8085 Microprocessor Programs

Courtesy : www.8085projects.info

Rachit Agrawal
07-CE-52
Kalol Institute of Technology & Research Center

PROGRAMS FOR 8085 MICROPROCESSOR

PROGRAMS FOR LEARNERS

1. **Store 8-bit data in memory**
2. **Exchange the contents of memory locations**
3. **Add two 8-bit numbers**
4. **Subtract two 8-bit numbers**
5. **Add two 16-bit numbers**
6. **Add contents of two memory locations**
7. **Subtract two 16-bit numbers.**
8. **Finding one's complement of a number**
9. **Finding Two's complement of a number**
10. **Pack the unpacked BCD numbers**
11. **Unpack a BCD number**
12. **Execution format of instructions**
13. **Right shift bit of data**
14. **Left Shifting of a 16-bit data**
15. **Alter the contents of flag register in 8085**

PROGRAMS FOR BEGINNERS

16. **Calculate the sum of series of numbers**
17. **Multiply two 8-bit numbers**
18. **Divide a 16 bit number by a 8-bit number**
19. **Find the negative numbers in a block of data.**
20. **Find the largest of given numbers**
21. **Count number of one's in a number**
22. **Arrange in ascending order**
23. **Calculate the sum of series of even numbers**
24. **Calculate the sum of series of odd numbers**
25. **Find the square of given number**
26. **Search a byte in a given number**
27. **Add two decimal numbers of 6 digit each**
28. **Add each element of array with the elements of another array**
29. **Separate even numbers from given numbers**
30. **Transfer contents to overlapping memory blocks**

PROGRAMS FOR TRAINEES

31. **Add parity bit to 7-bit ASCII characters**
32. **Find the number of negative, zero and positive numbers**
33. **Inserting string in a given array of characters**
34. **Deleting string in a given array of characters**
35. **Multiply two eight bit numbers with shift and add method**
36. **Divide 16-bit number with 8-bit number using shifting technique**
37. **Sub routine to perform the task of DAA**
38. **Program to test RAM**
39. **Program to generate fibonacci number**
40. **Generate a delay of 0.4 seconds**
41. **Arrange in DESCENDING Order**
42. **Data transfer from one memory block to other memory block.**
43. **Find the factorial of a number**
44. **Find the Square Root of a given number**
45. **Split a HEX data into two nibbles and store it**

46. **Add two 4-digit BCD numbers**
47. **Subtraction of two BCD numbers**
48. **Multiply two 2-digit BCD numbers**

PROGRAMS FOR EXPERTS

a.PROGRAMS TO WORK WITH COUNTERS

49. **Generate and display binary up counter**
50. **Generate and display BCD up counter with frequency 1Hz**
51. **Generate and display BCD down counter**
52. **Generate and display the contents of decimal counter**
53. **Debug the delay routine**

b.PROGRAMS IN CODE CONVERSION

54. **2-Digit BCD to binary conversion.**
55. **Binary to BCD conversion**
56. **Find the 7-segment codes for given numbers**
57. **Find the ASCII character**
58. **ASCII to Decimal Conversion**
59. **HEX to Decimal conversion**
60. **HEX to binary conversion**

c.PROGRAMS IN INTERFACING & APPLICTIONS

I. Interfacing with IC 8251(Serial Communcation/USART)

61. **Output byte from SOD pin**
62. **Generate square wave from SOD pin**
63. **Receive ASCII character through SID pin**
64. **Transmit message using 8251**
65. **Receive message using 8251**

II. Interfacing with IC 8255(Programmable Peripheral Interface - PPI)

66. **Initialize 8255**
67. **Blink port C bit 0 of 8255**
68. **Flashing of LEDs**
69. **Traffic Light Control**
70. **Stepper Motor Control**
71. **Keyboard interface(64-key-matrix-keyboard)**
72. **Seven Segment Display Interface (Eight Digits)**

III. Interfacing with IC 8279 (Keyboard and Display Controller)

73. **8 x 8 Keyboard Interface(Without Interrupt signal)**
74. **8 x 8 Keyboard Interface(With Interrupt signal)**
75. **8 x 4 Matrix Keyboard Interface**
76. **Interfacing of eight 7-segment digits**
77. **Interfacing of 4x4 matrix keyboard and 4 digit 7 segment display**
78. **Roll a message - 'HELL0123'**
79. **Roll your NAME**

1 Statement: Store the data byte 32H into memory location 4000H.

Program 1:

```
MVI A, 52H      : Store 32H in the accumulator
STA 4000H       : Copy accumulator contents at address 4000H
HLT             : Terminate program execution
```

Program 2:

```
LXI H           : Load HL with 4000H
MVI M           : Store 32H in memory location pointed by HL register pair (4000H)
HLT             : Terminate program execution
```

The result of both programs will be the same. In program 1 direct addressing instruction is used, whereas in program 2 indirect addressing instruction is used.

2 Statement: Exchange the contents of memory locations 2000H and 4000H

Program 1:

```
    LDA 2000H     : Get the contents of memory location 2000H into accumulator
    MOV B, A      : Save the contents into B register
    LDA 4000H     : Get the contents of memory location 4000Hinto accumulator
    STA 2000H     : Store the contents of accumulator at address 2000H
    MOV A, B      : Get the saved contents back into A register
    STA 4000H     : Store the contents of accumulator at address 4000H
```

Program 2:

```
    LXI H 2000H         : Initialize HL register pair as a pointer to memory location 2000H.
    LXI D 4000H         : Initialize DE register pair as a pointer to memory location 4000H.
    MOV B, M            : Get the contents of memory location 2000H into B register.
    LDAX D              : Get the contents of memory location 4000H into A register.
    MOV M, A            : Store the contents of A register into memory location 2000H.
    MOV A, B            : Copy the contents of B register into accumulator.
    STAX D              : Store the contents of A register into memory location 4000H.
    HLT                 : Terminate program execution.
```

In Program 1, direct addressing instructions are used, whereas in Program 2, indirect addressing instructions are used.

3 Statement: Add the contents of memory locations 4000H and 4001H and place the result in memory location 4002H.

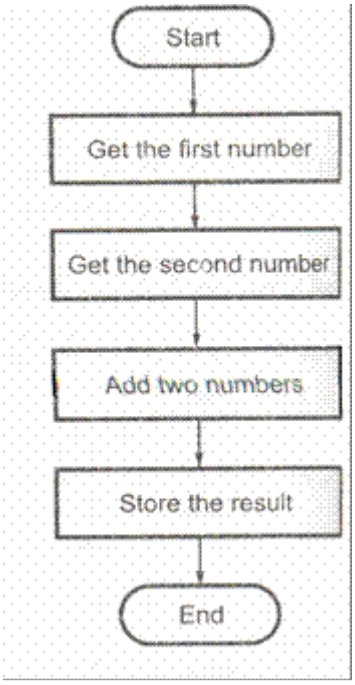
Sample problem

(4000H) = 14H
(4001H) = 89H
Result = 14H + 89H = 9DH

Source program

```
LXI H 4000H        : HL points 4000H
MOV A, M           : Get first operand
INX H              : HL points 4001H
ADD M              : Add second operand
INX H              : HL points 4002H
MOV M, A           : Store result at 4002H
HLT                : Terminate program execution
```

FLOWCHART



4 Statement: Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

Program - 4: Subtract two 8-bit numbers

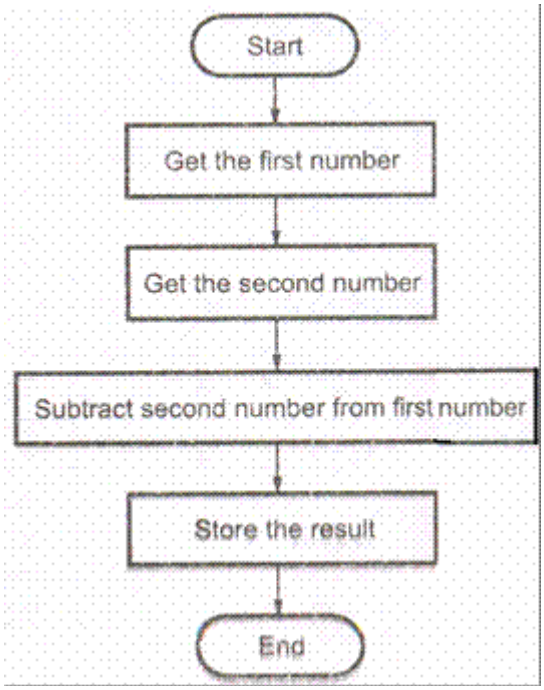
Sample problem:

(4000H) = 51H
(4001H) = 19H
Result = 51H - 19H = 38H

Source program:

```
LXI H, 4000H      : HL points 4000H
MOV A, M          : Get first operand
INX H             : HL points 4001H
SUB  M            : Subtract second operand
INX H             : HL points 4002H
MOV M, A          : Store result at 4002H.
HLT               : Terminate program execution
```

FLOWCHART



5 Statement: Add the 16-bit number in memory locations 4000H and 4001H to the 16-bit number in memory locations 4002H and 4003H. The most significant eight bits of the two numbers to be added are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H

Program - 5.a: Add two 16-bit numbers - Source Program 1

Sample problem:

(4000H) = 15H
(4001H) = 1CH
(4002H) = B7H
(4003H) = 5AH
Result = 1C15 + 5AB7H = 76CCH
(4004H) = CCH
(4005H) = 76H

Source Program 1:

```
LHLD 4000H      : Get first I6-bit number in HL
XCHG            : Save first I6-bit number in DE
LHLD 4002H      : Get second I6-bit number in HL
MOV A, E        : Get lower byte of the first number
ADD L           : Add lower byte of the second number
MOV L, A        : Store result in L register
MOV A, D        : Get higher byte of the first number
ADC H           : Add higher byte of the second number with CARRY
MOV H, A        : Store result in H register
SHLD 4004H      : Store I6-bit result in memory locations 4004H and 4005H.
HLT             : Terminate program execution
```

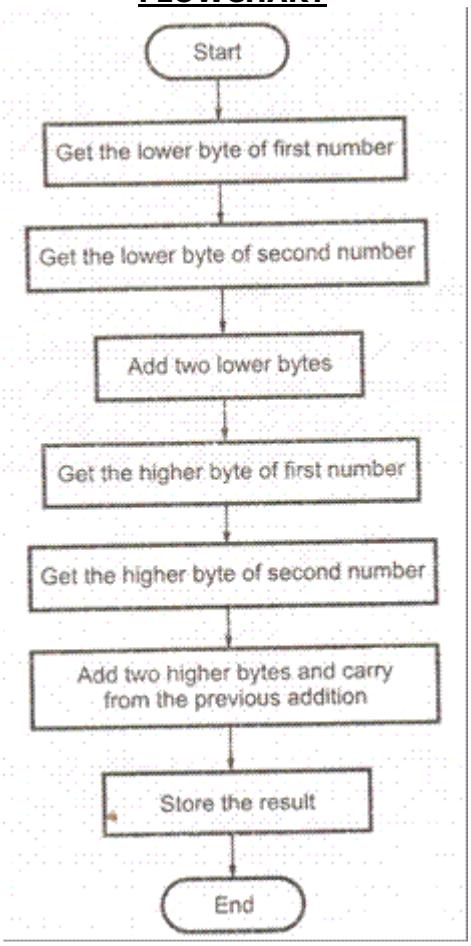
Program - 5b: Add two 16-bit numbers - Source Program 2

Source program 2:

```
LHLD 4000H      : Get first I6-bit number
XCHG            : Save first I6-bit number in DE
LHLD 4002H      : Get second I6-bit number in HL
DAD D           : Add DE and HL
SHLD 4004H      : Store I6-bit result in memory locations 4004H and 4005H.
HLT             : Terminate program execution
```

NOTE: In program 1, eight bit addition instructions are used (ADD and ADC) and addition is performed in two steps. First lower byte addition using ADD instruction and then higher byte addition using ADC instruction.In program 2, 16-bit addition instruction (DAD) is used.

FLOWCHART


Start
Get the lower byte of first number
Get the lower byte of second number
Add two lower bytes
Get the higher byte of first number
Get the higher byte of second number
Add two higher bytes and carry from the previous addition
Store the result
End

6 Statement: Add the contents of memory locations 40001H and 4001H and place the result in the memory locations 4002Hand 4003H.

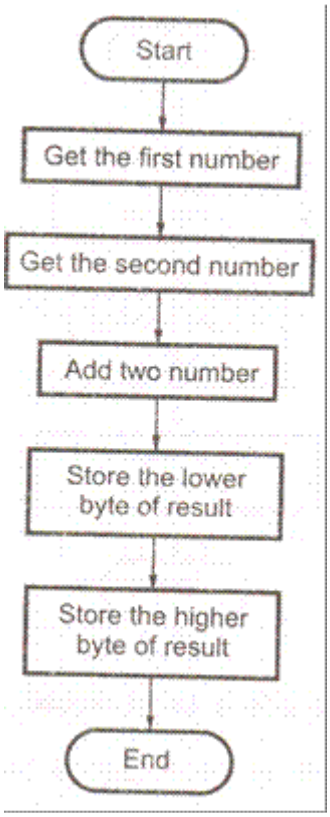
Sample problem:

(4000H) = 7FH
(400lH) = 89H
Result = 7FH + 89H = lO8H
(4002H) = 08H
(4003H) = 0lH

Source program:

```
LXI H, 4000H        :HL Points 4000H
MOV A, M            :Get first operand
INX H               :HL Points 4001H
ADD M               :Add second operand
INX H               :HL Points 4002H
MOV M, A            :Store the lower byte of result at 4002H
MVIA, 00            :Initialize higher byte result with 00H
ADC A               :Add carry in the high byte result
INX H               :HL Points 4003H
MOV M, A            :Store the higher byte of result at 4003H
HLT                 :Terminate program execution
```

FLOWCHART



7 Statement: Subtract the 16-bit number in memory locations 4002H and 4003H from the 16-bit number in memory locations 4000H and 4001H. The most significant eight bits of the two numbers are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

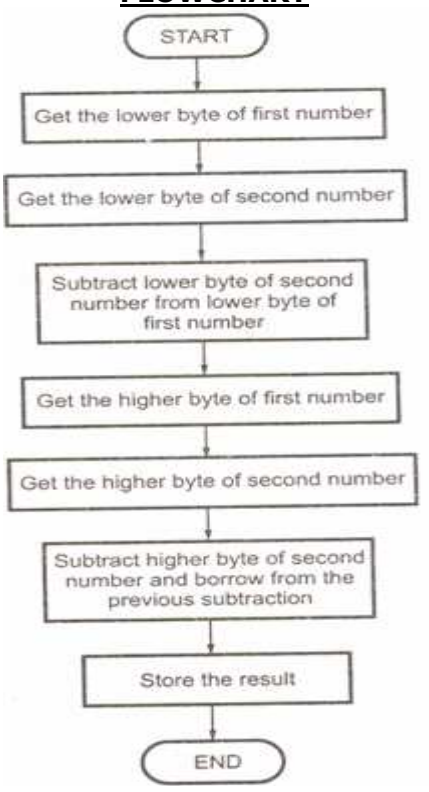
Sample problem

(4000H) = 19H
(400IH) = 6AH
(4004H) = I5H (4003H) = 5CH
Result = 6A19H - 5C15H = OE04H
(4004H) = 04H
(4005H) = OEH

Source program:

```
LHLD 4000H      : Get first 16-bit number in HL
XCHG            : Save first 16-bit number in DE
LHLD 4002H      : Get second 16-bit number in HL
MOV A, E        : Get lower byte of the first number
SUB L           : Subtract lower byte of the second number
MOV L, A        : Store the result in L register
MOV A, D        : Get higher byte of the first number
SBB H           : Subtract higher byte of second number with borrow
MOV H, A        : Store l6-bit result in memory locations 4004H and 4005H.
SHLD 4004H      : Store l6-bit result in memory locations 4004H and 4005H.
HLT             : Terminate program execution.
```

FLOWCHART

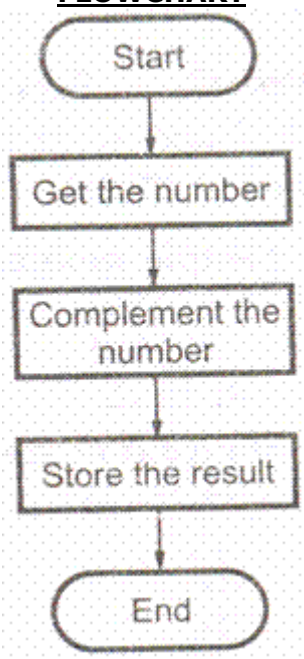


8 Statement: Find the l's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H.
Sample problem:

(4400H) = 55H
Result = (4300B) = AAB
Source program:

```
LDA 4400B          : Get the number
CMA                : Complement number
STA 4300H          : Store the result
HLT                : Terminate program execution
```

FLOWCHART



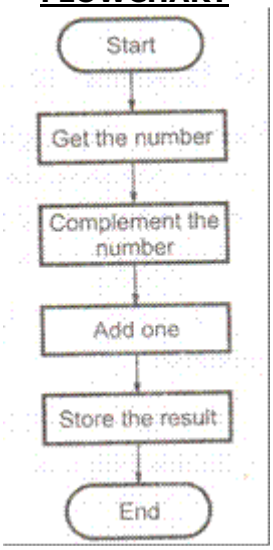
9 Statement: Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.
Sample problem:

(4200H) = 55H
Result = (4300H) = AAH + 1 = ABH

Source program:

```
LDA 4200H        : Get the number
CMA              : Complement the number
ADI, 01 H        : Add one in the number
STA 4300H        : Store the result
HLT              : Terminate program execution
```

FLOWCHART



10 Statement: Pack the two unpacked BCD numbers stored in memory locations 4200H and 4201H and store result in memory location 4300H. Assume the least significant digit is stored at 4200H.

