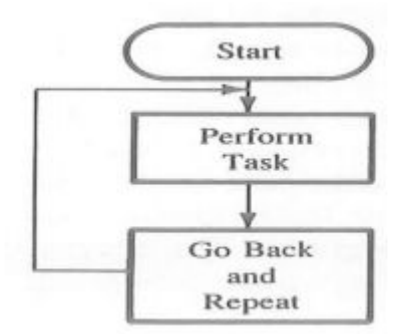
UNIT 3
Counters and time delays

Need for counter and Time delays

- ❖ Counters are used primarily to keep track of events
- ❖ Time delays are important in setting up reasonably accurate timing between two events
- ❖ The process of designing counters and time delays using software instructions is far more flexible and less time consuming than the design process using hardware

Counters

- ❖ The programming technique used to instruct the microprocessor to repeat tasks is called looping. A loop can be set up to change the sequence of execution and perform the task again.
- ❖ Loops can be classified into two groups:
  - ➢ Continuous loop - repeats a task continuously
  - ➢ Conditional loop - repeats a task until certain data conditions are met
- ❖ **Continuous loop** - a program with a continuous loop does not stop repeating until the system is reset
- ❖



Counters

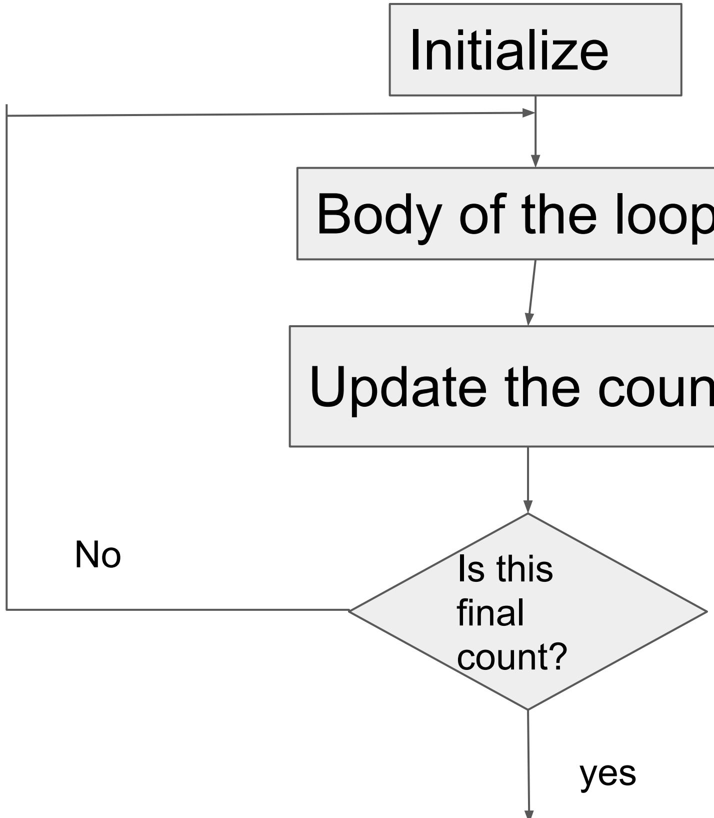
Conditional loop

A conditional loop is set up by the conditional jump instructions. These instructions check flags (zero, carry etc) and repeat the specified tasks if the conditions are satisfied. These includes counting and indexing .

- ❖ A loop counter is set up by loading a register with certain value.
- ❖ Then using the DCR (to decrement) or the INR(increment) the contents of the register are updated
- ❖ A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.

Counters

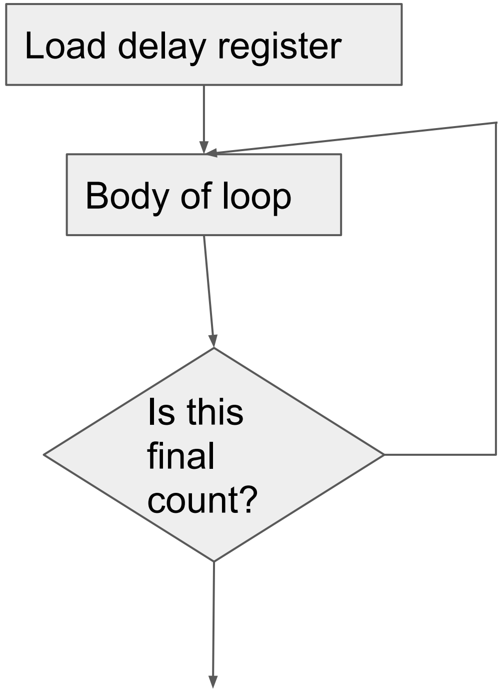
The operation of a loop counter can be described using the following flowchart.



Loop counter can be setup with a

- single register,
- register pair or
- loop within loop.

Time delay



- The procedure used to design a specific delay is similar to that used to set up a counter.
- 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 a conditional jump instruction.
- The loop causes the delay, depending upon the clock period of the system.

Calculating Time Delays

Each instruction passes through different combinations of the following cycles

- Opcode Fetch,
- Memory Read
- Memory write

Knowing the combinations of cycles, one can calculate how long such an instruction would require to complete.

- Number of Bytes
- Number of Machine cycles
- Number of T-State.

Calculating Time Delays

The time delay can be calculated as:

Time Delay = No.of T-States * Clock Period

For example

MVI B, 45H is a two byte instruction.

The hexcode is 06 45H

In the above code

06 is Opcode fetch (one Machine cycle- 4T states)

45H (one machine cycle - 3T states)

So the total time is (4T+3T=7T states)

If one T state time = 0.5 μs

Therefore for 7T states= 7 * 0.5 = 3.5 μs

Time delay

Time Delay can be designed using the following technique:

1. Using one register
2. Using the register pair
3. Using a loop with in a loop

Using a single register

Consider the following loop In C++

```
c=15;
do
{
c=c-1;
}while (c>0)
```

```
        MVI  C, FFH      7 T-states
LOOP    DCR C            4 T-states
        JNZ LOOP      10/7T-states
```

- The first instruction initializes the loop counter C and is executed only once requiring only 7T-States
- The following two instructions form a loop that requires 14 T-states to execute and is repeated 255 times until C becomes 0
- To calculate the delay, we use the following formula:
- $T_{delay} = T_o + T_L$

Using a single register

```
          MVI  C, FFH      7 T-states
LOOP      DCR C            4 T-states
          JNZ LOOP      10/7 T-states
```

