register is loaded with a number , depending on the time delay required and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction.

Time delay using

One register:



Label	Opcode	Operand	Comments	T states
	MVI	C,FFH	;Load register C	7
LOOP:	DCR	C	;Decrement C	4
	JNZ	LOOP	;Jump back to decrement C	10/7

Clock frequency of the system = 2 MHz

Clock period= 1/T= 0.5 μs

Time to execute MVI = 7 T states * 0.5= 3.5 μs

Time Delay in Loop T_L= T*Loop T states * N_{10}

= 0.5 * 14* 255

= 1785 μs = 1.8 ms

N_{10} = Equivalent decimal number of hexadecimal count loaded in the delay register

T_{LA}= Time to execute loop instructions

=T_L –(3T states* clock period)=1785-1.5=1783.5 μs

Time Delay using a register pair

Label	Opcode	Operand	Comments	T states
	LXI	B,2384H	Load BC with 16-bit count	10
LOOP:	DCX	B	Decrement BC by 1	**6**
	MOV	A,C	Place contents of C in A	**4**
	ORA	B	OR B with C to set Zero flag	**4**
	JNZ	LOOP	if result not equal to 0 , jump back to loop	**10**/7

Time Delay in Loop T_L= T*Loop T states * N_{10}

= 0.5 * 24* 9092

= 109 ms

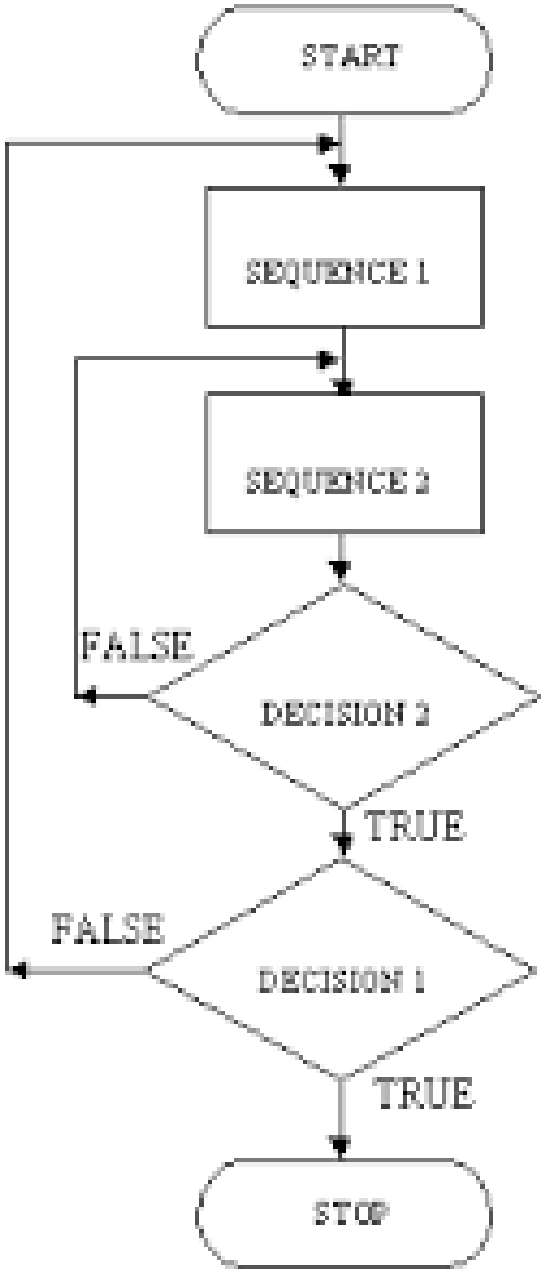
Time Delay using a LOOP within a LOOP

	MVI B,38H	7T
LOOP2:	MVI C,FFH	**7T**
LOOP1:	DCR C	4T
	JNZ LOOP1	10/7 T
	DCR B	**4T**
	JNZ LOOP 2	**10**/7T

Delay in Loop T_{L1}=1783.5 μs

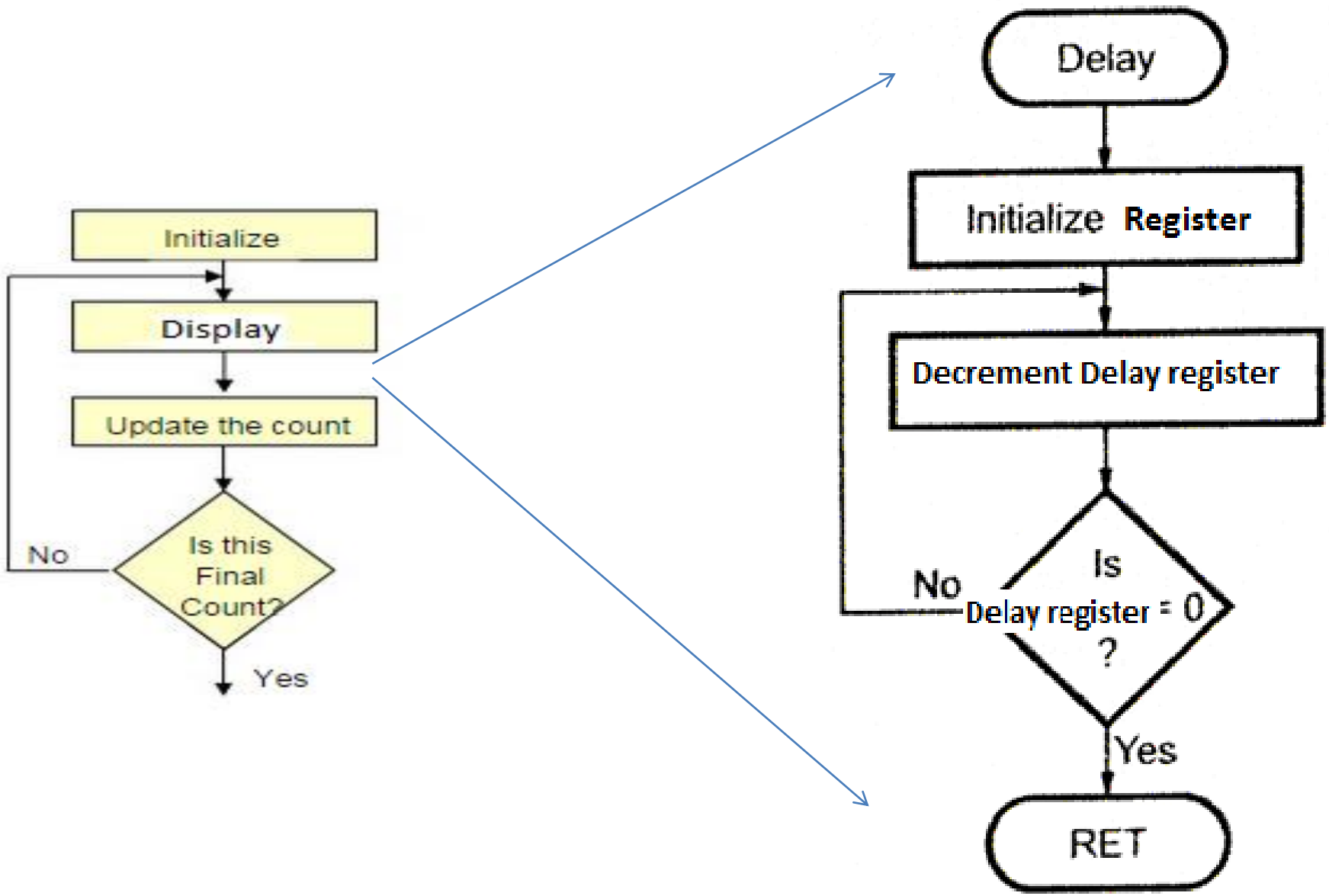
Delay in Loop T_{L2}= (0.5*21+T_{L1})*56