Sample problem:

(4200H) = 04

(4201H) = 09

Result = (4300H) = 94

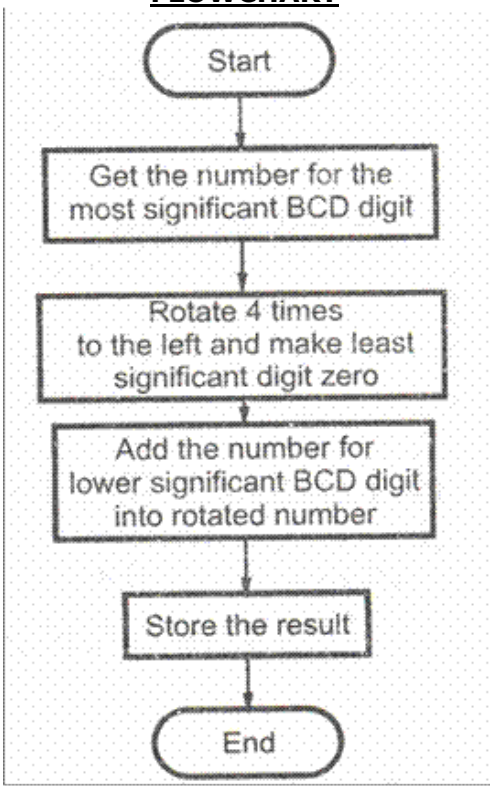
Source program

```
LDA 4201H        : Get the Most significant BCD digit
RLC
RLC
RLC
RLC              : Adjust the position of the second digit (09 is changed to 90)
ANI FOH          : Make least significant BCD digit zero
MOV C, A         : store the partial result
LDA 4200H        : Get the lower BCD digit
ADD C            : Add lower BCD digit
STA 4300H        : Store the result
HLT              : Terminate program execution
```

NOTE:

BCD NO.: The numbers "0 to 9" are called BCD (Binary Coded Decimal) numbers. A decimal number 29 can be converted into BCD number by splitting

FLOWCHART



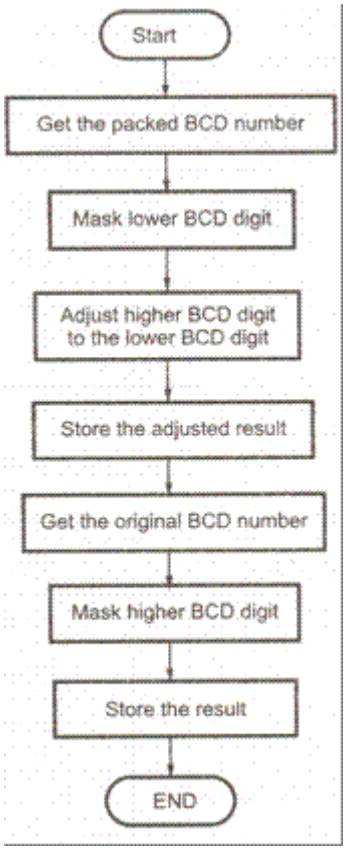
11 Statement: Two digit BCD number is stored in memory location 4200H. Unpack the BCD number and store the two digits in memory locations 4300H and 4301H such that memory location 4300H will have lower BCD digit.

Sample problem

(4200H) = 58
Result = (4300H) = 08 and
(4301H) = 05

Source program

```
LDA 4200H        : Get the packed BCD number
ANI FOH          : Mask lower nibble
RRC
RRC
RRC
RRC              : Adjust higher BCD digit as a lower digit
STA 4301H        : Store the partial result
LDA 4200H        : .Get the original BCD number
ANI OFH          : Mask higher nibble
STA 4201H        : Store the result
HLT              : Terminate program execution
```



12 Statement:Read the program given below and state the contents of all registers after the execution of each instruction in sequence.

Main program:

```
4000H    LXI SP, 27FFH
4003H    LXI H, 2000H
4006H    LXI B, 1020H
4009H    CALL SUB
400CH    HLT
```

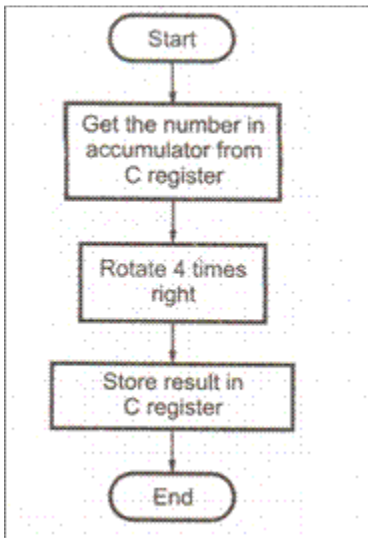
Subroutine program:

```
4100H    SUB: PUSH B
4101H    PUSH H
4102H    LXI B, 4080H
4105H    LXI H, 4090H
4108H    SHLD 2200H
4109H    DAD B
410CH    POP H
410DH    POP B
410EH    RET
```

Sr. No.	Instructions	Registers contents in Hex								Memory locations (addresses are in HEX)
		A	B	C	D	E	H	L	SP	PC	
1	LXI SP ,27FF	X	X	X	X	X	X	X	27FF	4003	
2	LXI H,2000	X	X	X	X	X	20	00	27FF	4006	
3	LXI B,1020	X	10	20	X	X	20	00	27FF	4009	
4	CALL SUB	X	10	20	X	X	20	00	27F0	4100	(27FE -- 40, 27FD –0C)
5	PUSH B	X	10	20	X	X	20	00	27FB	4101	(27Fe – 10, 27FB – 20)
6	PUSH H	X	10	20	X	X	20	00	27F9	4102	(27FA -- 20, 27F9 – 00)
7	LXI B,4080	X	40	80	X	X	20	00	27F9	4105	
8	LXI H,4090	X	40	80	X	X	40	90	27F9	4108	
9	DAD B	X	40	80	X	X	81	10	27F9	4109	
10	SHLD 2200	X	40	80	X	X	81	10	27F9	410C	(2200 --10, 2201 – 81)
11	POP H	X	40	80	X	X	20	00	27FB	4100	
12	POP B	X	10	20	X	X	20	00	27FD	410E	
13	RET	X	10	20	X	X	20	00	27FF	400C	
14	HLT	X	10	20	X	X	20	00	27FF	4000	

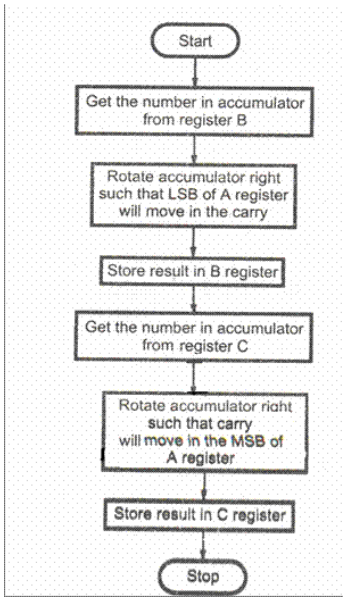
13 Statement:Write a program to shift an eight bit data four bits right. Assume that data is in register C.
Source program:

```
MOV A, C
RAR
RAR
RAR
RAR
MOV C, A
HLT
```



Statement:Write a program to shift a 16 bit data, 1 bit right. Assume that data is in BC register pair.
Source program:

```
MOV A, B
RAR
MOV B, A
MOV A, C
RAR
MOV C, A
HLT
```



14 Statement: Program to shift a 16-bit data 1 bit left. Assume data is in the HL register pair.

Source program:

DAD H : Adds HL data with HL data

HL= 1025 =0001 0000 0010 0101

```
   HL = 0001 0000 0010 0101
 + HL = 0001 0000 0010 0101
 ---------------------------
Result = 0010 0000 0100 1010
```

15 Statement: Write a set of instructions to alter the contents of flag register in 8085.

```
PUSH PSW      : Save flags on stack
POP H         : Retrieve flags in 'L'
MOV A, L      : Flags in accumulator
CMA           : Complement accumulator
MOV L, A      : Accumulator in 'L'
PUSH H        : Save on stack
POP PSW       : Back to flag register
HLT           :Terminate program execution
```

16 Statement: Calculate the sum of series of numbers. The length of the series is in memory location 4200H and the series begins from memory location 4201H.

a. Consider the sum to be 8 bit number. So, ignore carries. Store the sum at memory location 4300H.
b. Consider the sum to be 16 bit number. Store the sum at memory locations 4300H and 4301H.

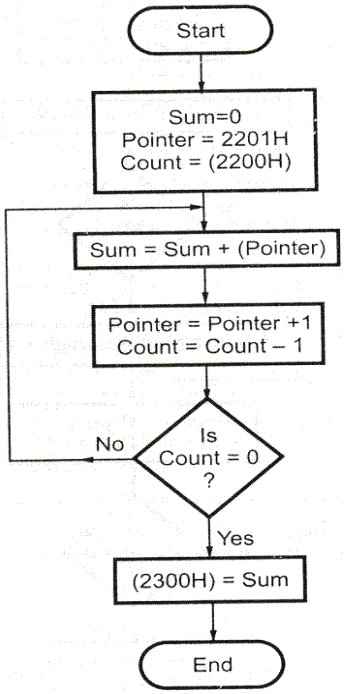
a. Sample problem

```
4200H = 04H
4201H = 10H
4202H = 45H
4203H = 33H
4204H = 22H
Result = 10 +41 + 30 + 12 =  H
4300H =  H
```

Source program:

```
LDA 4200H
MOV C, A      : Initialize counter
SUB A         : sum = 0
LXI H, 420lH  : Initialize pointer
BACK:  ADD M  : SUM = SUM + data
INX H         : increment pointer
DCR C         : Decrement counter
JNZ BACK      : if counter  0 repeat
STA 4300H     : Store sum
HLT           : Terminate program execution
```

FLOWCHART



b. Sample problem

4200H = 04H
420lH = 9AH
4202H = 52H
4203H = 89H
4204H = 3EH
Result = 9AH + 52H + 89H + 3EH = H
4300H = B3H Lower byte
4301H = 0lH Higher byte

Source program:

```
        LDA 4200H
        MOV C, A          : Initialize counter
        LXI H, 4201H          : Initialize pointer
        SUB A               :Sum low = 0
        MOV B, A          : Sum high = 0
BACK:   ADD M                 : Sum = sum + data
        JNC SKIP
        INR B               : Add carry to MSB of SUM
SKIP:   INX H                 : Increment pointer
        DCR C               : Decrement counter
        JNZ BACK          : Check if counter 0 repeat
        STA 4300H           : Store lower byte
        MOV A, B
        STA 4301H           : Store higher byte
        HLT               :Terminate program execution
```

17 Statement: Multiply two 8-bit numbers stored in memory locations 2200H and 2201H by repetitive addition and store the result in memory locations 2300H and 2301H.

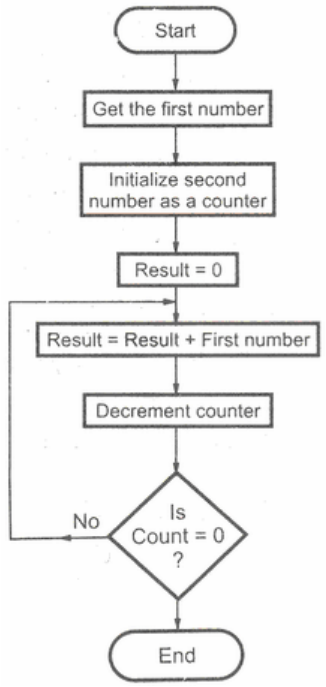
Sample problem:

(2200H) = 03H
(2201H) = B2H
Result = B2H + B2H + B2H = 216H
= 216H
(2300H) = 16H
(2301H) = 02H

Source program

```
        LDA 2200H
        MOV E, A
        MVI D, 00          : Get the first number in DE register pair
        LDA 2201H
        MOV C, A           : Initialize counter
        LX I H, 0000 H     : Result = 0
BACK:   DAD  D             : Result = result + first number
        DCR  C             : Decrement count
        JNZ  BACK          : If count  0 repeat
        SHLD 2300H         : Store result
        HLT                : Terminate program execution
```

FLOWCHART



18 Statement:Divide 16 bit number stored in memory locations 2200H and 2201H by the 8 bit number stored at memory location 2202H. Store the quotient in memory locations 2300H and 2301H and remainder in memory locations 2302H and 2303H.

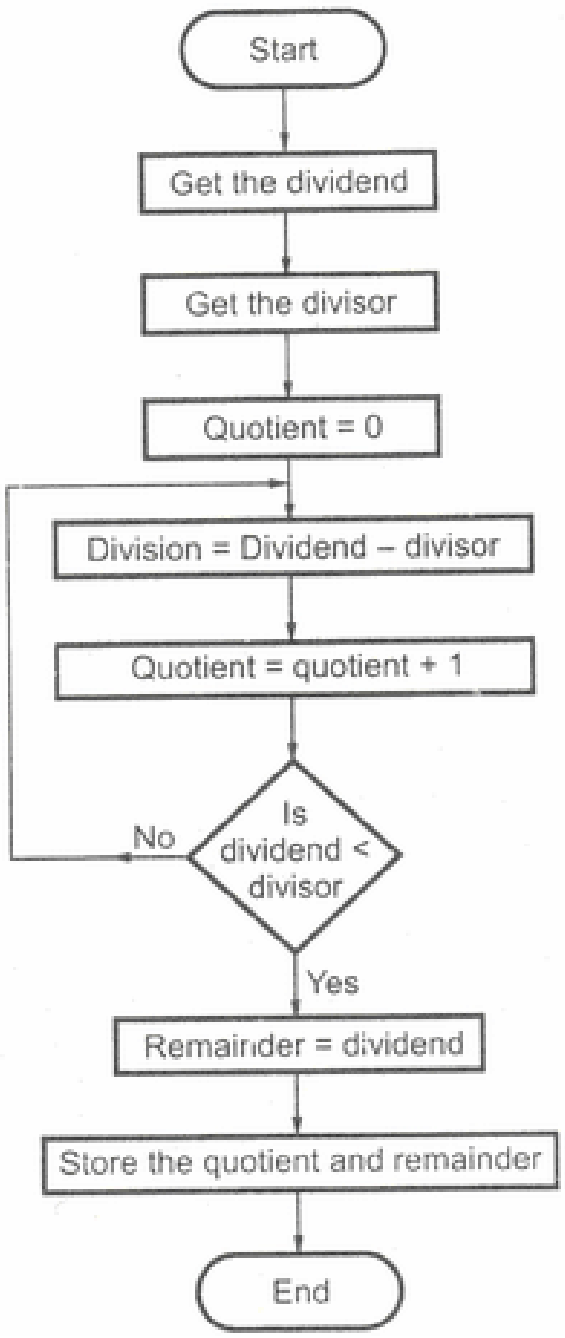
Sample problem

(2200H) = 60H
(2201H) = A0H
(2202H) = l2H
Result = A060H/12H = 8E8H Quotient and 10H remainder
(2300H) = E8H
(2301H) = 08H
(2302H= 10H
(2303H) 00H

Source program

```
LHLD 2200H          : Get the dividend
LDA 2202H           : Get the divisor
MOV C, A
LXI D, 0000H        : Quotient = 0
BACK: MOV A, L
SUB C               : Subtract divisor
MOV L, A            : Save partial result
JNC SKIP            : if CY  1 jump
DCR H               : Subtract borrow of previous subtraction
SKIP: INX D         : Increment quotient
MOV A, H
CPI, 00             : Check if dividend < divisor
JNZ BACK            : if no repeat
MOV A, L
CMP C
JNC BACK
SHLD 2302H          : Store the remainder
XCHG
SHLD 2300H          : Store the quotient
HLT                 : Terminate program execution
```

FLOWCHART


Start
Get the dividend
Get the divisor
Quotient = 0
Division = Dividend – divisor
Quotient = quotient + 1
Is dividend < divisor
No
Yes
Remainder = dividend
Store the quotient and remainder
End

19 Statement:Find the number of negative elements (most significant bit 1) in a block of data. The length of the block is in memory location 2200H and the block itself begins in memory location 2201H. Store the number of negative elements in memory location 2300H

Sample problem

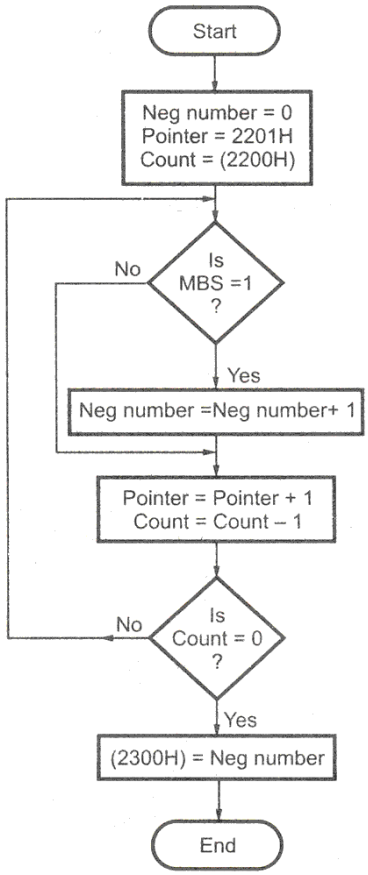
(2200H) = 04H
(2201H) = 56H
(2202H) = A9H
(2203H) = 73H
(2204H) = 82H

Result = 02 since 2202H and 2204H contain numbers with a MSB of 1.

Source program

```
        LDA 2200H
        MOV C, A          : Initialize count
        MVI B, 00         : Negative number = 0
        LXI H, 2201H      : Initialize pointer
BACK:   MOV A, M          : Get the number
        ANI 80H           : Check for MSB
        JZ SKIP           : If MSB = 1
        INR B             : Increment negative number count
SKIP:   INX H             : Increment pointer
        DCR C             : Decrement count
        JNZ BACK          : If count  0 repeat
        MOV A, B
        STA 2300H         : Store the result
        HLT               : Terminate program execution
```

FLOWCHART



20 Statement:Find the largest number in a block of data. The length of the block is in memory location 2200H and the block itself starts from memory location 2201H. Store the maximum number in memory location 2300H. Assume that the numbers in the block are all 8 bit unsigned binary numbers.