- To calculate the delay, we use the following formula:

  $$T_{delay} = T_o + T_L$$

  T_{delay} = total delay
  T_o = delay outside the loop
  T_L = delay of the loop
  T_o = is the sum of all delays outside the loop
  T_L = is calculated using the formula

  $$T_L = T * \text{loop T-States} * N \text{ (no. of iterations)}$$

Using one register

Using these formulas, we can calculate the time delay for the previous example:

To= 7 T-States

TL= (14 * 255) - 3 = 3567 T-States

(14 T-states for the 2 instructions repeated 255 times)

(FF=255) reduced by the 3 T-States for the final JNZ) (jumping and changing of sequence requires 10 T states whereas final JUMP jumps to the next instruction following it. So this requires only 7T states

$T_{delay}=[(T_o + T_L)/f]$ assume (f=2 MHz)

= (7 + 3567)/2 MHz

= (3574) x 0.5 μs

= 1.787 μs

Using a Register Pair as a Loop Counter

Using a single register, one can repeat a loop for a maximum count of 255 times

This count can be increased by using register pair for the loop counter instead of single register. A minor problem arises in how to test for the final count since DCX and INX do not modify the flags.–However, if the loop is looking for when the count becomes zero, we can use a small trick by Oringthe two registers in the pair and then checking the zero flag.

```
0010 0011 1000  0100
A= 1000 0011                    0000  0000
B= 0010 0011                    0000  0000
--------------------         --------------------
     1010    0011                0000  0000
```

Example

```
        LXI B, 2384H      10 T-States
LOOP    DCX  B            6   T-States
        MOV A, C          4 T-States
        ORA B             4 T-States
        JNZ LOOP          10 T-States
```

To = 10 T-States   (delay for the LXI instructions)

$2384H= 2 \times (16)^3+3 \times (16)^2+8 \times (16)^1+4 \times (16)^0$

$= 9092_{10}$

Clock period= 0.5 μs

$T_L = (0.5 \times 24 \times 9092) = 109$ ms

(24 T-States for the 4 instructions in the loop repeated 9092 times reduced by the 3 T-States for the JNZ in the last iteration)

Tdelay =109ms+ To

Nested loops

Nested loops can be set using two registers as two loop counters and updating the right register in the respective loop.

Example

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

Delay calculation

LOOP1 delay T_{L1}= 1783.5 µs

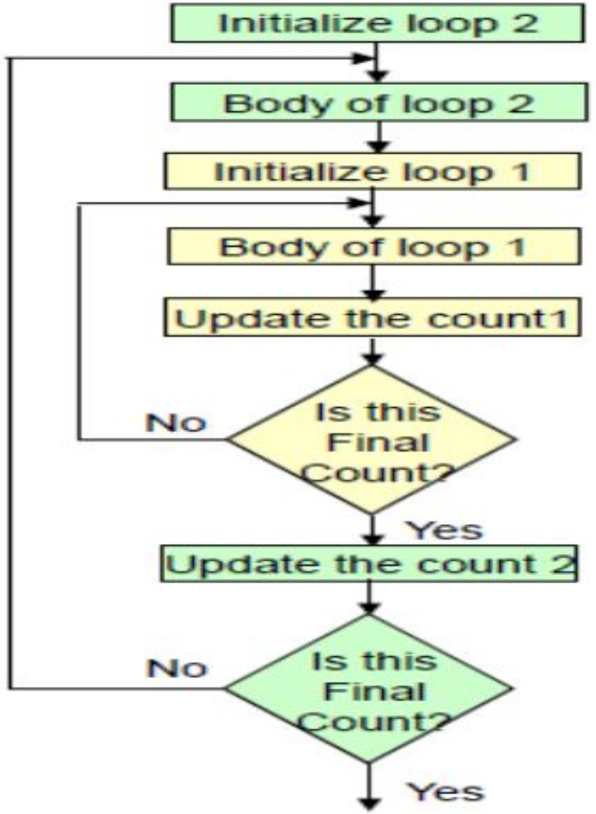
LOOP2 delay T_{L2} = 56(T_{L1} + 21 T-states x 0.5 µs)

= 56(1783.5 + 10.5) µs

= 100.46 ms

Loops in C++

```
for(i=0;i<n;i++)
{
     for(j=0;j<n;j++)
     {
      ;
     }
}
```



Increasing the delay

- The delay can be increased by using register pair for each loop counter in the nested loop setup
- It can also be increased by adding dummy instructions (like NOP= 4T state) in the body of the loop

Disadvantages of time delay

The disadvantage of using software delay is

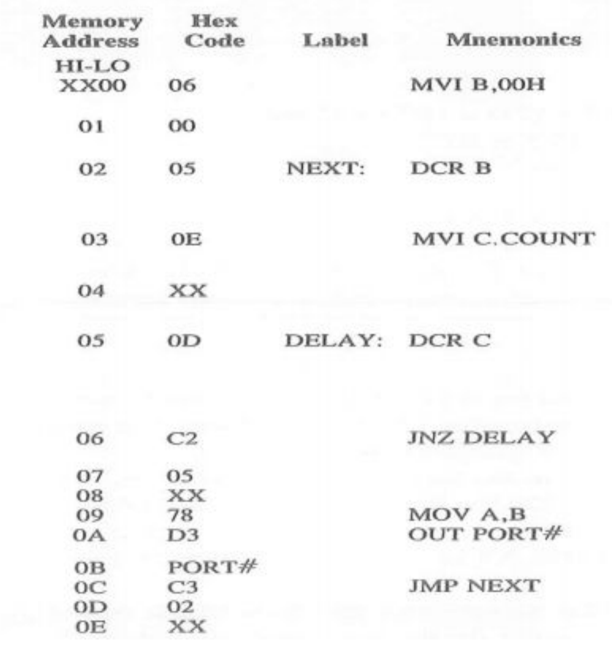
1. The accuracy of time delay depends on system’s clock
2. The microprocessor is occupied simply in a waiting loop.other wise it could perform other functions
3. The task of calculating accurate time delay is tedious