=100.46ms



Flowchart for time delay with two loops

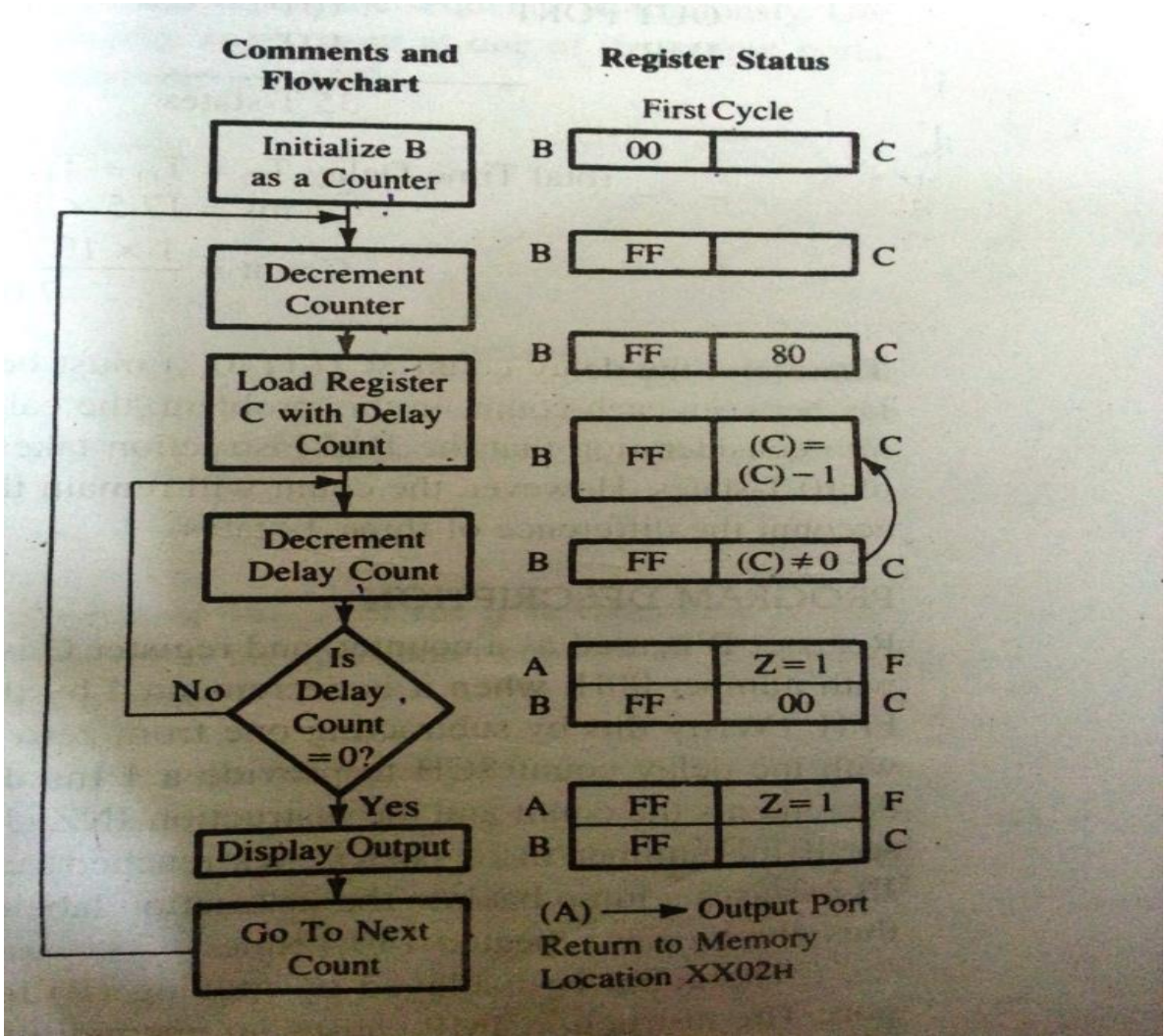
Flowchart of a counter with time delay



Illustrative Program: Hexadecimal Counter

Write a Program to count continuously from FFH to 00H using register C with delay count 8CH between each count and display the number at one of the output ports.

```
        MVI B,00H
NEXT:   DCR B
        MVI C,8CH
DELAY:  DCR C
        JNZ DELAY
        MOV A,B
        OUT PORT#
        JMP NEXT
```



Illustrative Program: Zero to nine (Modulo ten) Counter

```
START:   MVI B,00H
         MOV A,B
DSPLAY:  OUT  PORT #
         LXI H,16-bit
LOOP:    DCX H
         MOV A,L
         ORA H
         JNZ LOOP
         INR B
         MOV A,B
         CPI 0AH
         JNZ DSPLAY
         JZ START
```