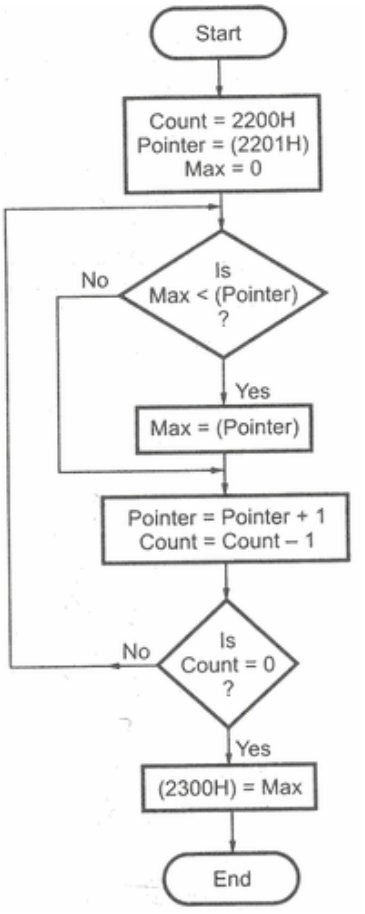
Sample problem

(2200H) = 04
(2201H) = 34H
(2202H) = A9H
(2203H) = 78H
(2204H) =56H
Result = (2202H) = A9H

Source program

```
        LDA 2200H
        MOV C, A          : Initialize counter
        XRA A             : Maximum = Minimum possible value = 0
        LXI H, 2201H      : Initialize pointer
BACK:   CMP M             : Is number> maximum
        JNC SKIP          : Yes, replace maximum
        MOV A, M
SKIP:   INX H
        DCR C
        JNZ BACK
        STA 2300H         : Store maximum number
        HLT               : Terminate program execution
```

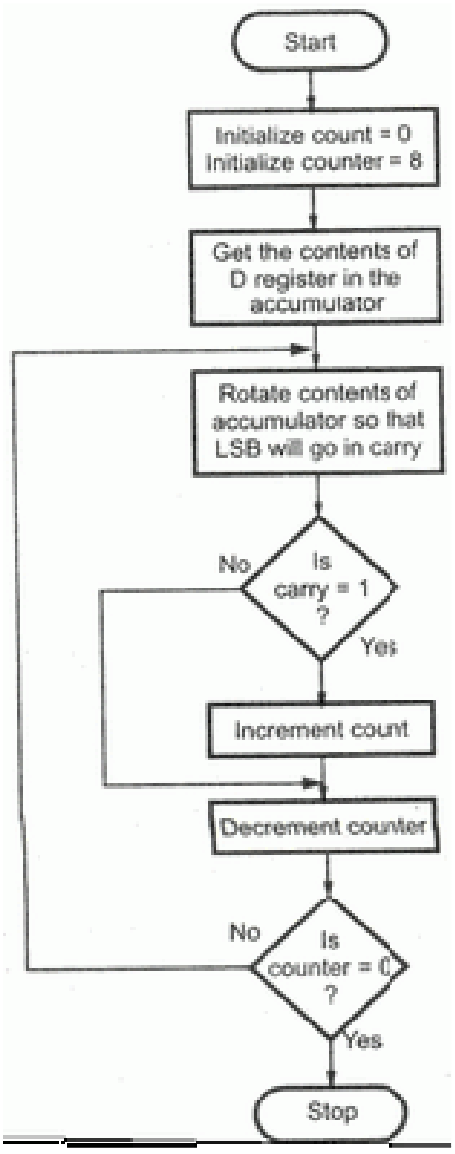
FLOWCHART



21 Statement:Write a program to count number of l's in the contents of D register and store the count in the B register.

Source program:

```
        MVI B, 00H
        MVI C, 08H
        MOV A, D
BACK:   RAR
        JNC SKIP
        INR B
SKIP:   DCR C
        JNZ BACK
        HLT
```

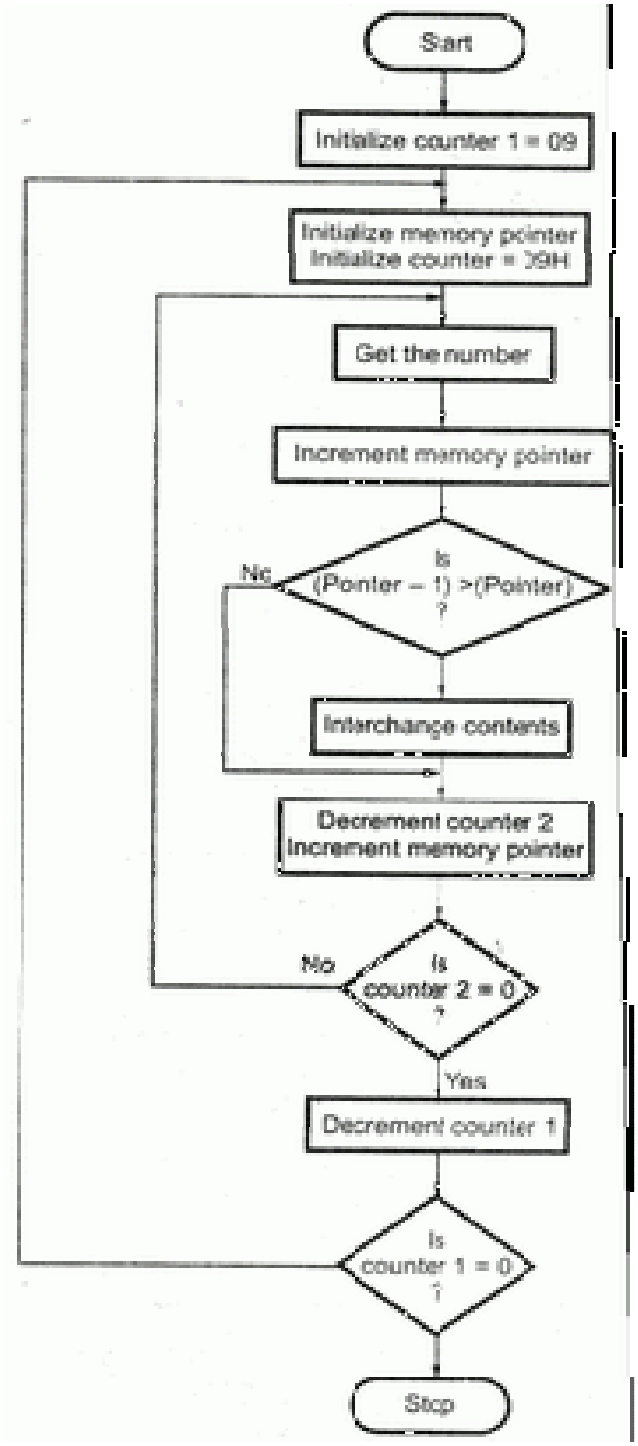


22 Statement:Write a program to sort given 10 numbers from memory location 2200H in the ascending order.

Source program:

```
        MVI B, 09           : Initialize counter
        START               : LXI H, 2200H: Initialize memory pointer
        MVI C, 09H          : Initialize counter 2
BACK:   MOV A, M            : Get the number
        INX H               : Increment memory pointer
        CMP M               : Compare number with next number
        JC SKIP             : If less, don't interchange
        JZ SKIP             : If equal, don't interchange
        MOV D, M
        MOV M, A
        DCX H
        MOV M, D
        INX H               : Interchange two numbers
SKIP:   DCR C               : Decrement counter 2
        JNZ BACK            : If not zero, repeat
        DCR B               : Decrement counter 1
        JNZ START
        HLT                 : Terminate program execution
```

FLOWCHART


Start
Initialize counter 1 = 09
Initialize memory pointer
Initialize counter = 09H
Get the number
Increment memory pointer
Is
(Pointer – 1) >(Pointer)
?
No
Interchange contents
Decrement counter 2
Increment memory pointer
Is
counter 2 = 0
?
No
Yes
Decrement counter 1
Is
counter 1 = 0
?
Stop

23 Statement:Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H.

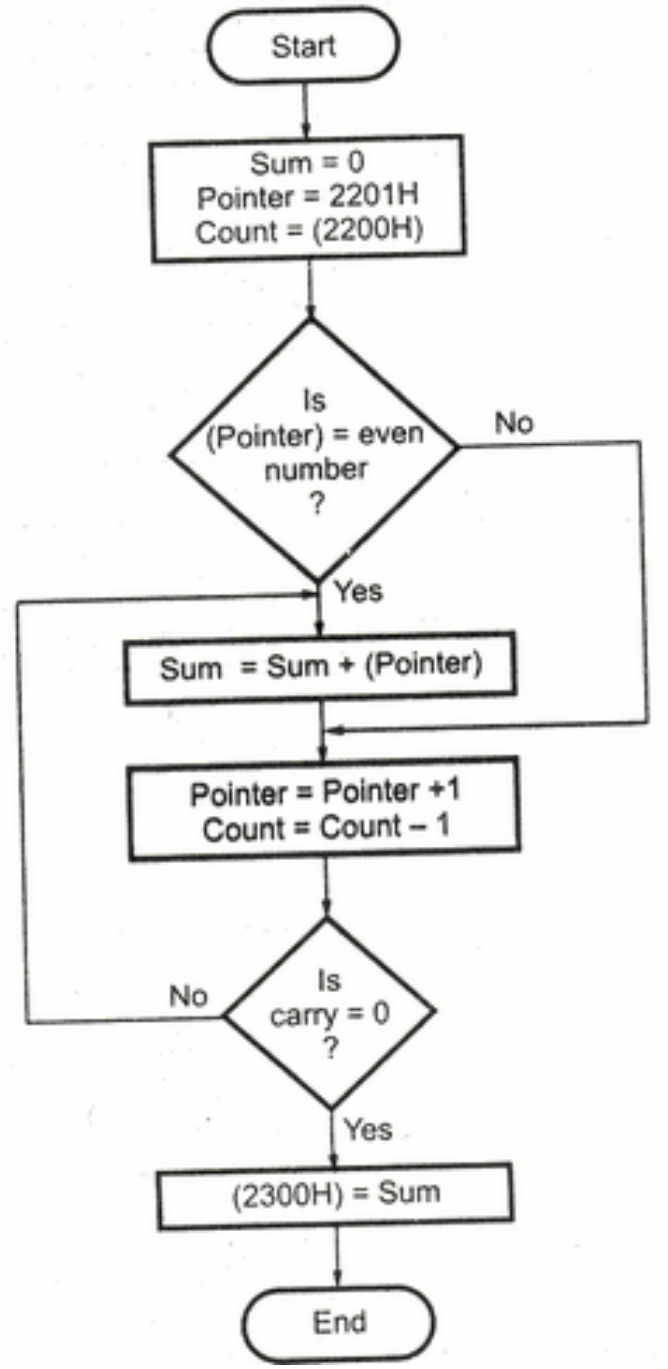
Sample problem:

2200H= 4H
2201H= 20H
2202H= l5H
2203H= l3H
2204H= 22H
Result 22l0H= 20 + 22 = 42H
= 42H

Source program:

```
        LDA 2200H
        MOV C, A          : Initialize      counter
        MVI B, 00H          : sum = 0
        LXI H, 2201H          : Initialize pointer
BACK: MOV A, M          : Get      the      number
        ANI      0lH          : Mask      Bit l      to      Bit7
        JNZ SKIP          : Don't add if number is ODD
        MOV A, B          : Get      the      sum
        ADD      M          : SUM      = SUM      + data
        MOV B, A          : Store      result      in      B      register
SKIP: INX H                : increment pointer
        DCR      C          : Decrement      counter
        JNZ      BACK          : if counter  0 repeat
        STA      2210H          : store      sum
        HLT                : Terminate program execution
```

FLOWCHART


Start
Sum = 0
Pointer = 2201H
Count = (2200H)
Is
(Pointer) = even
number
?
No
Yes
Sum = Sum + (Pointer)
Pointer = Pointer +1
Count = Count – 1
Is
carry = 0
?
No
Yes
(2300H) = Sum
End

24 Statement:Calculate the sum of series of odd numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 16-bit. Store the sum at memory locations 2300H and 2301H.

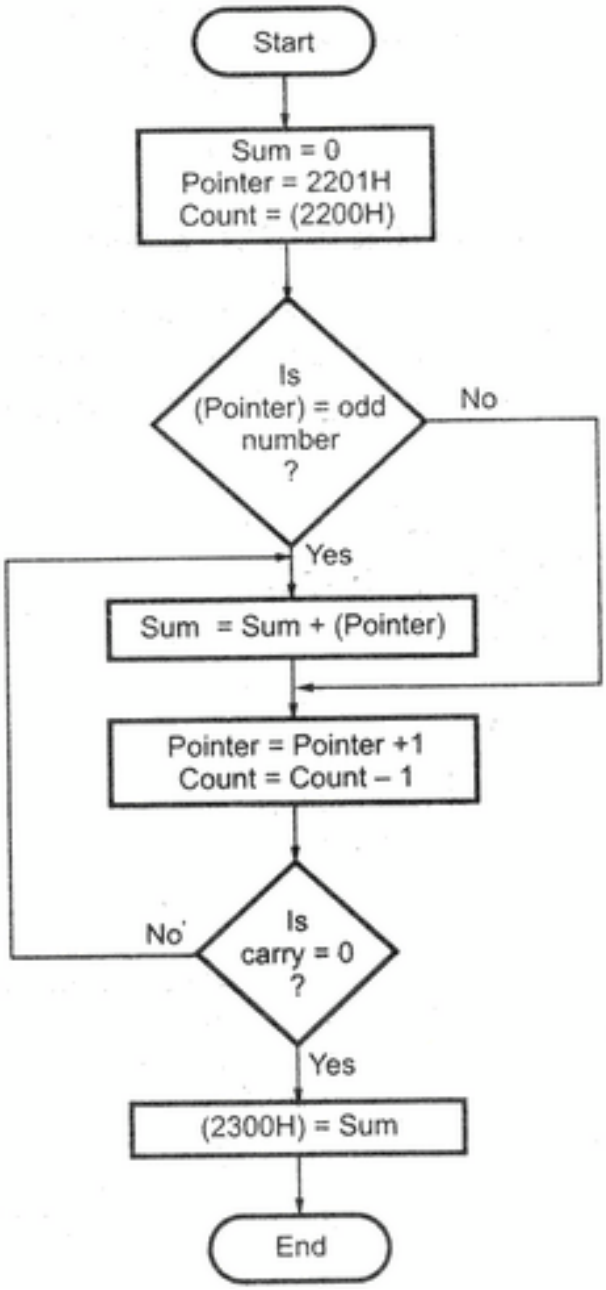
Sample problem:

2200H = 4H
2201H= 9AH
2202H= 52H
2203H= 89H
2204H= 3FH
Result = 89H + 3FH = C8H
2300H= H Lower byte
2301H = H Higher byte

Source program

```
        LDA 2200H
        MOV C, A          : Initialize counter
        LXI H, 2201H          : Initialize pointer
        MVI E, 00         : Sum low = 0
        MOV D, E          : Sum high = 0
BACK:   MOV A, M            : Get the number
        ANI 0lH           : Mask Bit 1 to Bit7
        JZ SKIP           : Don't add if number is even
        MOV A, E          : Get the lower byte of sum
        ADD M           : Sum = sum + data
        MOV E, A            : Store result in E register
        JNC SKIP
        INR D                 : Add carry to MSB of SUM
SKIP:   INX H                   : Increment pointer
```

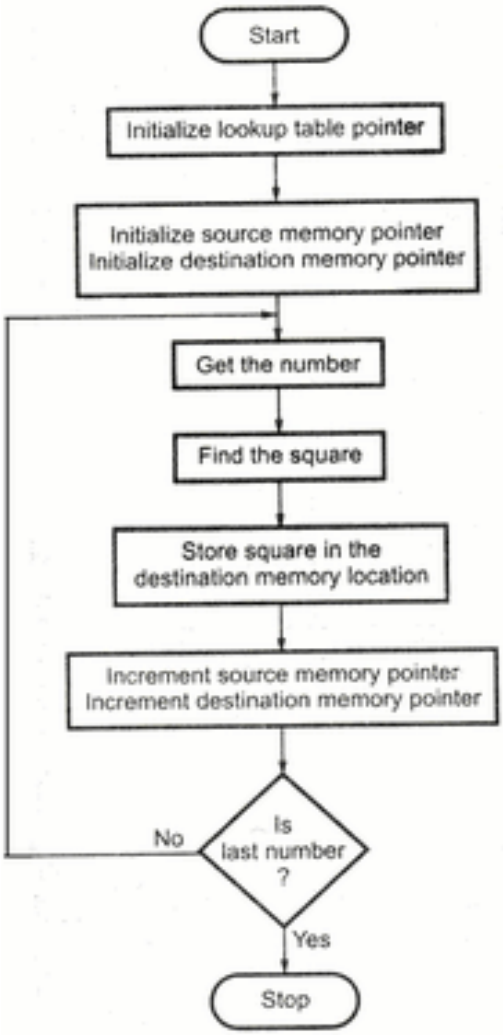
FLOWCHART


Start
Sum = 0
Pointer = 2201H
Count = (2200H)
Is
(Pointer) = odd
number
?
No
Yes
Sum = Sum + (Pointer)
Pointer = Pointer +1
Count = Count – 1
Is
carry = 0
?
No
Yes
(2300H) = Sum
End

25 Statement:Find the square of the given numbers from memory location 6100H and store the result from memory location 7000H.

Source Program:

```
        LXI H, 6200H      : Initialize lookup table pointer
        LXI D, 6100H          : Initialize source memory pointer
        LXI B, 7000H      : Initialize destination memory pointer
BACK:   LDAX D             : Get the number
        MOV L, A           : A point to the square
        MOV A, M           : Get the square
        STAX B           : Store the result at destination memory location
        INX D          : Increment source memory pointer
        INX B              : Increment destination memory pointer
        MOV A, C
        CPI 05H          : Check for last number
        JNZ BACK           : If not repeat
        HLT                : Terminate program execution
```

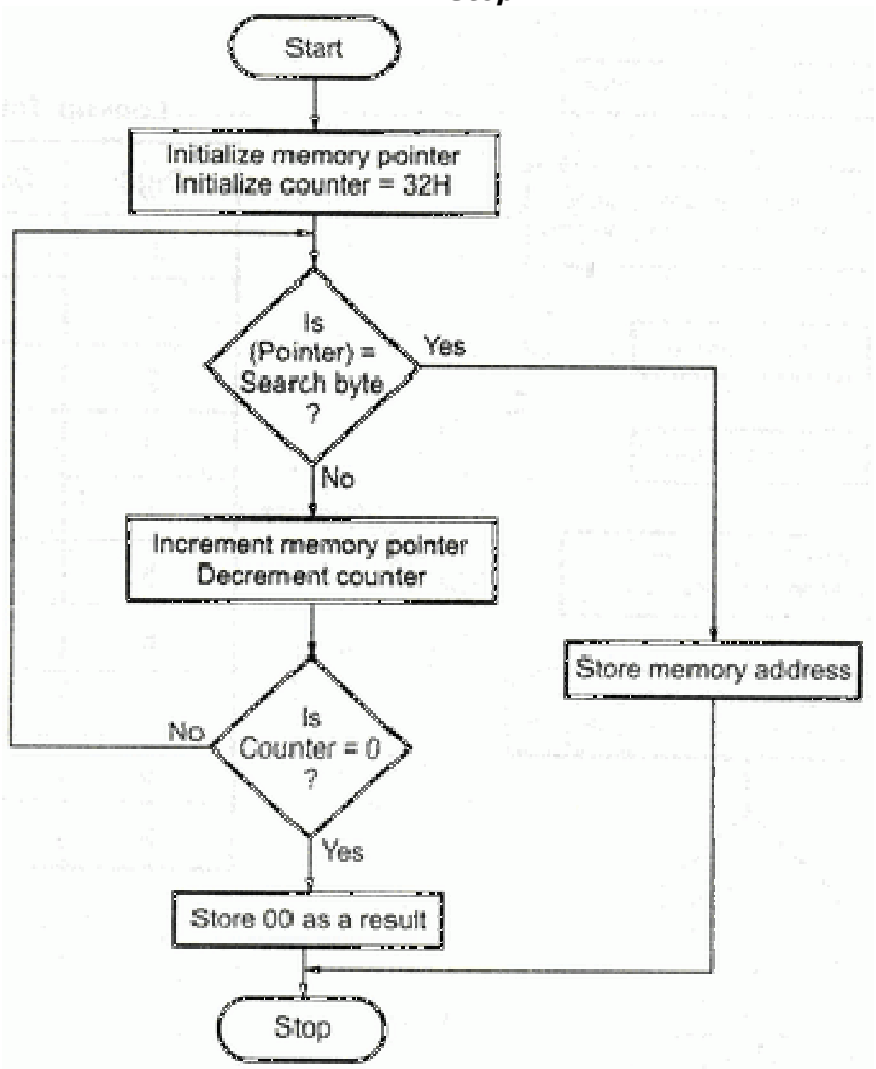


Lookup Table

Digit	Square
0	0H
1	1H
2	4H
3	9H
4	10H
5	19H
6	24H
7	31H
8	40H
9	51H

26 Statement: Search the given byte in the list of 50 numbers stored in the consecutive memory locations and store the address of memory location in the memory locations 2200H and 2201H. Assume byte is in the C register and starting address of the list is 2000H. If byte is not found store 00 at 2200H and 2201H.
Source program:

```
        LX I H, 2000H       : Initialize memory pointer 52H
        MVI B, 52H            : Initialize counter
BACK: MOV A, M              : Get the number
        CMP C             : Compare with the given byte
        JZ LAST             : Go last if match occurs
        INX H             : Increment memory pointer
        DCR B                 : Decrement counter
        JNZ B           : I f not zero, repeat
        LXI H, 0000H
        SHLD 2200H
        JMP END           : Store 00 at 2200H and 2201H
LAST: SHLD 2200H              : Store memory address
END:  HLT                   : Stop
```