Hexadecimal counter

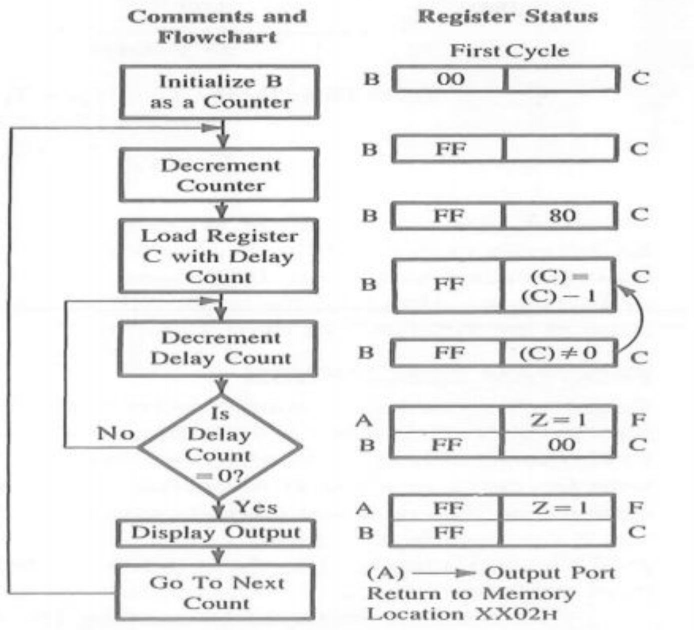
Problem statement: write a program to count continuously in hexadecimal from FFH to 00H in a system with a 0.5 μs . Use register C to set up a one millisecod (ms) delay between each count and display the numbers at one of the output ports

B register

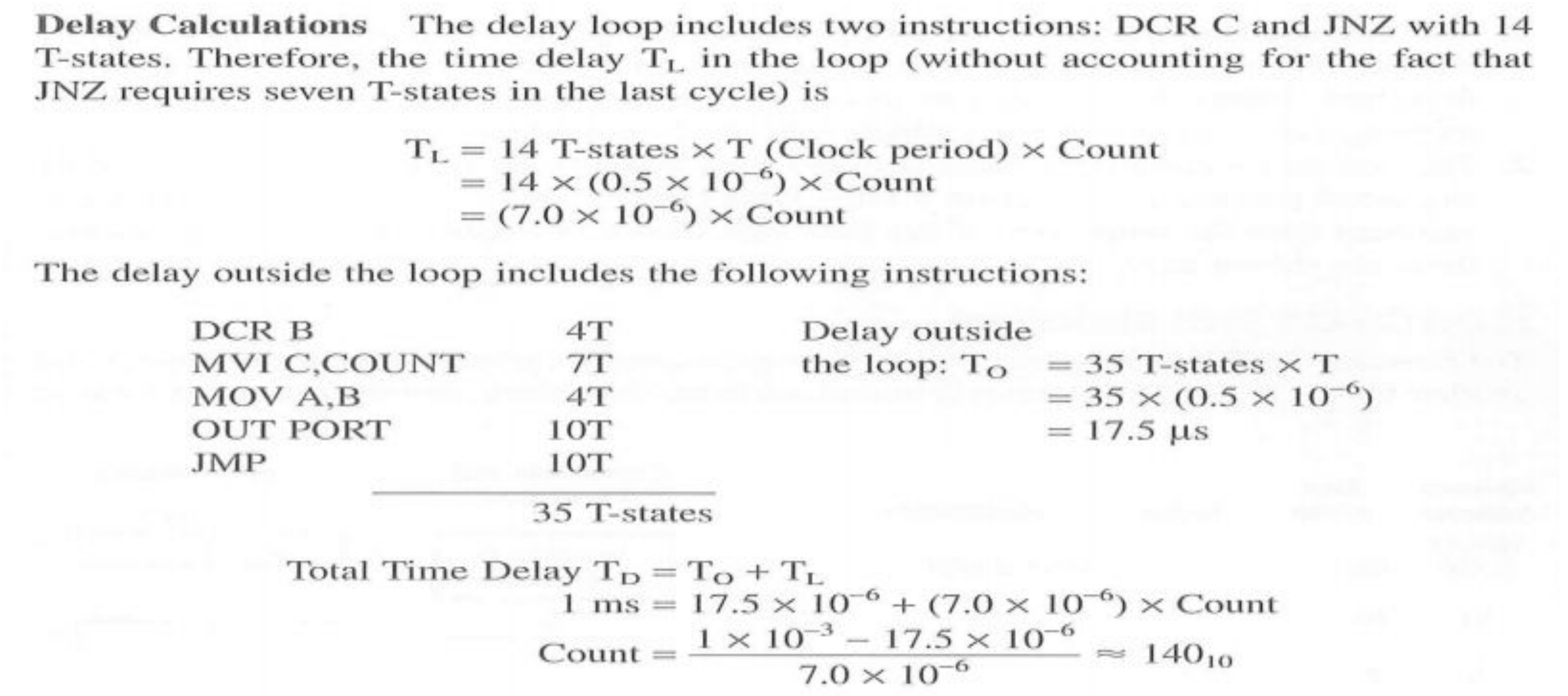
FF
FE
FD
;
00



Memory Address HI-LO	Hex Code	Label	Mnemonics
XX00	06		MVI B,00H
01	00		
02	05	NEXT:	DCR B
03	0E		MVI C.COUNT
04	XX		
05	0D	DELAY:	DCR C
06	C2		JNZ DELAY
07	05		
08	XX		
09	78		MOV A,B
0A	D3		OUT PORT#
0B	PORT#		
0C	C3		JMP NEXT
0D	02		
0E	XX		



Delay calculation of hexa decimal counter



Delay Calculations The delay loop includes two instructions: DCR C and JNZ with 14 T-states. Therefore, the time delay T_L in the loop (without accounting for the fact that JNZ requires seven T-states in the last cycle) is

$$\begin{aligned} T_L &= 14 \text{ T-states} \times T \text{ (Clock period)} \times \text{Count} \\ &= 14 \times (0.5 \times 10^{-6}) \times \text{Count} \\ &= (7.0 \times 10^{-6}) \times \text{Count} \end{aligned}$$

The delay outside the loop includes the following instructions:

DCR B	4T
MVI C,COUNT	7T
MOV A,B	4T
OUT PORT	10T
JMP	10T
	35 T-states

Delay outside the loop: T_O

$$\begin{aligned} T_O &= 35 \text{ T-states} \times T \\ &= 35 \times (0.5 \times 10^{-6}) \\ &= 17.5\ \mu s \end{aligned}$$

$$\begin{aligned} \text{Total Time Delay } T_D &= T_O + T_L \\ 1 \text{ ms} &= 17.5 \times 10^{-6} + (7.0 \times 10^{-6}) \times \text{Count} \\ \text{Count} &= \frac{1 \times 10^{-3} - 17.5 \times 10^{-6}}{7.0 \times 10^{-6}} \approx 140_{10} \end{aligned}$$

$140_{10} = 8C_H$ must be loaded in C register to obtain 1ms delay

Zero to nine (Modulo TEN) counter

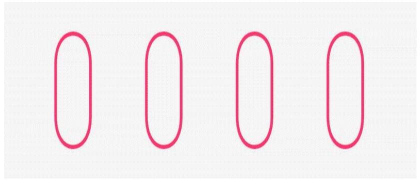
Problem statement : Write a program to count from 0 to 9 with a one-second delay between each count.at the count of 9, the counter should reset itself to 0 and repeat the sequence continuously. Use register pair HL, to set up the delay, and display each count at one of the output ports. The clock freq is 1 MHz.