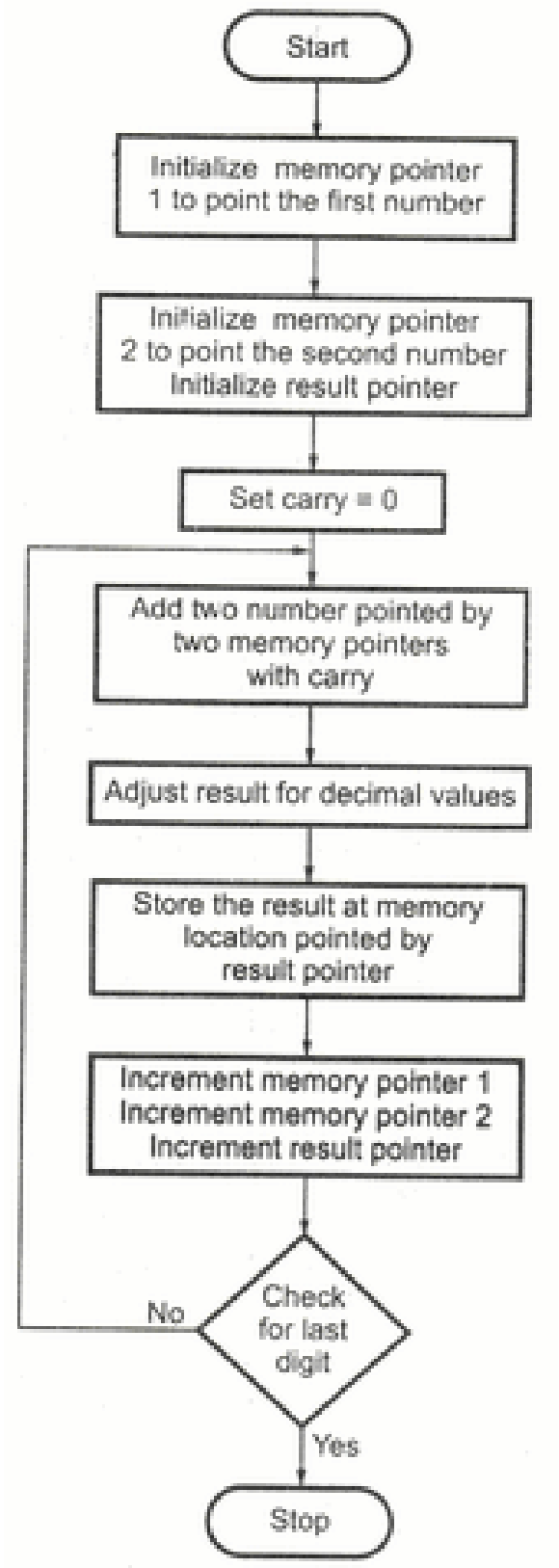


27 Statement: Two decimal numbers six digits each, are stored in BCD package form. Each number occupies a sequence of byte in the memory. The starting address of first number is 6000H Write an assembly language program that adds these two numbers and stores the sum in the same format starting from memory location 6200H.

Source Program:

```
        LXI H, 6000H          : Initialize pointer l to first number
        LXI D, 6l00H          : Initialize pointer2 to second number
        LXI B, 6200H    : Initialize pointer3 to result
        STC
        CMC                   : Carry = 0
BACK:   LDAX D                : Get the digit
        ADD M                 : Add two digits
        DAA                   : Adjust for decimal
        STAX.B                : Store the result
        INX H                 : Increment pointer 1
        INX D                 : Increment pointer2
        INX B                 : Increment result pointer
        MOV A, L
        CPI 06H               : Check for last digit
        JNZ BACK              : If not last digit repeat
        HLT                   : Terminate program execution
```

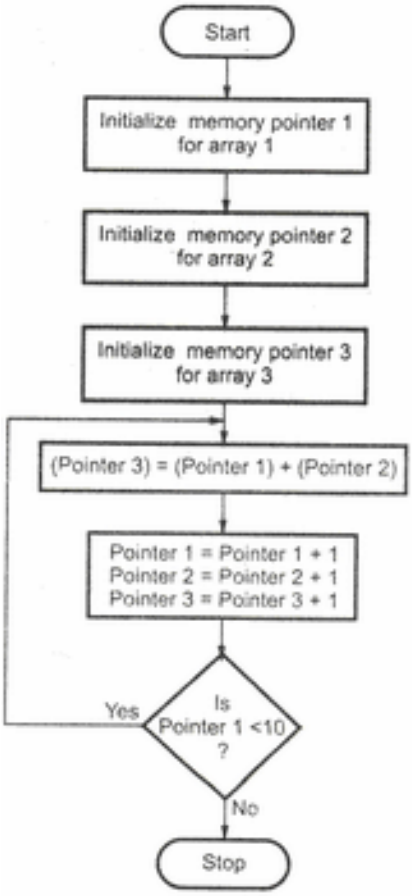
FLOWCHART


Start
Initialize memory pointer 1 to point the first number
Initialize memory pointer 2 to point the second number
Initialize result pointer
Set carry = 0
Add two number pointed by two memory pointers with carry
Adjust result for decimal values
Store the result at memory location pointed by result pointer
Increment memory pointer 1
Increment memory pointer 2
Increment result pointer
Check for last digit
No
Yes
Stop

28 Statement: Add 2 arrays having ten 8-bit numbers each and generate a third array of result. It is necessary to add the first element of array 1 with the first element of array-2 and so on. The starting addresses of array l, array2 and array3 are 2200H, 2300H and 2400H, respectively.

Source Program:

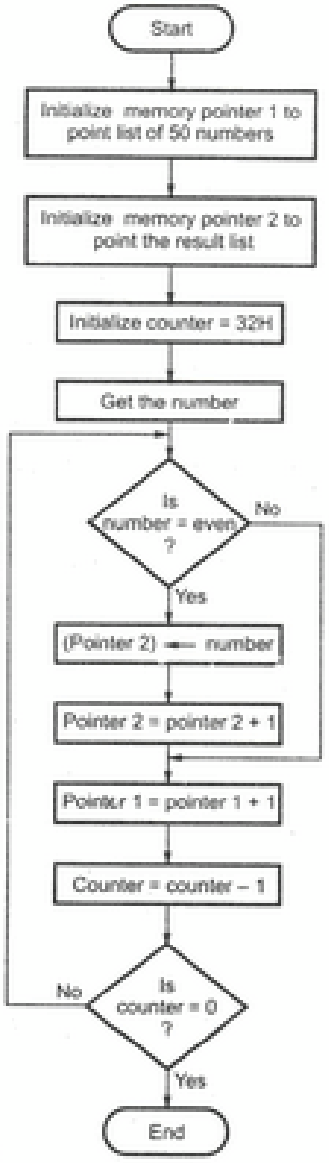
```
        LXI H, 2200H        : Initialize memory pointer 1
        LXI B, 2300H        : Initialize memory pointer 2
        LXI D, 2400H        : Initialize result pointer
BACK:   LDAX B              : Get the number from array 2
        ADD M               : Add it with number in array 1
        STAX D              : Store the addition in array 3
        INX H               : Increment pointer 1
        INX B               : Increment pointer2
        INX D               : Increment result pointer
        MOV A, L
        CPI 0AH             : Check pointer 1 for last number
        JNZ BACK            : If not, repeat
        HLT                 : Stop
```



29 Statement: Write an assembly language program to separate even numbers from the given list of 50 numbers and store them in the another list starting from 2300H. Assume starting address of 50 number list is 2200H.

Source Program:

```
        LXI H, 2200H        : Initialize memory pointer l
        LXI D, 2300H        : Initialize memory pointer2
        MVI C, 32H          : Initialize counter
BACK:MOV A, M               : Get the number
        ANI 0lH             : Check for even number
        JNZ SKIP            : If ODD, don't store
        MOV A, M            : Get the number
        STAX       D        : Store the number in result list
        INX D               : Increment pointer 2
SKIP: INX H                 : Increment pointer l
        DCR C               : Decrement counter
        JNZ BACK            : If not zero, repeat
        HLT                 : Stop
```



30 Statement: Write assembly language program with proper comments for the following:

A block of data consisting of 256 bytes is stored in memory starting at 3000H. This block is to be shifted (relocated) in memory from 3050H onwards. Do not shift the block or part of the block anywhere else in the memory.

Source Program:

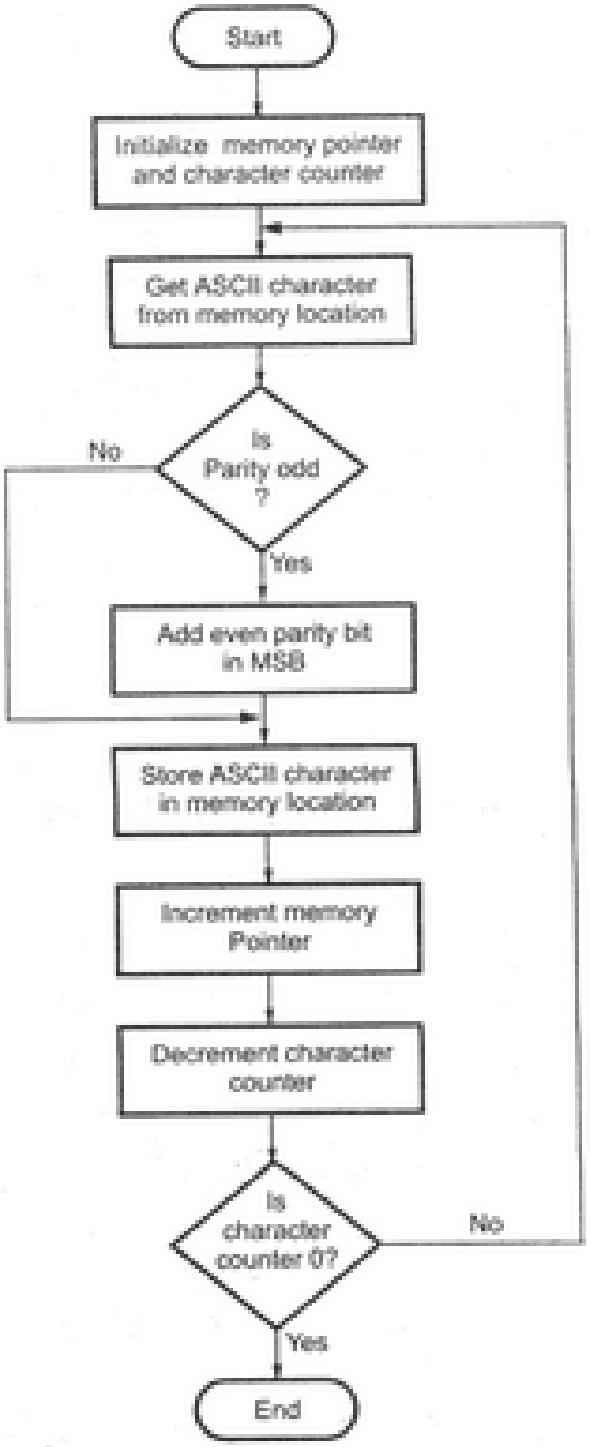
Two blocks (3000 - 30FF and 3050 - 314F) are overlapping. Therefore it is necessary to transfer last byte first and first byte last.

```
    MVI C, FFH          : Initialize counter
    LX I H, 30FFH       : Initialize source memory pointer 3l4FH
    LXI D, 314FH        : Initialize destination memory pointer
BACK: MOV A, M          : Get byte from source memory block
    STAX D              : Store byte in the destination memory block
    DCX H               : Decrement source memory pointer
    DCX                 : Decrement destination memory pointer
    DCR C               : Decrement counter
    JNZ BACK            : If counter   0 repeat
    HLT                 : Stop execution
```

31 Statement: Add even parity to a string of 7-bit ASCII characters. The length of the string is in memory location 2040H and the string itself begins in memory location 2041H. Place even parity in the most significant bit of each character.

Source Program:

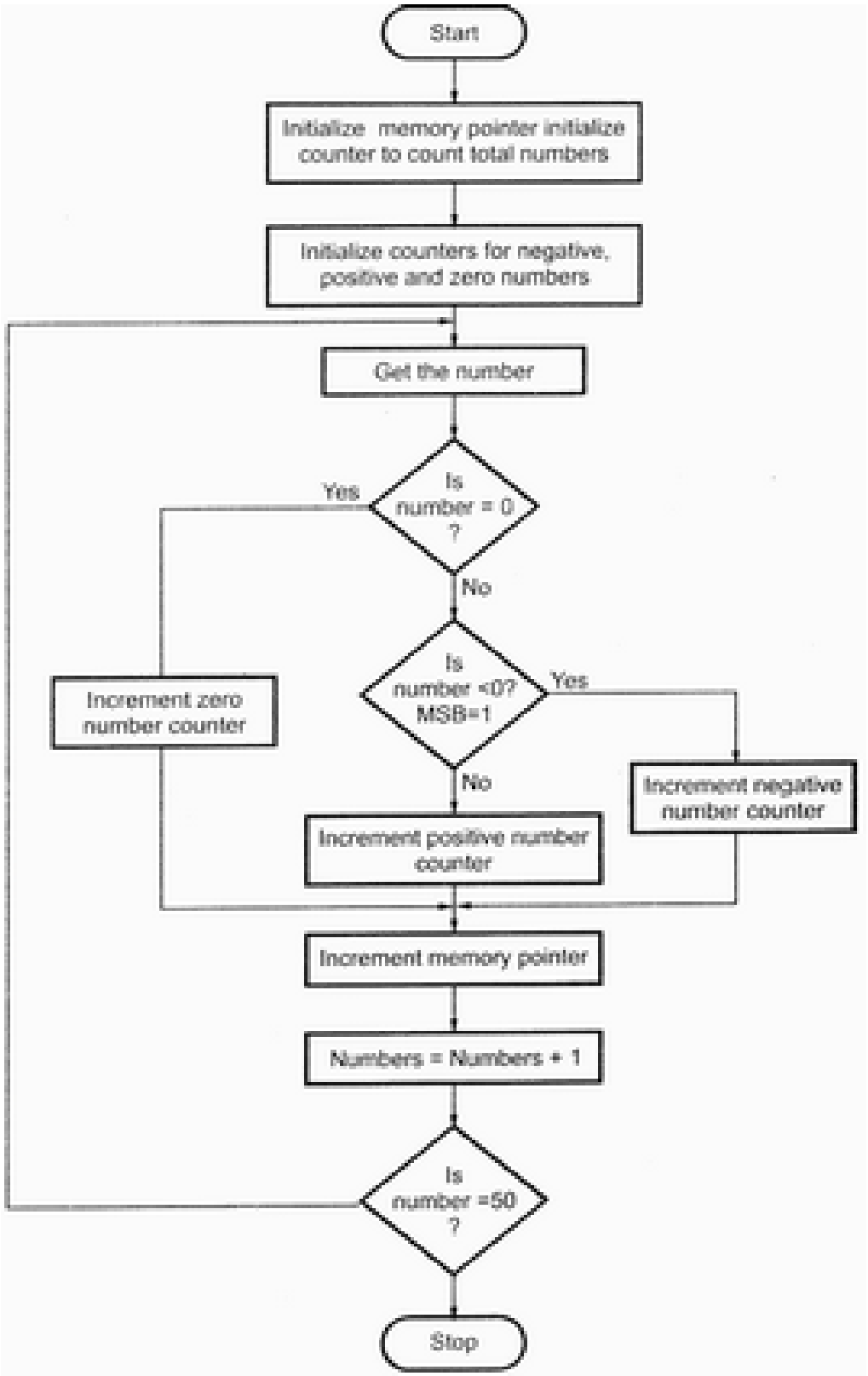
```
    LXI H, 2040H
    MOV C ,M            : Counter for character
REPEAT:INX H            : Memory pointer to character
    MOV A,M             : Character in accumulator
    ORA A               : ORing with itself to check parity.
    JPO PAREVEN         : If odd parity place
    ORI 80H             even parity in D7 (80).
PAREVEN:MOV M , A       : Store converted even parity character.
    DCR C               : Decrement counter.
    JNZ REPEAT          : If not zero go for next character.
    HLT                 : Terminate program execution
```


Start
Initialize memory pointer and character counter
Get ASCII character from memory location
Is Parity odd ?
No
Yes
Add even parity bit in MSB
Store ASCII character in memory location
Increment memory Pointer
Decrement character counter
Is character counter 0?
No
Yes
End

32 Statement: A list of 50 numbers is stored in memory, starting at 6000H. Find number of negative, zero and positive numbers from this list and store these results in memory locations 7000H, 7001H, and 7002H respectively.

Source Program:

```
      LXI H, 6000H          : Initialize memory pointer
      MVI C, 00H            : Initialize number counter
      MVI B, 00H            : Initialize negative number counter
      MVI E, 00H            : Initialize zero number counter
BEGIN:MOV A, M              : Get the number
      CPI 00H               : If number = 0
      JZ ZERONUM            : Goto zeronum
      ANI 80H               : If MSB of number = 1i.e. if
      JNZ NEGNUM            : number is negative goto NEGNUM
      INR D                 : otherwise increment positive number counter
      JMP LAST
ZERONUM:INR E               : Increment zero number counter
      JMP LAST
NEGNUM:INR B                : Increment negative number counter
LAST:INX H                  : Increment memory pointer
      INR C                 : Increment number counter
      MOV A, C
      CPI 32H               : If number counter = 5010 then
      JNZ BEGIN             : Store otherwise check next number
      LXI H, 7000           : Initialize memory pointer.
      MOV M, B              : Store negative number.
      INX H
      MOV M, E              : Store zero number.
      INX H
      MOV M, D              : Store positive number.
      HLT                   : Terminate execution
```


Start
Initialize memory pointer initialize counter to count total numbers
Initialize counters for negative, positive and zero numbers
Get the number
Is number = 0 ?
Yes
No
Is number <0? MSB=1
Yes
No
Increment zero number counter
Increment negative number counter
Increment positive number counter
Increment memory pointer
Numbers = Numbers + 1
Is number =50 ?
Stop

33 Statement:Write an 8085 assembly language program to insert a string of four characters from the tenth location in the given array of 50 characters.