Instructions in this program

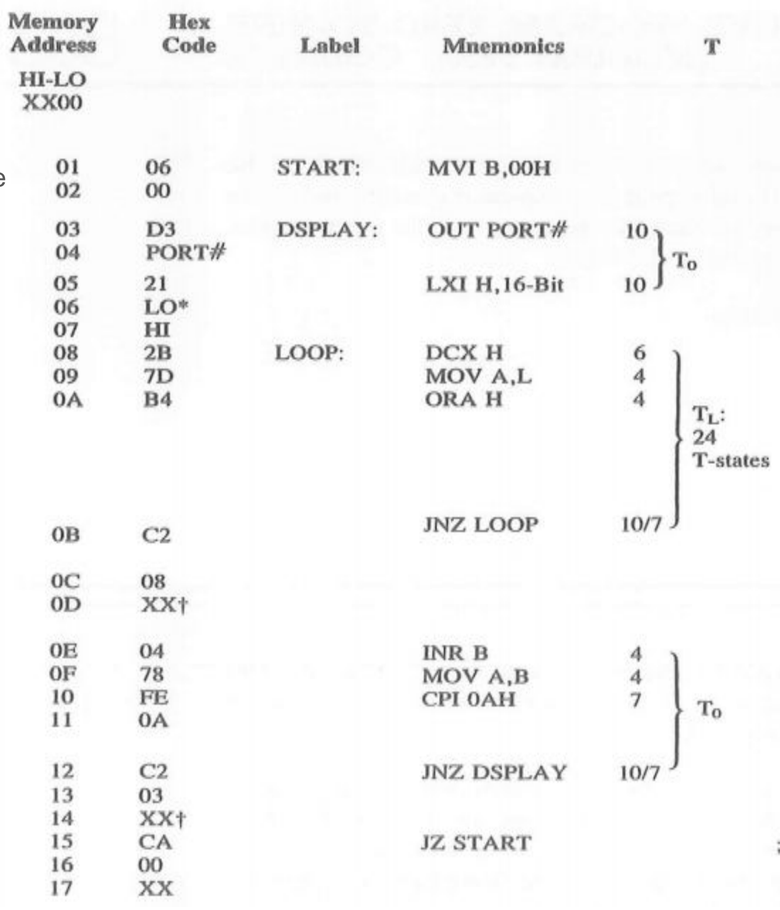
LXI : Load Register Pair immediate

DCX : Decrement Register Pair

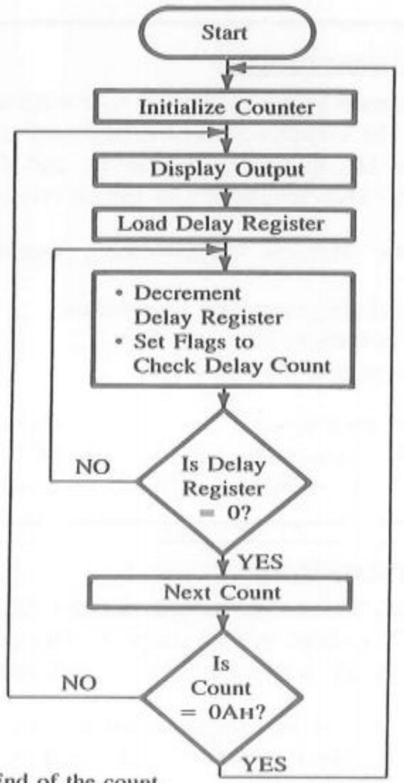
INX : Increment Register Pair



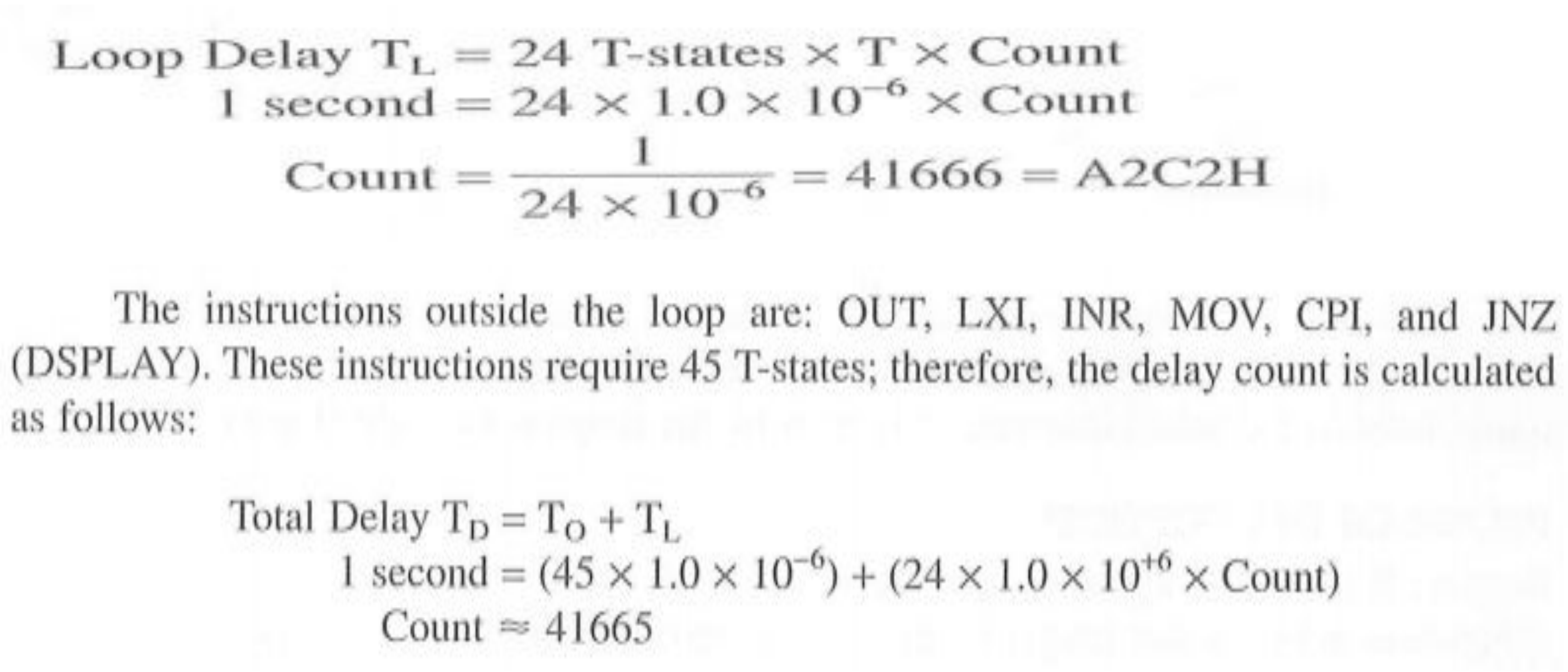
01:00



Memory Address	Hex Code	Label	Mnemonics	T	Comments and Flowchart
HI-LO XX00					
01	06	START:	MVI B,00H		
02	00				
03	D3	DSPLAY:	OUT PORT#	10	} T_O
04	PORT#				
05	21		LXI H,16-Bit	10	
06	LO*				
07	HI				
08	2B	LOOP:	DCX H	6	T_L: 24 T-states
09	7D		MOV A,L	4	
0A	B4		ORA H	4	
0B	C2		JNZ LOOP	10/7	
0C	08				
0D	XX†				
0E	04		INR B	4	T_O
0F	78		MOV A,B	4	
10	FE		CPI 0AH	7	
11	0A				
12	C2		JNZ DSPLAY	10/7	
13	03				
14	XX†				
15	CA		JZ START		;End of the count, start again
16	00				
17	XX				



Delay calculations



$$\text{Loop Delay } T_L = 24 \text{ T-states} \times T \times \text{Count}$$
$$1 \text{ second} = 24 \times 1.0 \times 10^{-6} \times \text{Count}$$
$$\text{Count} = \frac{1}{24 \times 10^{-6}} = 41666 = \text{A2C2H}$$