Solution:

Step 1: Move bytes from location 10 till the end of array by four bytes downwards.

Step 2: Insert four bytes at locations 10, 11, 12 and 13.

Source Program:

```
LXI H, 2l31H          : Initialize pointer at the last location of array.
LXI D, 2l35H          : Initialize another pointer to point the last location of
array after insertion.
AGAIN: MOV A, M       : Get the character
STAX D          : Store at the new location
DCX D           : Decrement destination pointer
DCX H           : Decrement source pointer
MOV A, L        : [check whether desired
CPI 05H         bytes are shifted or not]
JNZ AGAIN       : if not repeat the process
INX H           : adjust the memory pointer
LXI D, 2200H    : Initialize the memory pointer to point the string to be
inserted
REPE: LDAX D    : Get the character
MOV M, A        : Store it in the array
INX D           : Increment source pointer
INX H           : Increment destination pointer
MOV A, E        : [Check whether the 4 bytes
CPI 04          are inserted]
JNZ REPE        : if not repeat the process
HLT             : stop
```

34 Statement:Write an 8085 assembly language program to delete a string of 4 characters from the tenth location in the given array of 50 characters.

Solution: Shift bytes from location 14 till the end of array upwards by 4 characters i.e. from location 10 onwards.

Source Program:

```
LXI H, 2l0DH          :Initialize source memory pointer at the 14thlocation of the
array.
LXI D, 2l09H          : Initialize destn memory pointer at the 10th location of the
array.
MOV A, M        : Get the character
STAX D          : Store character at new location
INX D           : Increment destination pointer
INX H           : Increment source pointer
MOV A, L        : [check whether desired
CPI 32H         bytes are shifted or not]
JNZ REPE        : if not repeat the process
HLT             : stop
```

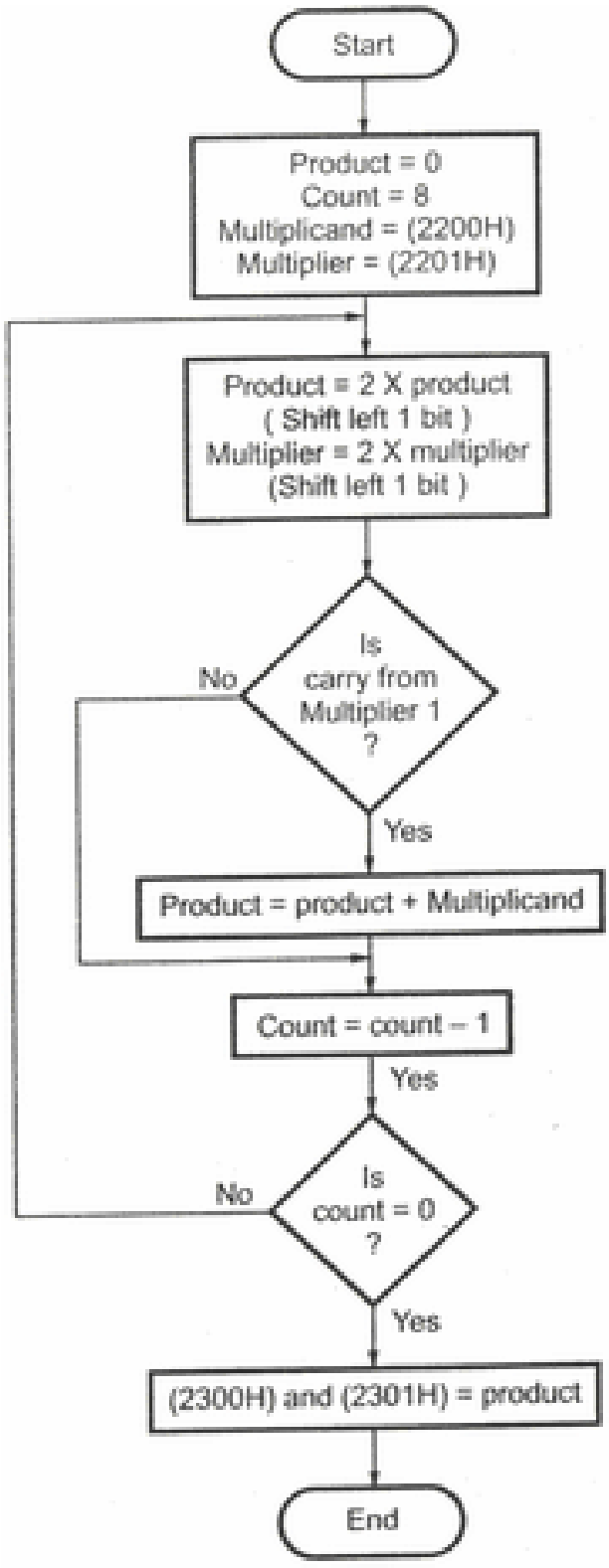
35 Statement:Multiply the 8-bit unsigned number in memory location 2200H by the 8-bit unsigned number in memory location 2201H. Store the 8 least significant bits of the result in memory location 2300H and the 8 most significant bits in memory location 2301H.

Sample problem:

(2200) = 1100 (0CH)
(2201) = 0101 (05H)
Multiplicand = 1100 (1210)
Multiplier = 0101 (510)
Result = 12 x 5 = (6010)

Source program

```
        LXI H, 2200        : Initialize the memory pointer
        MOV E, M           : Get multiplicand
        MVI D, 00H         : Extend to 16-bits
        INX H              : Increment memory pointer
        MOV A, M           : Get multiplier
        LXI H, 0000        : Product = 0
        MVI B, 08H         : Initialize counter with count 8
MULT:   DAD H              : Product = product x 2
        RAL
        JNC SKIP           : Is carry from multiplier 1 ?
        DAD D              : Yes, Product =Product + Multiplicand
SKIP:   DCR B              : Is counter = zero
        JNZ MULT           : no, repeat
        SHLD 2300H         : Store the result
        HLT                : End of program
```

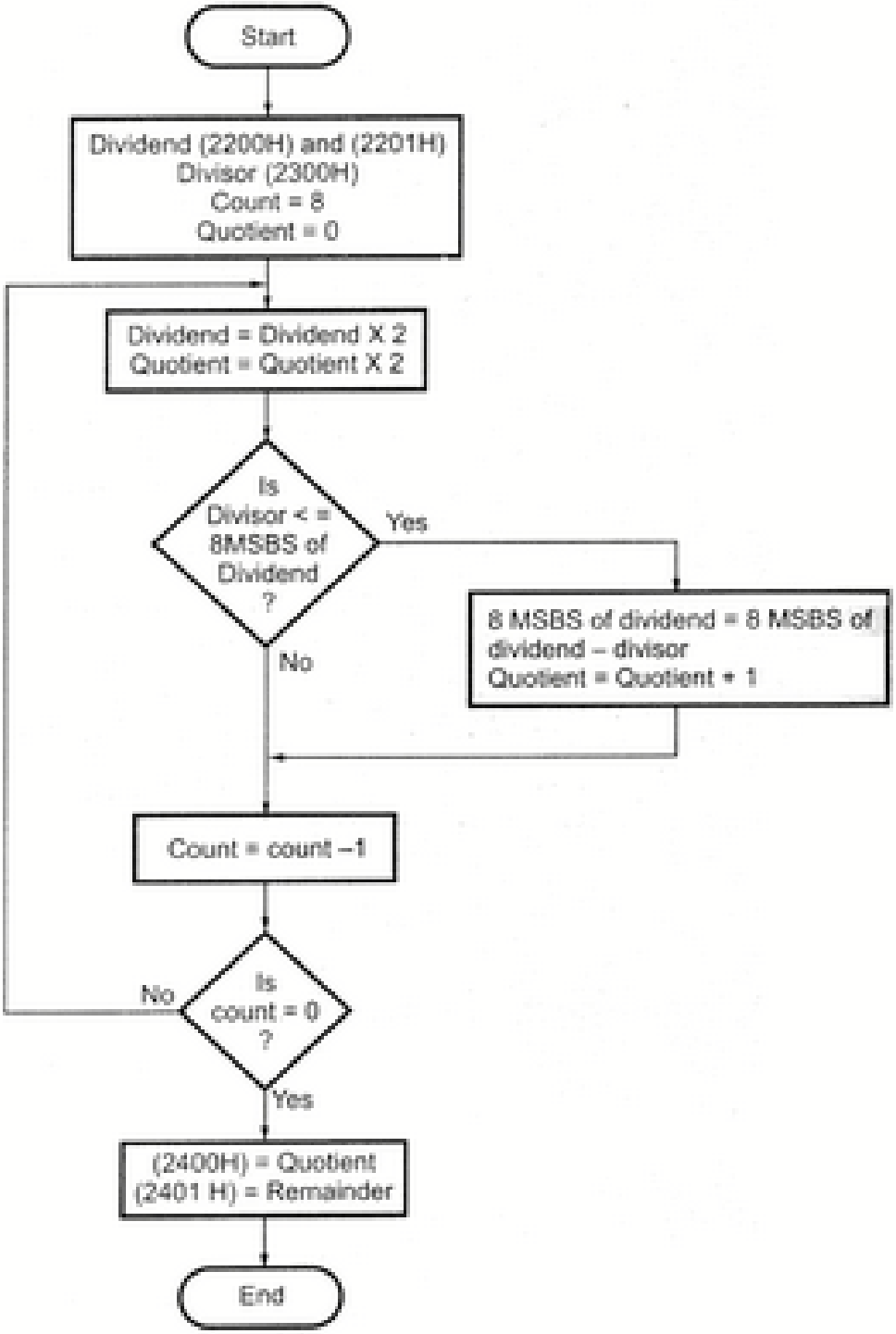

Start
Product = 0
Count = 8
Multiplicand = (2200H)
Multiplier = (2201H)
Product = 2 X product
(Shift left 1 bit)
Multiplier = 2 X multiplier
(Shift left 1 bit)
Is
carry from
Multiplier 1
?
No
Yes
Product = product + Multiplicand
Count = count – 1
Yes
Is
count = 0
?
No
Yes
(2300H) and (2301H) = product
End

36 Statement:Divide the 16-bit unsigned number in memory locations 2200H and 2201H (most significant bits in 2201H) by the B-bit unsigned number in memory location 2300H store the quotient in memory location 2400H and remainder in 2401H.

Assumption: The most significant bits of both the divisor and dividend are zero.

Source program

```
      MVI E, 00          : Quotient = 0
      LHLD    2200H           : Get dividend
      LDA 2300           : Get divisor
      MOV B, A           : Store     divisor
      MVI C, 08          : Count = 8
NEXT: DAD H              : Dividend = Dividend x 2
      MOV A, E
      RLC
      MOV E, A           : Quotient = Quotient x 2
      MOV A, H
      SUB B                : Is most significant byte of Dividend > divisor
      JC SKIP            : No, go to Next step
      MOV H, A           : Yes, subtract divisor
      INR E                : and Quotient = Quotient + 1
SKIP:DCR C                 : Count = Count - 1
      JNZ NEXT           : Is count =0 repeat
      MOV A, E
      STA 2401H            : Store Quotient
      Mov A, H
      STA 2410H            : Store remainder
      HLT                : End of program.
```


Start
Dividend (2200H) and (2201H)
Divisor (2300H)
Count = 8
Quotient = 0
Dividend = Dividend X 2
Quotient = Quotient X 2
Is Divisor < = 8MSBS of Dividend ?
Yes
No
8 MSBS of dividend = 8 MSBS of dividend – divisor
Quotient = Quotient + 1
Count = count –1
Is count = 0 ?
No
Yes
(2400H) = Quotient
(2401 H) = Remainder
End

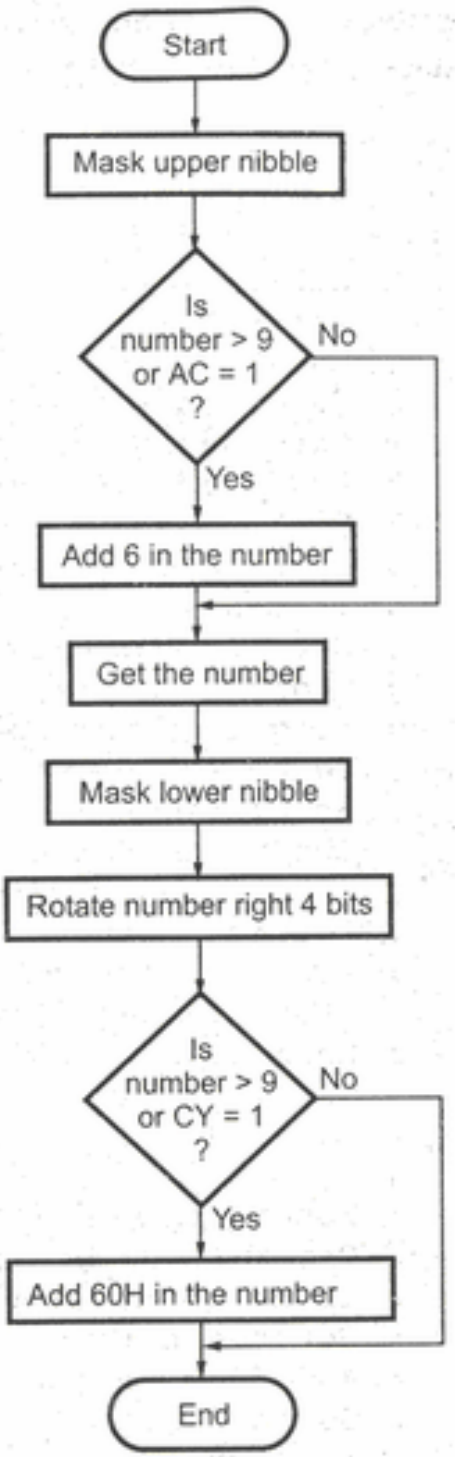
37 Statement:Assume the DAA instruction is not present. Write a sub routine which will perform the same task as DAA.
Sample Problem:

Execution of DAA instruction:
1. If the value of the low order four bits (03-00) in the accumulator is greater than 9 or if auxiliary carry flag is set, the instruction adds 6 '(06) to the low-order four bits.
2. If the value of the high-order four bits (07-04) in the accumulator is greater than 9 or if carry flag is set, the instruction adds 6(06) to the high-order four bits.

Source Program:

```
      LXI SP, 27FFH       : Initialize stack pointer
      MOV E, A            : Store the contents of accumulator
      ANI 0FH             : Mask upper nibble
      CPI 0A H            : Check if number is greater than 9
      JC SKIP             : if no go to skip
      MOV A, E            : Get the number
      ADI 06H             : Add 6 in the number
      JMP SECOND          : Go for second check
SKIP: PUSH PSW            : Store accumulator and flag contents in stack
      POP B               : Get the contents of accumulator in B register and flag
register contents in                    C register
      MOV A, C            : Get flag register contents in accumulator
      ANI 10H             : Check for bit 4
      JZ SECOND           : if zero, go for second check
      MOV A, E            : Get the number
      ADI 06              : Add 6 in the number
SECOND: MOV E, A          : Store the contents of accumulator
      ANI FOH             : Mask lower nibble
      RRC
      RRC
      RRC
      RRC                 : Rotate number 4 bit right
      CPI 0AH             : Check if number is greater than 9
      JC SKIPl            : if no go to skip 1
      MOV A, E            : Get the number
      ADI 60 H            : Add 60 H in the number
      JMP LAST            : Go to last
SKIP1: JNC LAST           : if carry flag = 0 go to last
      MOV A, E            : Get the number
      ADI 60 H            : Add 60 H in the number
LAST: HLT
```

Note: To check auxiliary carry flag it is necessary to get the flag register contents in one of the registers and then we can check the auxiliary carry flag by checking bit 4 of that register. To get the flag register contents in any general purpose register we require stack operation and therefore stack pointer is initialized at the beginning of the source program.


Start
Mask upper nibble
Is number > 9 or AC = 1 ?
No
Yes
Add 6 in the number
Get the number
Mask lower nibble
Rotate number right 4 bits
Is number > 9 or CY = 1 ?
No
Yes
Add 60H in the number
End

38 Statement:To test RAM by writing '1' and reading it back and later writing '0' (zero) and reading it back. RAM addresses to be checked are 40FFH to 40FFH. In case of any error, it is indicated by writing 01H at port 10H.

Source Program:

```
     LXI H, 4000H       : Initialize memory pointer
BACK: MVI M, FFH              : Writing '1' into RAM
     MOV A, M           : Reading data       from RAM
     CPI FFH            : Check for ERROR
     JNZ ERROR            : If yes go to ERROR
     INX H                : Increment memory pointer
     MOV A, H
     CPI SOH            : Check for last check
     JNZ BACK           : If not last, repeat
     LXI H, 4000H             : Initialize memory pointer
BACKl: MVI M, OOH             : Writing '0' into RAM
     MOV A, M           : Reading data from RAM
     CPI OOH            : Check for       ERROR
     INX H                : Increment memory pointer
     MOV A, H
     CPI SOH            : Check for last check
     JNZ BACKl            : If not last, repeat
     HLT                : Stop Execution
```

39 Statement:Write an assembly language program to generate fibonacci number.

Source Program:

```
     MVI D, COUNT       : Initialize counter
     MVI B, 00          : Initialize variable to store previous number
     MVI C, 01          : Initialize variable to store current number
     MOV A, B           :[Add two numbers]
BACK: ADD C                   :[Add two numbers]
     MOV B, C           : Current number is now previous number
     MOV C, A           : Save result as a new current number
     DCR D                : Decrement count
     JNZ BACK           : if count  0 go to BACK
     HLT                : Stop.
```

40 Statement:Write a program to generate a delay of 0.4 sec if the crystal frequency is 5 MHz.

Calculation: In 8085, the operating frequency is half of the crystal frequency, ie.Operating frequency = 5/2 = 2.5 MHz
Time for one T -state =
Number of T-states required =
= 1 x 106

Source Program:

```
LXI B, count            : 16 - bit count
BACK: DCX B             : Decrement count
MOV A, C
ORA B                 : Logically OR Band C
JNZ BACK            : If result is not zero repeat
```

41 Statement: Arrange an array of 8 bit unsigned no in descending order

Source Program:

```
START:MVI B, 00          ; Flag = 0
      LXI H, 4150        ; Count = length of array
      MOV C, M
      DCR C              ; No. of pair = count -1
      INX H              ; Point to start of array
LOOP:MOV A, M            ; Get kth element
      INX H
      CMP M              ; Compare to (K+1) th element
      JNC LOOP 1         ; No interchange if kth >= (k+1) th
      MOV D, M           ; Interchange if out of order
      MOV M, A           ;
      DCR H
      MOV M, D
      INX H
      MVI B, 01H         ; Flag=1
LOOP 1:DCR C             ; count down
      JNZ LOOP           ;
      DCR B              ; is flag = 1?
      JZ START           ; do another sort, if yes
      HLT                ; If flag = 0, step execution
```

42 Statement: Transfer ten bytes of data from one memory to another memory block. Source memory block starts from memory location 2200H where as destination memory block starts from memory location 2300H.