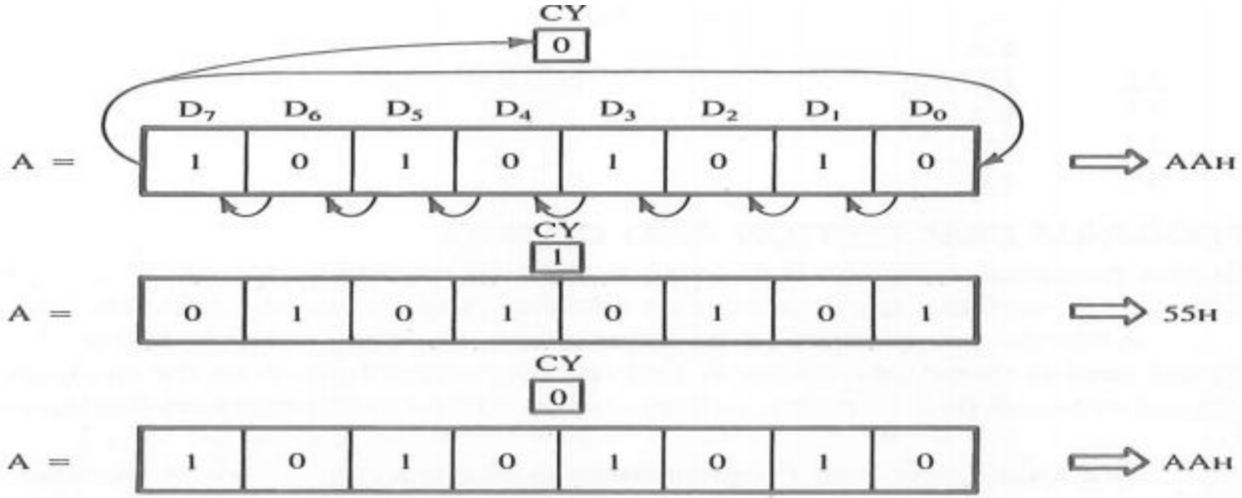
The instructions outside the loop are: OUT, LXI, INR, MOV, CPI, and JNZ (DSPLAY). These instructions require 45 T-states; therefore, the delay count is calculated as follows:

$$\text{Total Delay } T_D = T_O + T_L$$
$$1 \text{ second} = (45 \times 1.0 \times 10^{-6}) + (24 \times 1.0 \times 10^{+6} \times \text{Count})$$
$$\text{Count} \approx 41665$$

Logic operations - Rotate

RLC : Rotate Accumulator Left
RAL : Rotate Accumulator Left through carry
RRC : Rotate Accumulator Right
RAR : Rotate Accumulator Right through carry

RLC - Rotate Accumulator Left
Each bit is shifted to next bit in left position. Bit D_7 becomes D_0
CY flag is modified according to bit D_7
Assume the A= AAH and CY=0. illustrate the accumulator contents after the execution of RLC

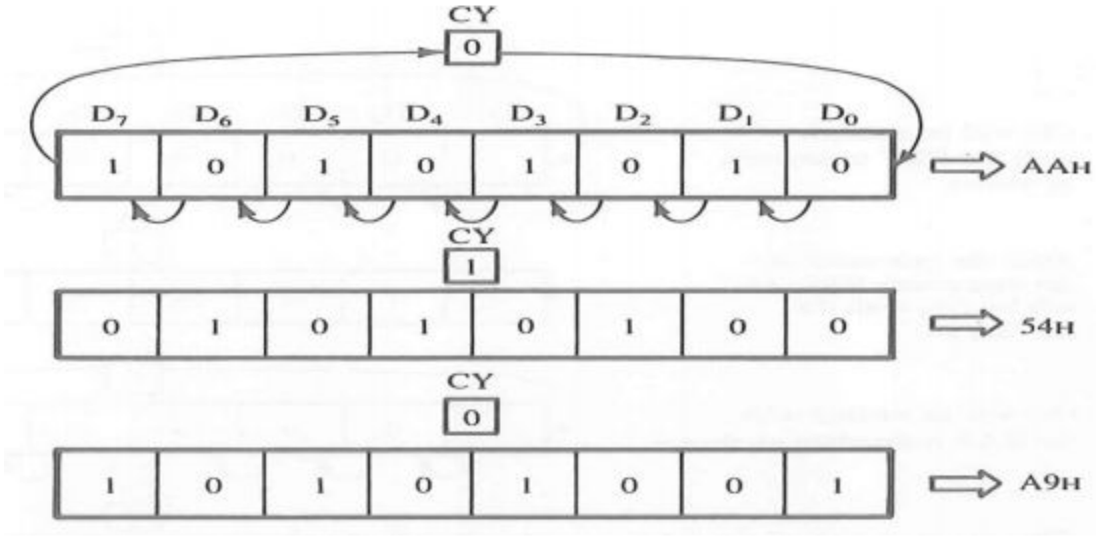


Logic operations - Rotate - RAL

RAL - Rotate Accumulator Left Through Carry (9 bit rotation)