Source Program:

```
      LXI H, 4150      : Initialize memory pointer
      MVI B, 08        : count for 8-bit
      MVI A, 54
LOOP : RRC
      JC LOOP1
      MVI M, 00        : store zero it no carry
      JMP COMMON
LOOP2: MVI M, 01       : store one if there is a carry
COMMON:  INX H
      DCR B       : check for carry
      JNZ LOOP
      HLT             : Terminate the program
```

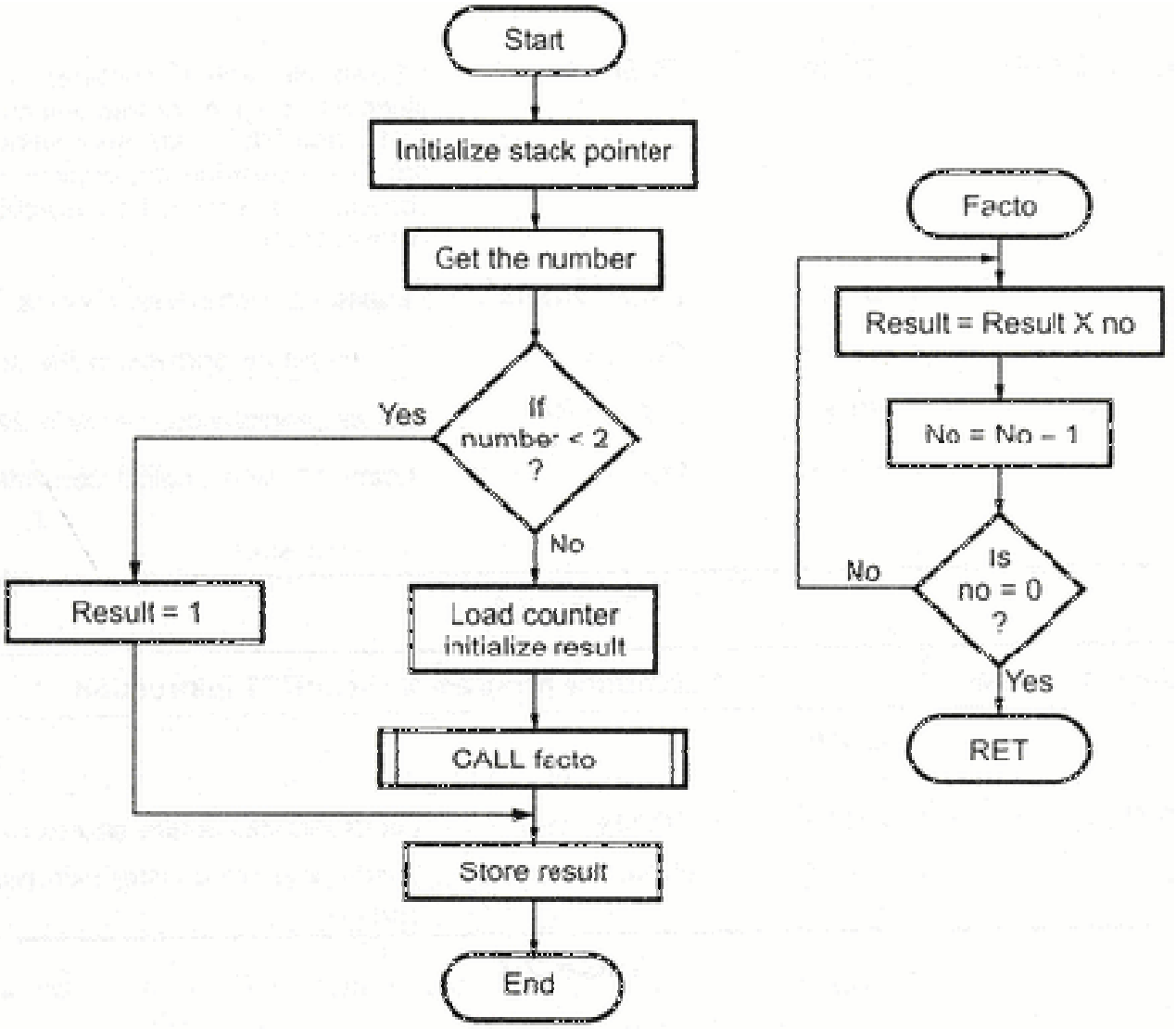
43 Statement: Program to calculate the factorial of a number between 0 to 8

Source program

```
      LXI SP, 27FFH       ; Initialize stack pointer
      LDA 2200H             ; Get the number
      CPI 02H             ; Check if number is greater than 1
      JC LAST
      MVI D, 00H            ; Load number as a result
      MOV E, A
      DCR A
      MOV C,A             ; Load counter one less than number
      CALL FACTO            ; Call subroutine FACTO
      XCHG                  ; Get the result in HL
      SHLD 2201H            ; Store result in the memory
      JMP END
LAST: LXI H, 000lH            ; Store result = 01
END: SHLD 2201H
      HLT
```

Subroutine Program:

```
FACTO:LXI H, 0000H
      MOV B, C            ; Load counter
BACK: DAD D
      DCR B
      JNZ BACK            ; Multiply by successive addition
      XCHG                ; Store result in DE
      DCR C             ; Decrement counter
      CNZ FACTO           ; Call subroutine FACTO
      RET                 ; Return to main program
```


Start
Initialize stack pointer
Get the number
If number < 2 ?
Yes
No
Result = 1
Load counter initialize result
CALL facto
Store result
End
Facto
Result = Result X no
No = No – 1
Is no = 0 ?
No
Yes
RET

44 Statement:Write a program to find the Square Root of an 8 bit binary number. The binary number is stored in memory location 4200H and store the square root in 4201H.

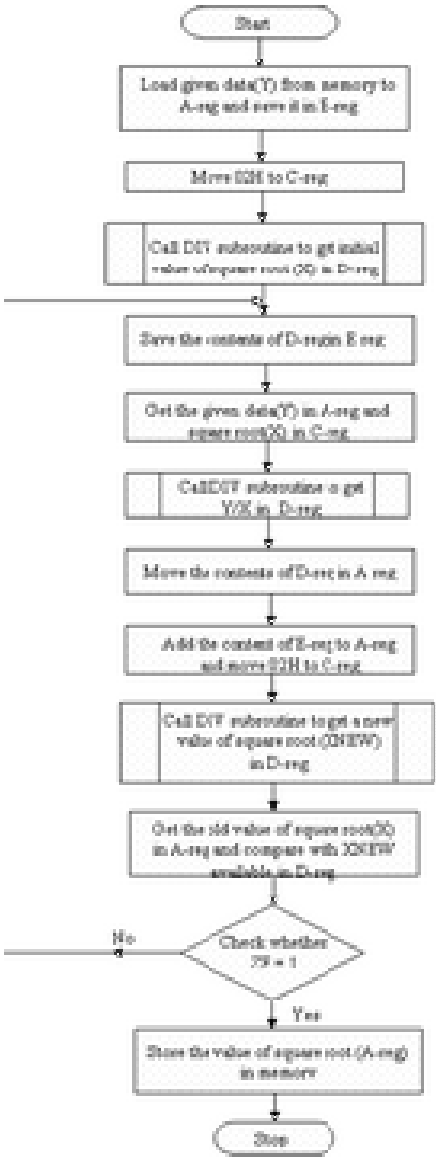
Source Program:

```
        LDA 4200H       : Get the given data(Y) in A register
        MOV B,A         : Save the data in B register
        MVI C,02H       : Call the divisor(02H) in C register
        CALL DIV        : Call division subroutine to get initial value(X) in D-
reg
REP:    MOV E,D         : Save the initial value in E-reg
        MOV A,B         : Get the dividend(Y) in A-reg
        MOV C,D         : Get the divisor(X) in C-reg
        CALL DIV        : Call division subroutine to get initial value(Y/X) in D-
reg
        MOV A, D        : Move Y/X in A-reg
        ADD E           : Get the((Y/X) + X) in A-reg
        MVI C, 02H      : Get the divisor(02H) in C-reg
        CALL DIV        : Call division subroutine to get ((Y/X) + X)/2 in D-
reg.This is XNEW
        MOV A, E        : Get Xin A-reg
        CMP D           : Compare X and XNEW
        JNZ REP         : If XNEW is not equal to X, then repeat
        STA 4201H       : Save the square root in memory
        HLT             : Terminate program execution
```

Subroutine:

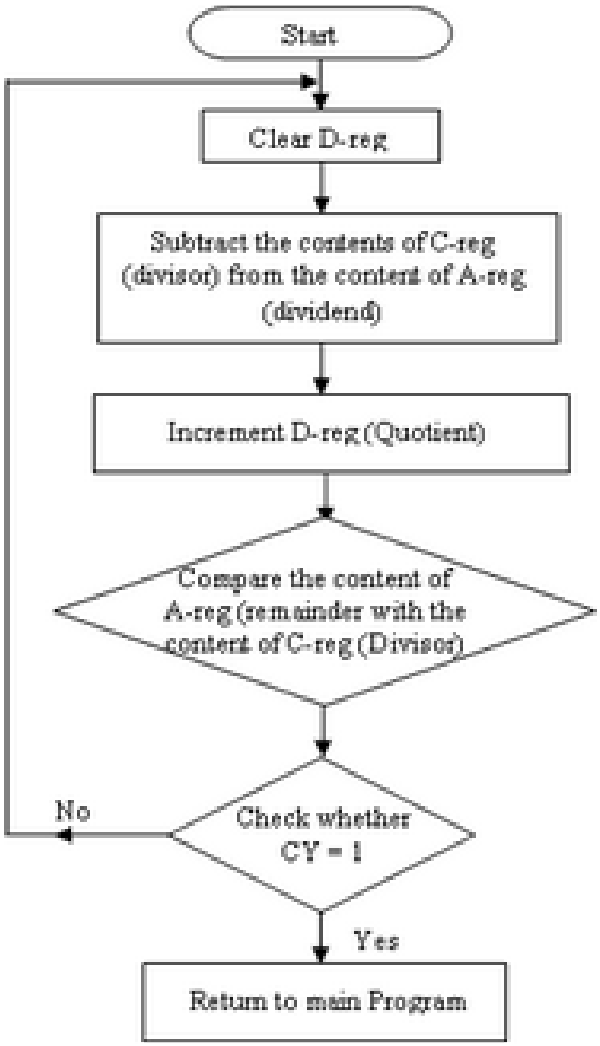
```
DIV:    MVI D, 00H      : Clear D-reg for Quotient
NEXT:   SUB C           : Subtact the divisor from dividend
        INR D           : Increment the quotient
        CMP C           : Repeat subtraction until the
        JNC NEXT        : divisor is less than dividend
        RET             : Return to main program
```

Note: The square root can be taken y an iterative technique. First, an initial value is assumed. Here, the initial value of square root is taken as half the value of given number. Te new value of square root is computed by using an expression XNEW = (X + Y/X)/2 where, X is the initial value of square root and Y is the given number. Then, XNEW is compared wit initial value. If they are not equal then the above process is repeated until X is equal to XNEW after taking XNEW as initial value. (i.e., X ←XNEW)



Flowchart Main program

Flowchart subroutine



45 Statement:Write a simple program to Split a HEX data into two nibbles and store it in memory

Source Program:

```
LXI H, 4200H        : Set pointer data for array
MOV B,M             : Get the data in B-reg
MOV A,B             : Copy the data to A-reg
ANI OFH             : Mask the upper nibble
INX H               : Increment address as 4201
MOV M,A             : Store the lower nibble in memory
MOV A,B             : Get the data in A-reg
ANI FOH             : Bring the upper nibble to lower nibble position
RRC
RRC
RRC
RRC
INX H
MOV M,A             : Store the upper nibble in memory
HLT                 : Terminate program execution
```

46 Statement: Add two 4 digit BCD numbers in HL and DE register pairs and store result in memory locations, 2300H and 2301H. Ignore carry after 16 bit.

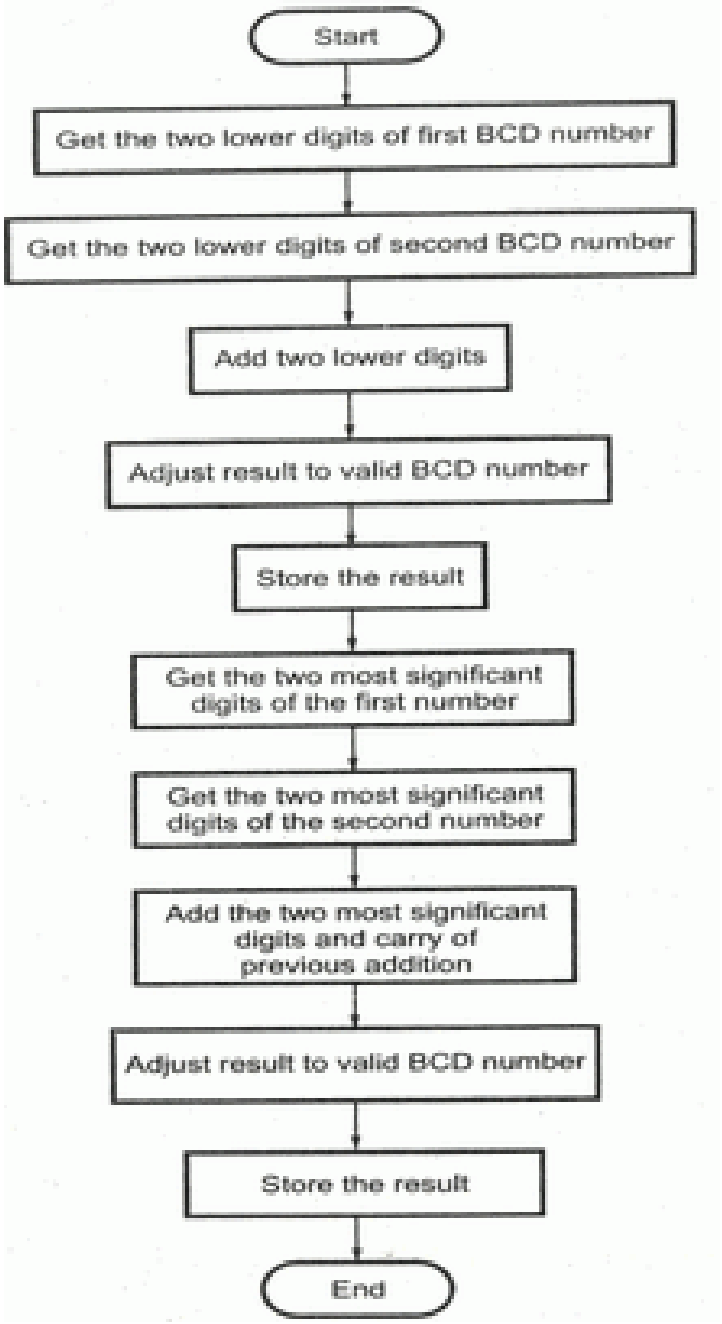
Sample Problem:

(HL) =3629
(DE) =4738
Step 1 : 29 + 38 = 61 and auxiliary carry flag = 1
:.add 06
61 + 06 = 67
Step 2 : 36 + 47 + 0 (carry of LSB) = 7D

Lower nibble of addition is greater than 9, so add 6.
7D + 06 = 83
Result = 8367

Source program

```
MOV A, L        : Get lower 2 digits of no. 1
ADD E           : Add two lower digits
DAA             : Adjust result to valid BCD
STA 2300H       : Store partial result
MOV A, H        : Get most significant 2 digits of number
ADC D           : Add two most significant digits
DAA             : Adjust result to valid BCD
STA 2301H       : Store partial result
HLT             : Terminate program execution.
```


Start
Get the two lower digits of first BCD number
Get the two lower digits of second BCD number
Add two lower digits
Adjust result to valid BCD number
Store the result
Get the two most significant digits of the first number
Get the two most significant digits of the second number
Add the two most significant digits and carry of previous addition
Adjust result to valid BCD number
Store the result
End

47 Statement: Subtract the BCD number stored in E register from the number stored in the D register.
Source Program:

```
    MVI A,99H
    SUB E          : Find the 99's complement of subtrahend
    INR A          : Find 100's complement of subtrahend
    ADD D        : Add minuend to 100's complement of subtrahend
    DAA          : Adjust for BCD
    HLT          : Terminate program execution
```

Note: When two BCD numbers are subtracted, we can use DAA instruction for ajusting the result to BCD. Therefore, the subtraction of BCD number is carried out 10's complement or 100's complement.
The 10's complement of a decimal number is equal to the 99's complement plus 1.
The 99's complement of a number can be found by subtracting the number from 99.
The steps for finding 100's complement BCD subtraction are :

- **Find the 100's complement of subtrahend**
- **Add two numbers using BCD adition**

48 Statement: Write an assembly language program to multiply 2 BCD numbers
Source Program:

```
    MVI C, Multiplier       : Load BCD multiplier
    MVI B, 00             : Initialize counter
    LXI H, 0000H            : Result = 0000
    MVI E, multiplicand       : Load multiplicand
    MVI D, 00H            : Extend to 16-bits
BACK: DAD D                 : Result Result + Multiplicand
    MOV A, L              : Get the lower byte of the result
    ADI, 00H
    DAA                   : Adjust the lower byte of result to BCD.
    MOV L, A              : Store the lower byte of result
    MOV A, H              : Get the higher byte of the result
    ACI, 00H
    DAA                   : Adjust the higher byte of the result to BCD
    MOV H, A              : Store the higher byte of result.
    MOV A, B              : [Increment
    ADI 01H               : counter
    DAA                   : adjust it to BCD and
    MOV B,A               : store it]
    CMP C                   : Compare if count = multiplier
    JNZ BACK              : if not equal repeat
    HLT                   : Stop
```

49 Statement:Write a program for displaying binary up counter. Counter should count numbers from 00 to FFH and it should increment after every 0.5 sec. Assume operating frequency of 8085 equal to 2MHz. Display routine is available.