Each bit is shifted to the adjacent left position. Bit D_7 becomes the carry bit and the carry bit is shifted into D_0

The carry flag is modified according to D_7

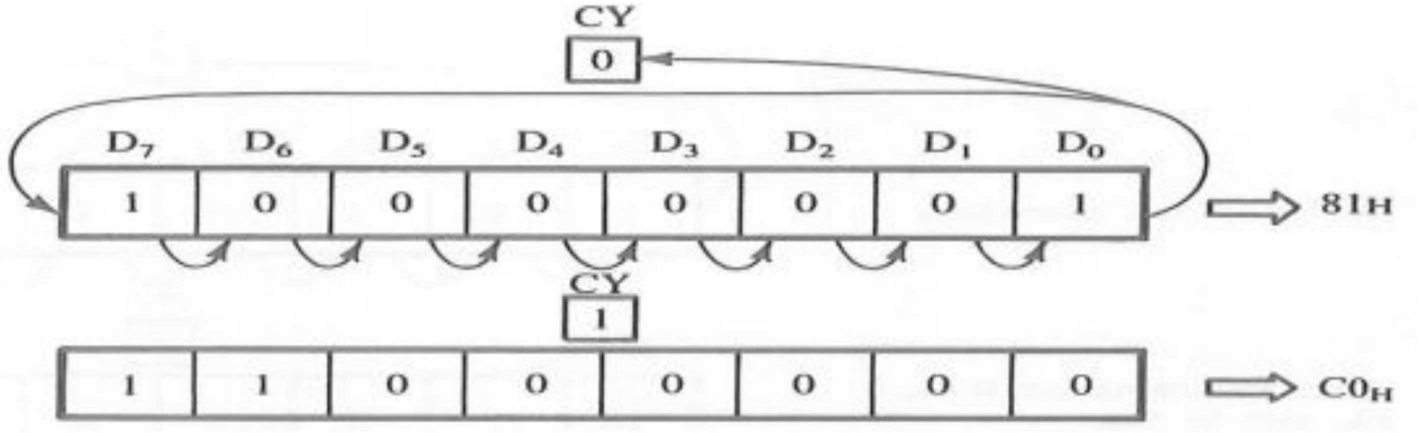


Logical operations - Rotate - RRC

RRC - Rotate Accumulator Right

Each bit is shifted to right to the adjacent position. Bit D_0 becomes D_7
CY flag is modified according to bit D_0

Assume the contents of A= 81H and CY=0



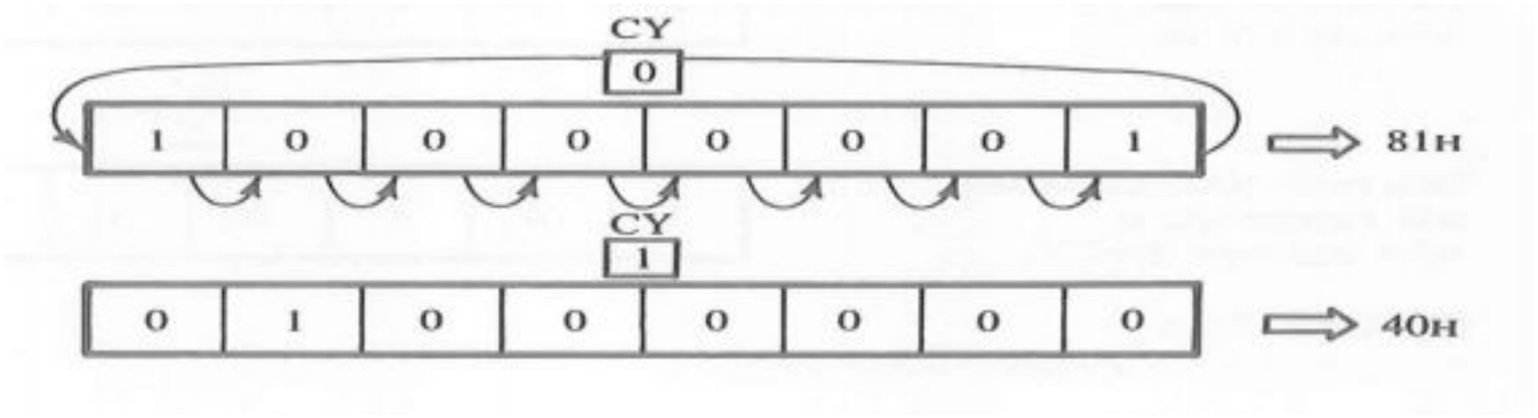
Logical operations - Rotate - RAR

RRC - Rotate Accumulator Right along with carry

Each bit is shifted to right to the adjacent position. Bit D_0 becomes the carry bit, and the carry bit is shifted into D_7