Source Program:

```
      LXI SP, 27FFH       : Initialize stack pointer
      MVI C, OOH          : Initialize counter
BACK: CALL Display        : Call display subroutine
      CALL Delay          : Call delay subroutine
      INR C               : Increment counter
      MOV A, C
      CPI OOH             : Check counter is > FFH
      JNZ BACK            : If not, repeat
      HLT                 : Stop
```

Delay Subroutine:

```
Delay: LXI B, count       : Initialize count
      BACK: DCX  D        : Decrement count
      MOV A, E
      ORA D               : Logically OR D and E
      JNZ BACK            : If result is not 0 repeat
      RET                 : Return to main program
```

$$\text{ie. Operating frequency} = 2\ \text{MHz}$$

$$\text{Time for one T-state} = \frac{1}{2\,\text{MHz}} = 0.5\,\mu\text{sec}$$

$$\text{Number of T-states required} = \frac{\text{Required Time}}{\text{Time required for 1 T-state}} = \frac{0.5\,\text{sec}}{0.5\,\mu\text{sec}}$$

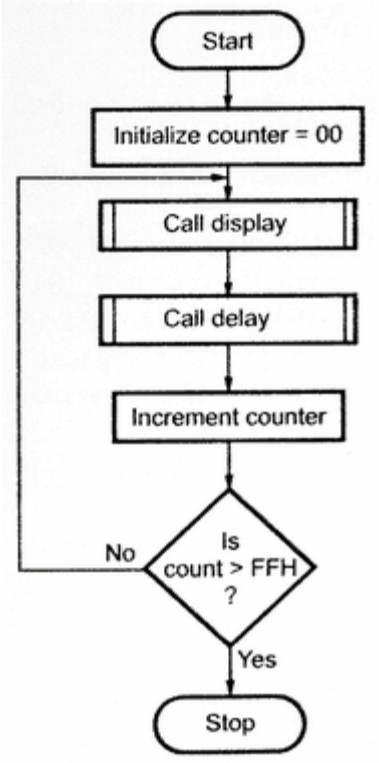
$$= 1 \times 10^6$$

$$1 \times 10^6 = 10 + (count - 1) \times 24 + 21$$

$$count = \frac{1 \times 10^6 - 31}{24} + 1 \gg 41666_{10}$$

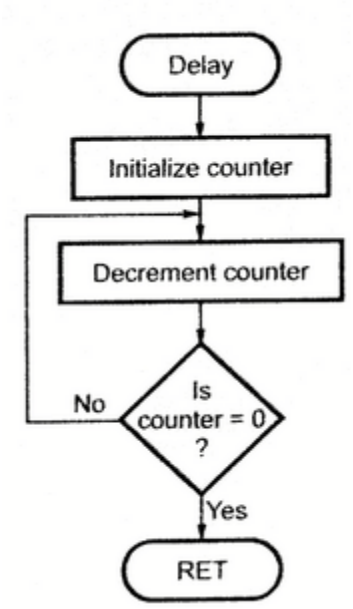
$$count = 41666_{10}$$

$$= A2C2H$$



Program flowchart

Delay routine flowchart



50 Statement:Write a program for displaying BCD up counter. Counter should count numbers from 00 to 99H and it should increment after every 1 sec. Assume operating frequency of 8085 equal to 3MHz. Display routine is available.

Source Program:

```
      LXI SP, 27FFH       : Initialize stack pointer
      MVI C, OOH          : Initialize counter
BACK: CALL Display        : Call display subroutine
      CALL Delay          : Call delay subroutine
      MOV A, C
      ADI A, 0 1          : Increment counter
      DAA                 : Adjust it for decimal
      MOV C,A             : Store count
      CPI ,00             : Check count is > 99
      JNZ BACK            : If not, repeat
      HLT                 : Stop
```

Delay Subroutine:

```
Delay:MVI B, Multiplier-count      : Initialize multiplier count
BACK 1:LXI D, Initialize Count
BACK:   DCX  D                     : Decrement count
        MOV A, E
      ORA D                 : Logically OR D and E
      JNZ BACK              : If result is not a, repeat
      DCR B                 : Decrement multiplier count
      JNZ BACK 1            : If not zero, repeat
      RET                   : Return to main program.
```

Operating Frequency : 3MHz

$$\text{Time for one T - state} = \frac{1}{3\text{MHz}} = 0.333\mu\text{sec}$$

$$\frac{\text{Required Time}}{\text{Time required for 1 T- state}} = 3\times10^6$$

$$= \frac{1}{0.333\mu\text{sec}}$$

$$\text{Let us take multiplier count} = 3$$

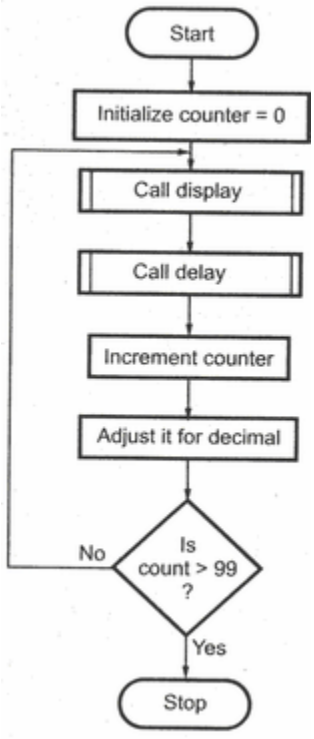
$$\text{No. of T states required by inner loop} = \frac{3\times10^6}{3} = 1\times10^6$$

$$1 \times 10^6 = 10 + (count - 1) \times 24 + 21$$

$$count = \frac{1 \times 10^6 - 31}{24} + 1 \gg 41666_{10}$$

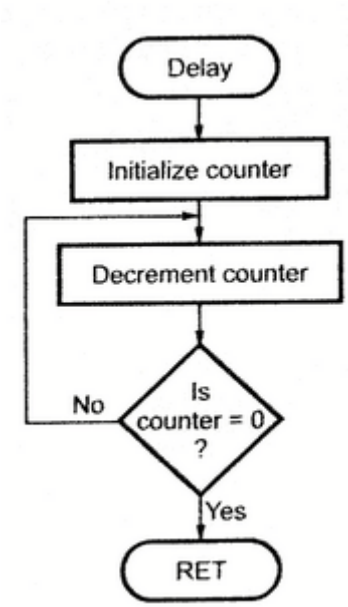
$$count = 41666_{10}$$

$$= A2C2H$$



Source program flowchart

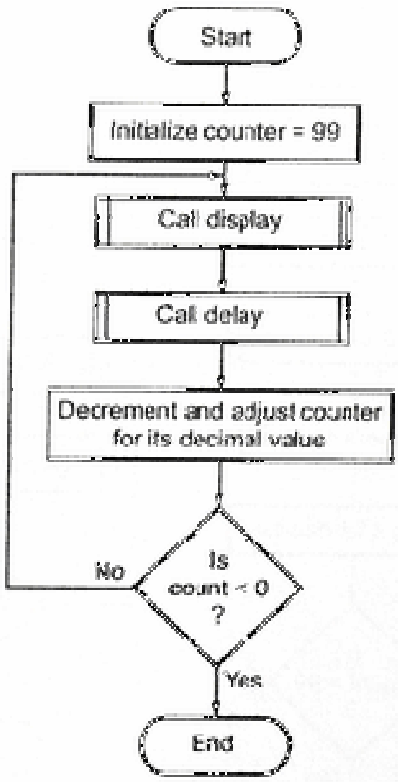
Routine flowchart



51 Statement:Write a program for displaying BCD down counter. Counter should count numbers from 99 to 00 and it should increment after every 1 sec. Assume operating frequency of 8085 equal to 3MHz. Display routine is available

Source Program 1:

```
        LXI SP, 27FFH       : Initialize stack pointer
    MVI C, 99H              : Initialize counter = 99
BACK:CALL Display           : Call display subroutine
    CALL Delay              : Call delay subroutine
    ADI 99H                 : See Addition below
    DAA                     : Adjust for decimal
    CPI 99H                 : Compare with last count
    JNZ BACK                :If no, repeat
    HLT
```



Addition:

```
          1001   1001      99H
    +     1001   1001      99H
    ------------------------
          10011  0010
DAA+      0110   0110
    ------------------------
          1001   1000      98H
```

Source Program2:

```
    LXI SP, 27FFH       : Initialize stack pointer
    MVI C, 99H          : Initialize counter = 99
BACK: CALL Display      : Call display subroutine
    CALL Delay          : Call delay subroutine
    MOV A, C            : Get count
    ANI 0FH             : Check for lower nibble
    JNZ SKIP            : If it is not 0FH go to skip
    MOV A,C             : Else get the count
    SBI ,06             : Subtract 06
    MOV C,A             : Store the count
    DCR C               : Decrement count
    CPI 99H             : Check it for last count
    JNZ BACK            : If not, repeat
    HLT                 : Stop
```

52 Statement:Write assembly language program to with proper comments for the following: To display decimal decrementing counter (99 to 00) at port 05 H with delay of half seconds between .each count. Write as well the delay routine giving delay of half seconds. Operating frequency of microprocessor is 3.072 MHz. Neglect delay for the main program.

Source Program:

```
      MVI C, 99H          : Initialize counter
BACK: MOV A, C
      ANI OF              : Mask higher nibble
      CPI OF
      JNZ SKIP
      MOV A, C
      SUI 06              : Subtract 6 to adjust decimal count
      MOV D, A
SKIP: MOV A, C
      OUT 05              : send count on output port
      CALL Delay          : Wait for 0.5 seconds
      DCR C               : decrement count
      MOV A, C
      CPI FF
      JNZ BACK            : If not zero, repeat
      HLT                 : Stop execution
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D               : 6 T-states
      MOV A, D            : 4 T-states
      ORA E               : 4 T-states
      JNZ Back            : 10 T-states
      RET
```

$$1\ \text{T-state} = \frac{1}{\text{Operating frequency}} = \frac{1}{3.072\ \text{X}\ 10^{6}}$$

$$= 3.2552 \times 10^{-7}$$

$$\text{Number of T-states required} = \frac{0.5\ \text{sec}}{3.2552 \times 10^{-7}} = 1.536\text{x}\ 10^{6}$$

$$1.536\text{x}\ 10^{6} = 10 + (count - 1) \times 24 + 21$$

$$count = \frac{1.536\text{x}\ 10^{6} - 31}{24} + 1 = 63999.708$$

$$count = 64000$$

$$= FA00H$$

53 Statement:The delay routine given below is in infinite loop, identify the error and correct the program.

Delay routine with error:

```
DELAY  : LXI H, N
L1     : DCX H
         JNZ L1
```

Sol.: 1) The fault in the above program is at instruction JNZ L1. This condition always evaluates to be true hence loops keep on executing and hence infinite loop.

2) Reason for infinite looping: - The instruction DCX H decrease the HL pair count one by one but it does not affect the zero flag. So when count reaches to OOOOH in HL pair zero flag is not affected and JNZ L1 evaluates to be true and loop continues. Now HL again decrements below OOOOH and HL becomes FFFFH and thus execution continues.

3) The modification in the program is as follows:

```
DELAY  :  LXI H, N          :Load 16 bit count
L1     :  DCX H             : Decrement count
          MOV A, L
          ORA H          : logically OR Hand L
          JNZ L1           : If result is not 0 repeat
```

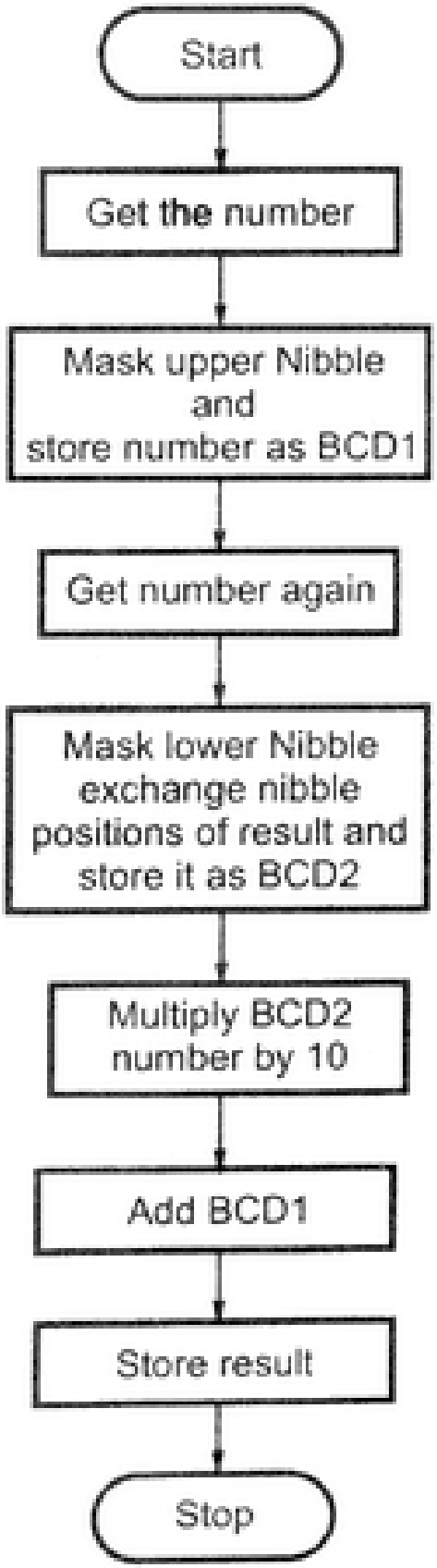
54 Statement: Convert a 2-digit BCD number stored at memory address 2200H into its binary equivalent number and store the result in a memory location 2300H.

Sample Problem

(2200H) = 67H
(2300H) = 6 x OAH + 7 = 3CH + 7 = 43H

Source Program:

```
     LDA 2200H        : Get the BCD number
     MOV B, A         : Save it
     ANI OFH          : Mask most significant four bits
     MOV C, A         : Save unpacked BCDI in C register
     MOV A, B         : Get BCD again
     ANI FOH          : Mask least significant four bits
     RRC              : Convert most significant four bits into unpacked BCD2
     RRC
     RRC
     RRC
     MOV B, A         : Save unpacked BCD2 in B register
     XRA A            : Clear accumulator (sum = 0)
     MVI D, 0AH       : Set D as a multiplier of 10
Sum: ADD D            : Add 10 until (B) = 0
     DCR B            : Decrement BCD2 by one
     JNZ SUM          : Is multiplication complete? i if not, go back and add again
     ADD C            : Add BCD1
     STA 2300H        : Store the result
     HLT              : Terminate program execution
```


Start
Get the number
Mask upper Nibble and store number as BCD1
Get number again
Mask lower Nibble exchange nibble positions of result and store it as BCD2
Multiply BCD2 number by 10
Add BCD1
Store result
Stop

55 Statement: Write a main program and a conversion subroutine to convert the binary number stored at 6000H into its equivalent BCD number. Store the result from memory location 6100H.

Sample Problem: (6000) H = 8AH

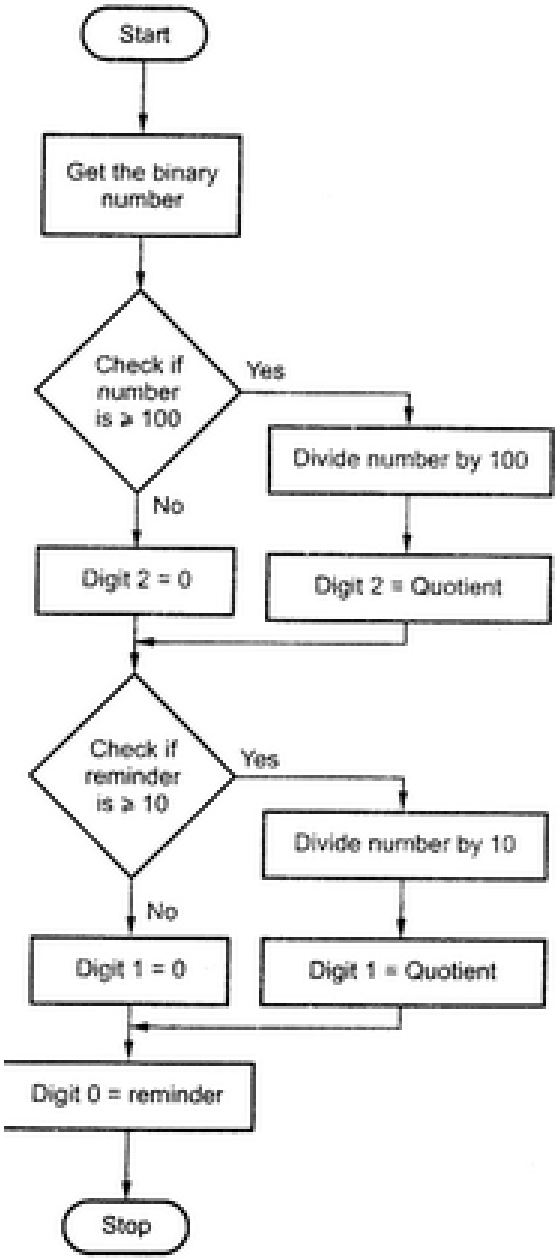
1.8AH ? 64H (Decimal 100) :. Divide by 64H (Decimal 100)
8AH/64H ? Quotient = 1, Remainder = 26H
26H < 64H (Decimal 100) :. Go to step 2 and Digit 2 = 1
2.26H ? OAH (Decimal 10) :. Divide by OAH (Decimal 10)
26H/OAH ? Quotient = 3, Remainder = O8H
OSH < OAH (Decimal 10) :. Go to step 3 and Digit 1 = 3
3. Digit 0 = O8H

Source Program:

```
LXI SP, 27FFH          : Initialize stack pointer
LDA 6000H              : Get the binary number in accumulator
CALL SUBROUTINE        : Call subroutine
HLT                    : Terminate program execution
```

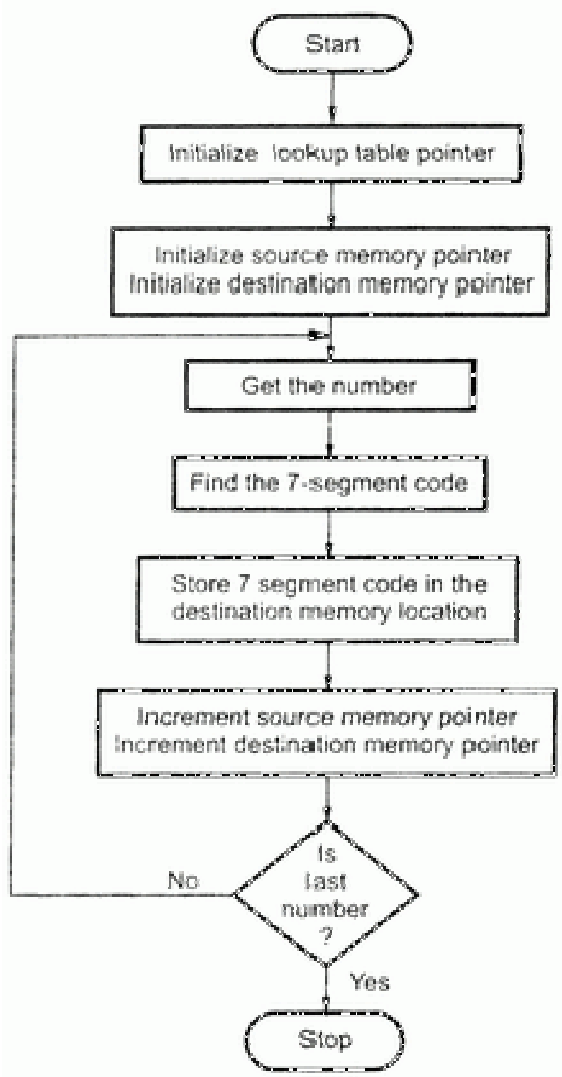
Subroutine to convert binary number into its equivalent BCD number:

```
PUSH B          : Save BC register pair contents
PUSH D          : Save DE register pair contents
MVI B, 64H      : Load divisor decimal 100 in B register
MVI C, 0AH      : Load divisor decimal 10 in C register
MVI D, 00H      : Initialize Digit 1
MVI E, 00H      : Initialize Digit 2
STEP1: CMP B    : Check if number < Decimal 100
JC STEP 2       : if yes go to step 2
SUB B           : Subtract decimal 100
INR E           : update quotient
JMP STEP 1      : go to step 1
STEP2: CMP C    : Check if number < Decimal 10
JC STEP 3       : if yes go to step 3
SUB C           : Subtract decimal 10
INR D           : Update quotient
JMP STEP 2      : Continue division by 10
STEP3: STA 6100H    : Store Digit 0
MOV A, D        : Get Digit 1
STA 6101H       : Store Digit 1
MOV A, E        : Get Digit 2
STA 6102H       : Store Digit 2
POP D           : Restore DE register pair
POP B           : Restore BC register pair
RET             : Return to main program
```


Start
Get the binary number
Check if number is ⩾ 100
Yes
Divide number by 100
No
Digit 2 = 0
Digit 2 = Quotient
Check if reminder is ⩾ 10
Yes
Divide number by 10
No
Digit 1 = 0
Digit 1 = Quotient
Digit 0 = reminder
Stop

56 Statement: Find the 7-segment codes for given 5 numbers from memory location 6000H and store the result from memory location 7000H.
Sample Problem: (6000) H = 8AH
Source Program

```
LXI H, 6200H      : Initialize lookup table pointer
LXI D, 6000H      : Initialize source memory pointer
LXI B, 7000H      : Initialize destination memory pointer
BACK: LDAX D      : Get the number
MOV L, A          : A point to the 7-segment code
MOV A, M          : Get the 7-segment code
STAX B            : Store the result at destination memory location
INX D             : Increment source memory pointer
INX B             : Increment destination memory pointer
MOV A, C
CPI O5H           : Check for last number
JNZ BACK          : If not repeat
HLT               : End of program
```



Lookup Table

Digit	Code
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F

57 Statement: Write an assembly language program to convert the contents of the five memory locations starting from 2000H into an ASCII character. Place the result in another five memory locations starting from 2200H.