CY flag is modified according to bit D_0

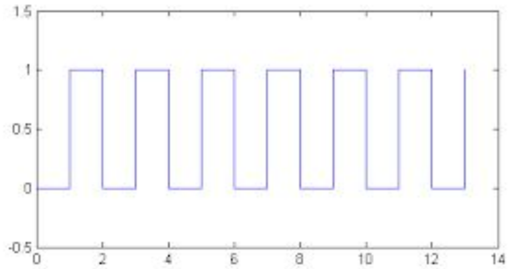
Assume the contents of A= 81H and CY=0

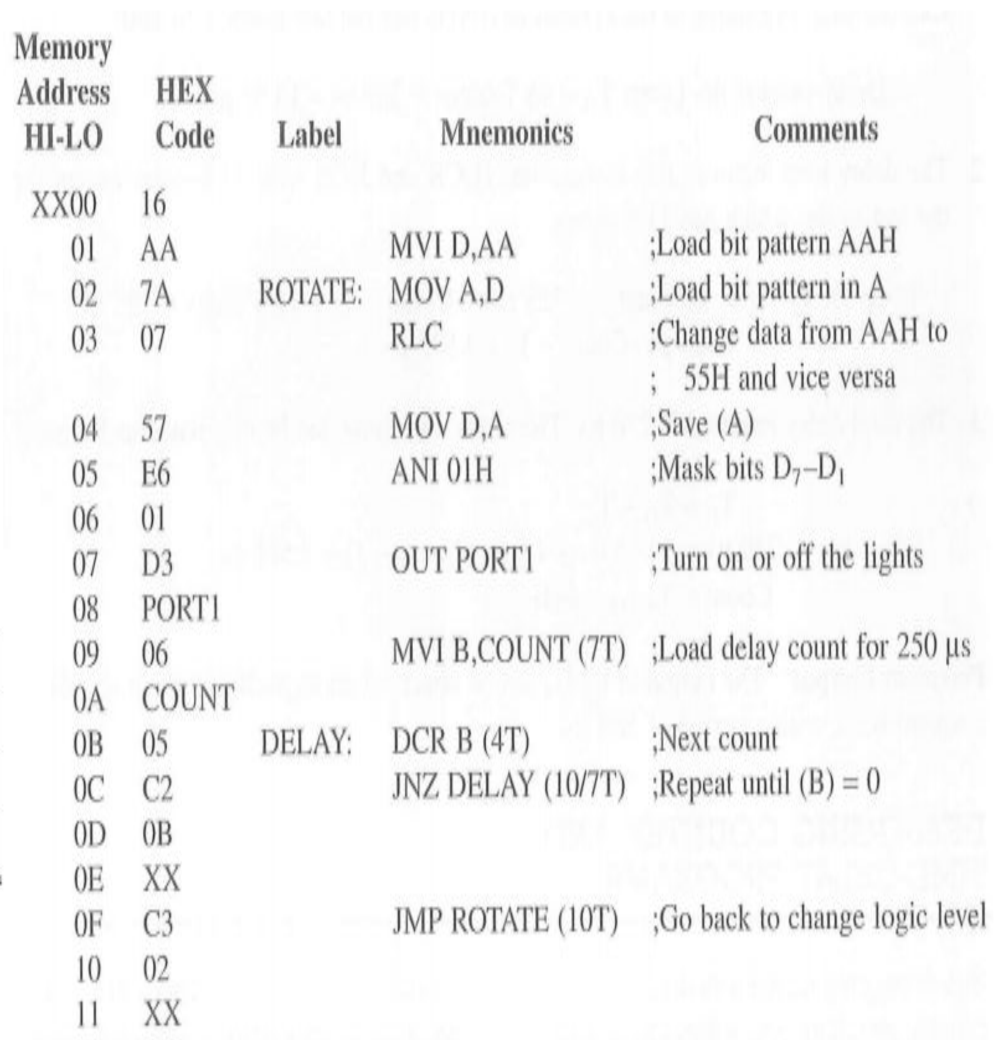


Generating pulse waveforms

Problem statement : write a program to generate a continuous square wave with the period of 500 μs . Assume the system clock period is 325 ns, and use bit D_0 to output the square wave. The instructions used in this program.

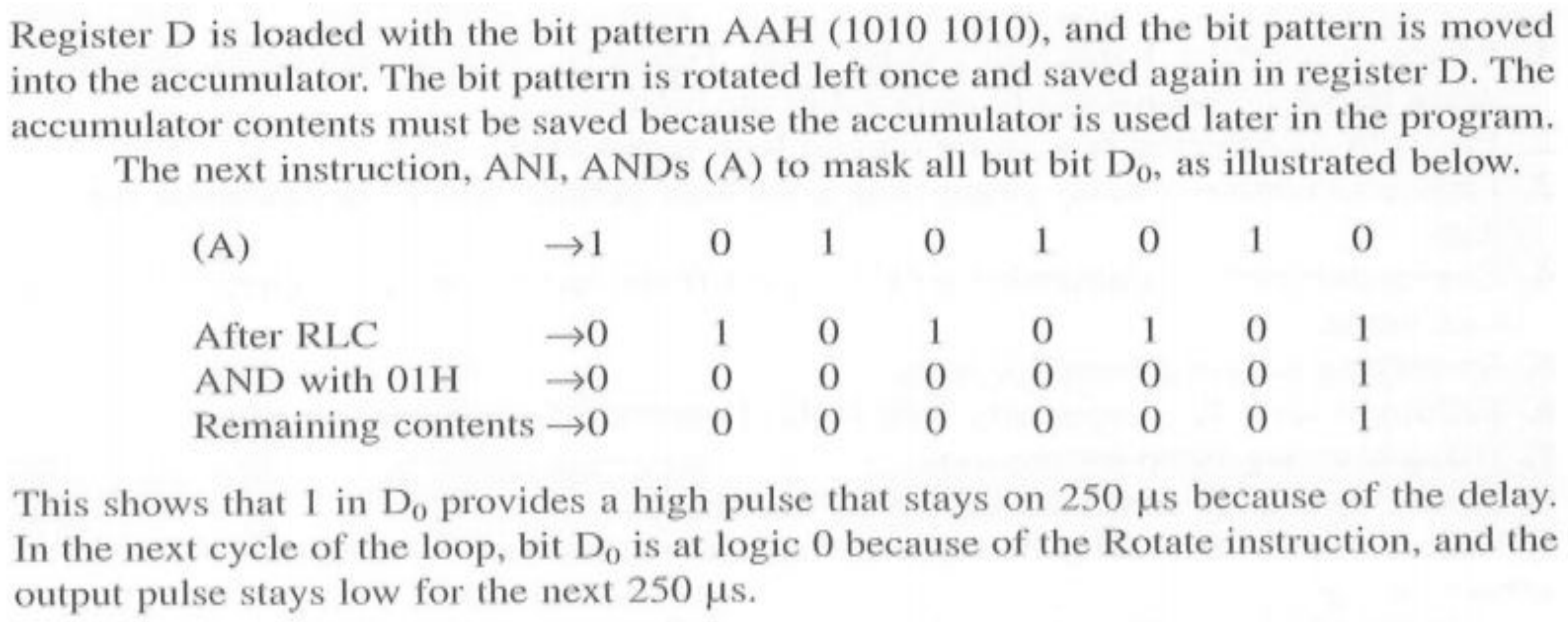
Mov A,D	4T
RLC	4T
MOV D,A	4T
ANI,01	7T
Out	10T
MVI	7T
JMP rotate	10T
	----------
	46 T





Memory Address HI-LO	HEX Code	Label	Mnemonics	Comments
XX00	16			
01	AA		MVI D,AA	;Load bit pattern AAH
02	7A	ROTATE:	MOV A,D	;Load bit pattern in A
03	07		RLC	;Change data from AAH to ; 55H and vice versa
04	57		MOV D,A	;Save (A)
05	E6		ANI 01H	;Mask bits D_7–D_1
06	01			
07	D3		OUT PORT1	;Turn on or off the lights
08	PORT1			
09	06		MVI B,COUNT (7T)	;Load delay count for 250 μs
0A	COUNT			
0B	05	DELAY:	DCR B (4T)	;Next count
0C	C2		JNZ DELAY (10/7T)	;Repeat until (B) = 0
0D	0B			
0E	XX			
0F	C3		JMP ROTATE (10T)	;Go back to change logic level
10	02			
11	XX			

Generating pulse waveforms



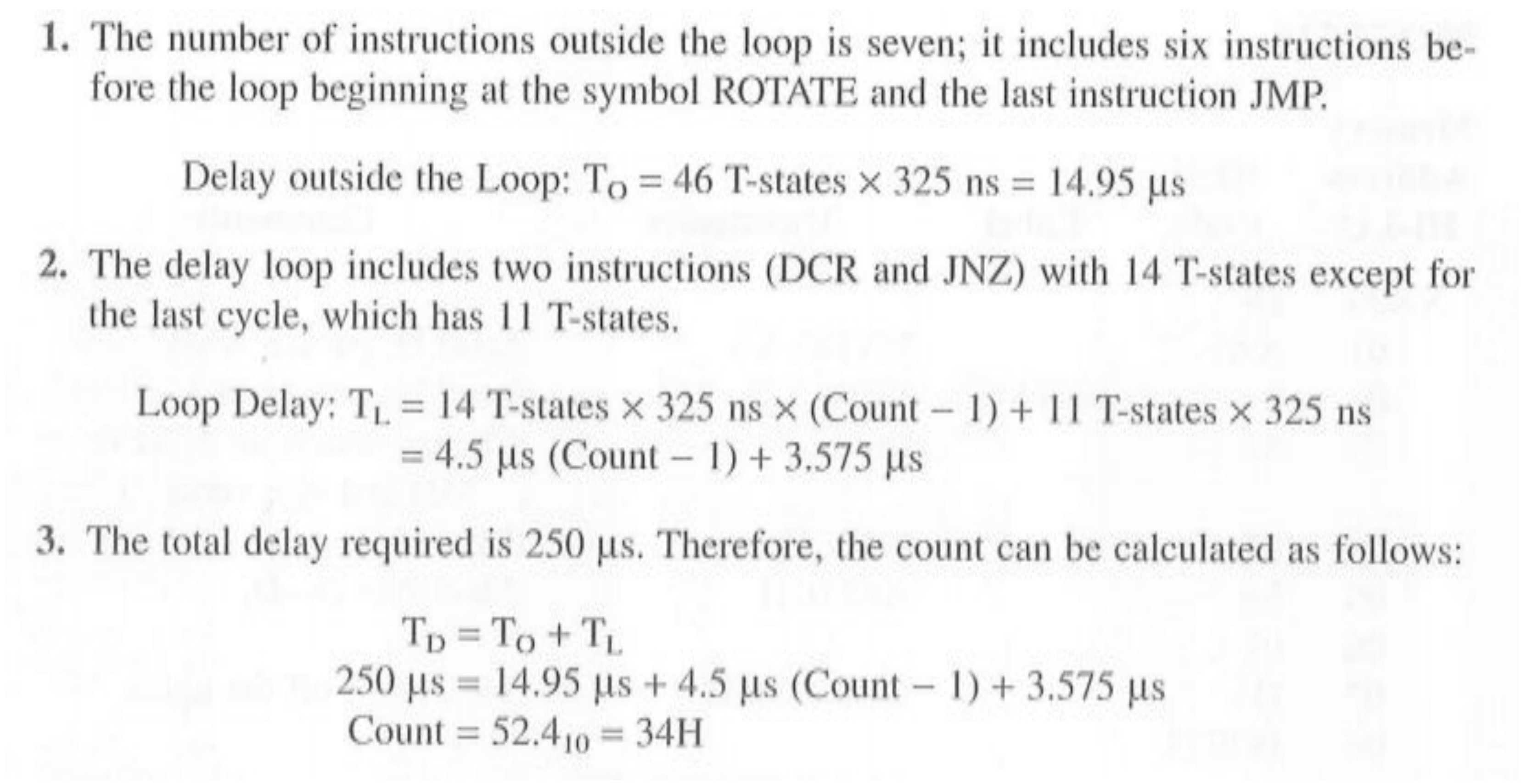
Register D is loaded with the bit pattern AAH (1010 1010), and the bit pattern is moved into the accumulator. The bit pattern is rotated left once and saved again in register D. The accumulator contents must be saved because the accumulator is used later in the program.

The next instruction, ANI, ANDs (A) to mask all but bit D_0, as illustrated below.

(A)	→1	0	1	0	1	0	1	0
After RLC	→0	1	0	1	0	1	0	1
AND with 01H	→0	0	0	0	0	0	0	1
Remaining contents	→0	0	0	0	0	0	0	1

This shows that 1 in D_0 provides a high pulse that stays on 250 μs because of the delay. In the next cycle of the loop, bit D_0 is at logic 0 because of the Rotate instruction, and the output pulse stays low for the next 250 μs.

Delay calculations



1. The number of instructions outside the loop is seven; it includes six instructions before the loop beginning at the symbol ROTATE and the last instruction JMP.

   Delay outside the Loop: $T_O = 46$ T-states $\times$ 325 ns = 14.95 µs

2. The delay loop includes two instructions (DCR and JNZ) with 14 T-states except for the last cycle, which has 11 T-states.

   Loop Delay: $T_L = 14$ T-states $\times$ 325 ns $\times$ (Count − 1) + 11 T-states $\times$ 325 ns
   = 4.5 µs (Count − 1) + 3.575 µs

3. The total delay required is 250 µs. Therefore, the count can be calculated as follows:

$$T_D = T_O + T_L$$
$$250\ \mu s = 14.95\ \mu s + 4.5\ \mu s\ (\text{Count} - 1) + 3.575\ \mu s$$
$$\text{Count} = 52.4_{10} = 34H$$