Sample Problem

(2000H) = 1
(2001H) = 2
(2002H) = 9
(2003H) = A
(2004H) = B
Result:(2200H) = 31
(2201H) = 32
(2202H) = 39
(2203H) = 41
(2204H) = 42

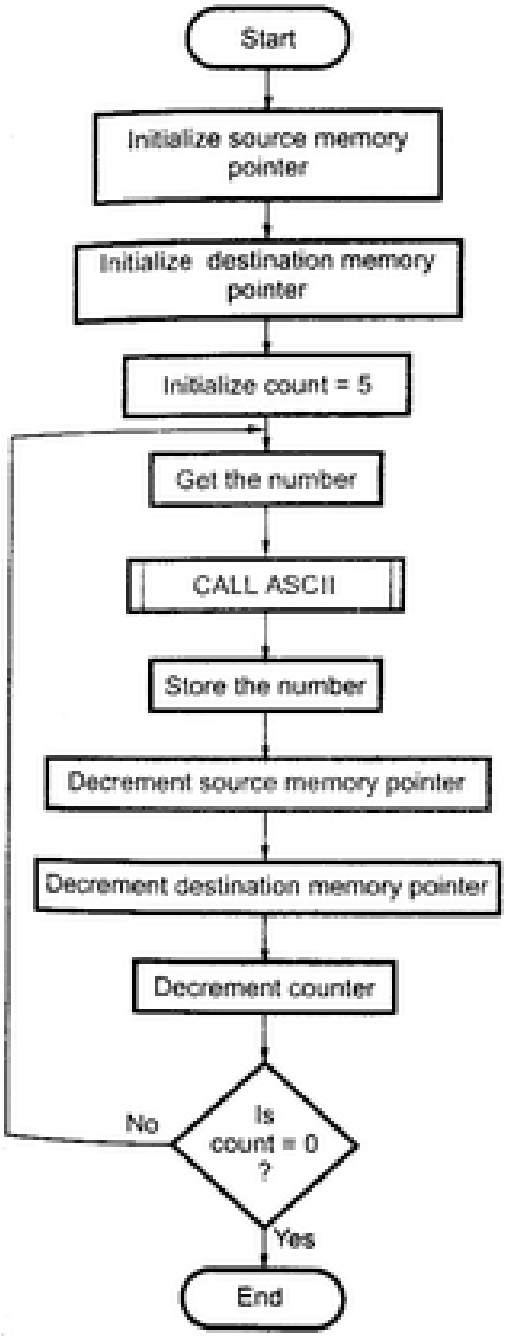
Source program:

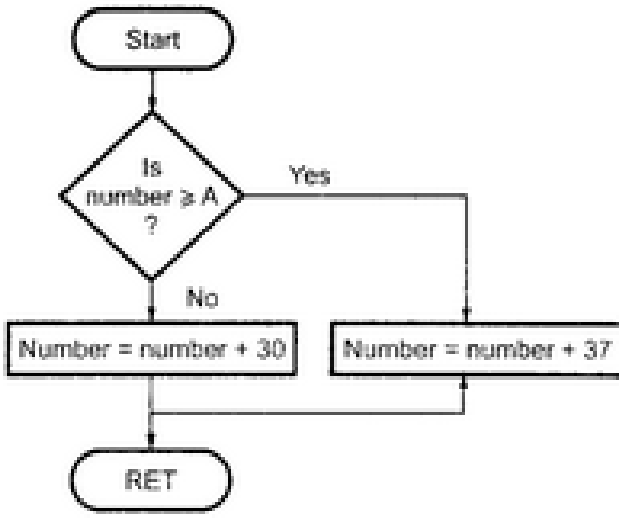
```
    LXI SP, 27FFH      : Initialize stack pointer
    LXI H, 2000H       : Source memory pointer
    LXI D, 2200H       : Destination memory pointer
    MVI C, O5H         : Initialize the counter
BACK: MOV A, M         : Get the number
    CALL ASCII         : Call subroutine ASCII
    STAX D             : Store result
    INX H              : Increment source memory pointer
    INX D              : Increment destination memory pointer
    DCR C              : Decrement count by 1
    CJNZ               : if not zero, repeat
    HLT                : Stop program execution subroutine ASCII
ASCII: CPI, OAH        : Check if number is OAR
    JNC NEXT           : If yes go to next otherwise continue
    ADI 30H
    JMP LAST
NEXT: ADI 37H
LAST: RET              : Return to main program
```

Subroutine:

Subroutine 'ASCII' converts a hexadecimal digit to ASCII.The digit is passed using accumulator and the result is stored in accumulator.Stack starts From 27FEH to 27FDH.

Note: The ASCII Code (American Standard Code for Information Interchange) is commonly used for communication. In such cases we need to convert binary number to its ASCII equivalent. It is a seven bit code. In this code number 0 through 9 are represented as 30 through 39 respectively and letters A through Z are represented as 41H through 5AH. Therefore, by adding 30H we can convert number into its ASCII equivalent and by adding 37H we can convert letter to its ASCII equivalent.


Start
Initialize source memory pointer
Initialize destination memory pointer
Initialize count = 5
Get the number
CALL ASCII
Store the number
Decrement source memory pointer
Decrement destination memory pointer
Decrement counter
Is count = 0 ?
No
Yes
End


Start
Is number ≥ A ?
Yes
No
Number = number + 30
Number = number + 37
RET

58 Statement: convert the ASCII number in memory to its equivalent decimal number

Source Program:

```
     LXI      H, 4150      : Point to data
     MOV      A, M           : Get operand
     SUI      30           : convert to decimal
     CPI      0A           : Check whether it is valid decimal number
     JC       LOOP           : yes, store result
     MVI      A, FF          : No, make result=FF
LOOP: INX     H
     MOV      M, A
     HLT                   : (A) = (4151)
```

Note: The ASCII Code (American Standard Code for Information Interchange) is commonly used for communication. It is a seven bit code. In this code number 0 through 9 are represented as 30 through 39 respectively and letters A through Z are represented as 41H through 5AH. Therefore, by subtracting 30H we can convert an ASCII number into its decimal equivalent.

59 Statement: Convert the HEX number in memory to its equivalent decimal number

Source Program:

```
      LXI H, 4150            ; Point to data
      LXI B, 0000            ; Initialize hundreds= 0, Tens=0
      MOV A, M             ; Get hex data to A
LOOP: SUI       64
       JC LOOP 1
       INR B                 ; hundreds= hundreds+1
       JMP LOOP
LOOP 1: ADI 64               ; if subtracted extra, add it clear carry flag
LOOP 2: SUI       0A
       JC LOOP 3
       INR C             ; Tens=tens+1
       JMP LOOP 2
LOOP 3: ADI 0A               ; If subtracted extra, add it again
      INX H                  ; A = Units
      MOV M, B             ; store hundreds
      MOV B, A             ; Combine Tens in C &
      MOV A, C             ; Units in A to form a
      RLC                  ; Single 8-bit number
      RLC
      RLC
      RLC
      ADD       B
      INX H
      MOV M, A             ; Store tens & Units
      HLT
```

Note: In this experiment the number is converted to its equivalent decimal number using the following logic.
First count the number of hundreds, the number of tens & units present in that hex number. Then add up to get the equivalent decimal number.
Converting A9 we get:
A9 /64=45 Hundreds = 01
Since 64(100 decimal) cannot be subtracted from 45 no. of hundreds = 01. Now count tens
45/0A=3B Tens = 01
Now from 09, 0A cannot be subtracted. Hence tens = 06 the decimal equivalent of A9 is 169.

60 Statement: Convert an 8 bit hex no to its binary form & store in memory.
Source Program:

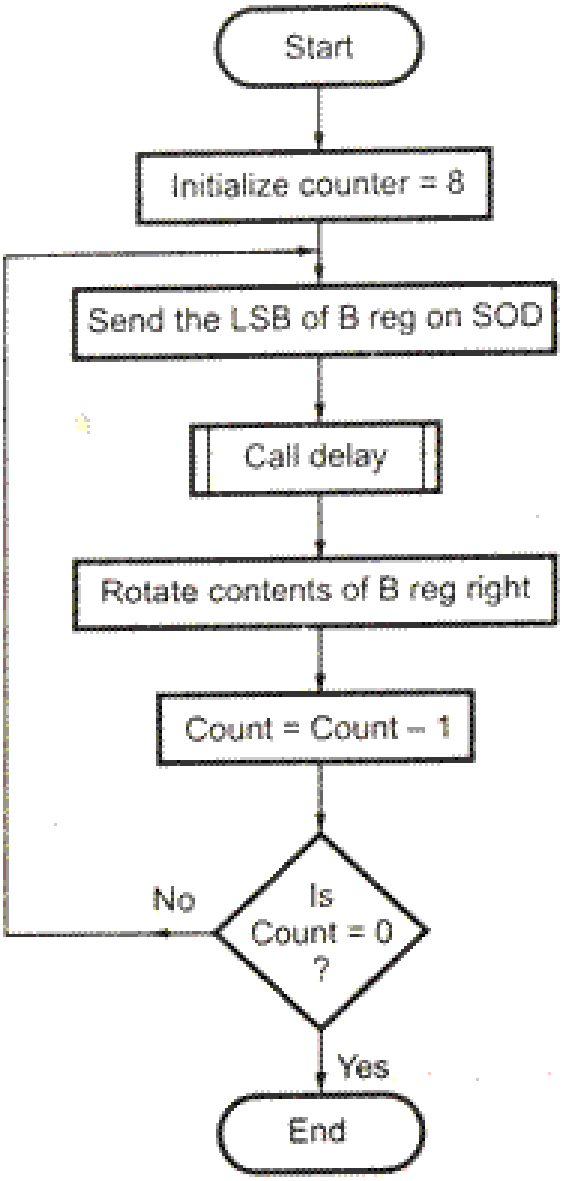
```
        LXI H, 4150       : Initialize memory pointer
        MVI B, 08             : count for 8-bit
        MVI A, 54
    LOOP : RRC
        JC LOOP1
        MVI M, 00             : store zero it no carry
        JMP COMMON
    LOOP2: MVI M, 01      : store one if there is a carry
COMMON:  INX H
        DCR B       : check for carry
        JNZ LOOP
        HLT             : Terminate the program
```

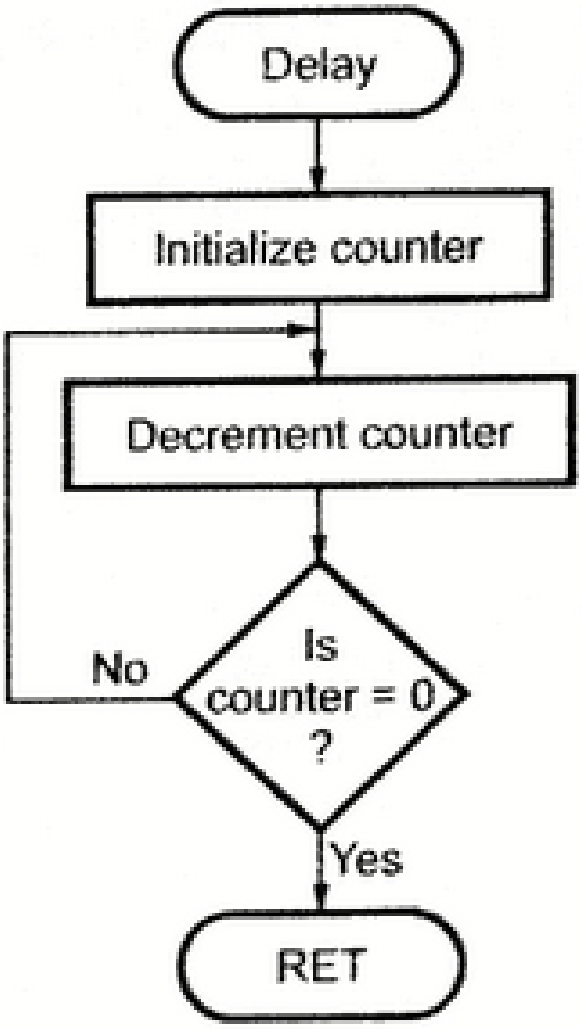
61 Statement: Write a program to output contents of B register LSB to MSB on the SOD pin.
Source program:

```
    MVI C, 08H            : Initialize count with 8
    MOV A, B
BACK: RRC                 : Rotate B register contents right
    MOV B, A            : Save contents of register B
    JNC SKIP            : If no carry skip
    MVI A, COH
    SIM                 : If carry, send high on SOD
    JMP NEXT
SKIP: MVI A, 40H
    SIM                 : If no carry, send low on SOD.
NEXT: CALL DELAY            : Wait for specific time
    DCR C           : Decrement count by 1
    JNZ BACK            : if count = 0 Stop, if not repeat
    HLT                 : Stop program execution
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
    MOV A, D
    ORA E
    JNZ Back
    RET
```


Start
Initialize counter = 8
Send the LSB of B reg on SOD
Call delay
Rotate contents of B reg right
Count = Count – 1
Is Count = 0 ?
No
Yes
End


Delay
Initialize counter
Decrement counter
Is counter = 0 ?
No
Yes
RET

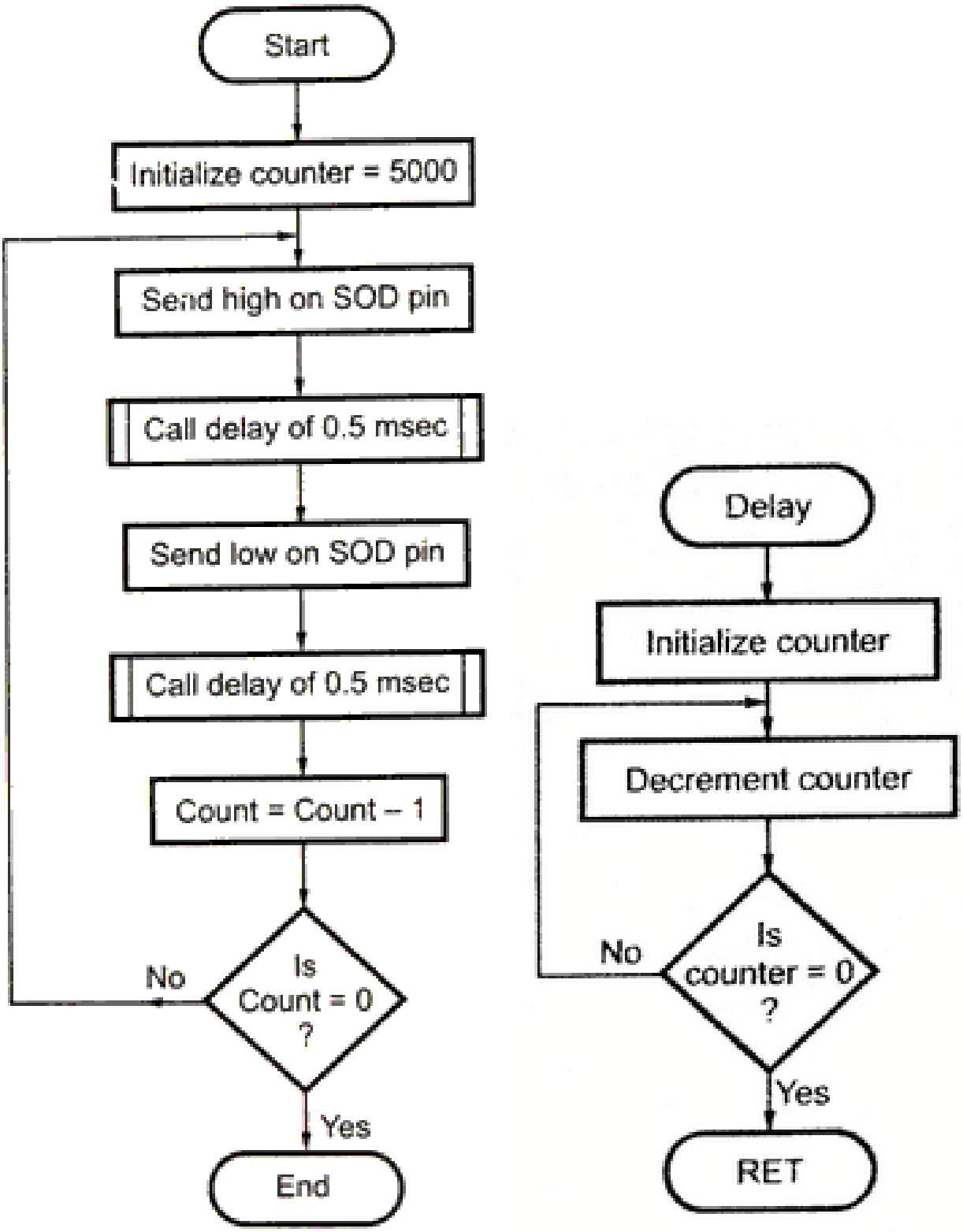
62 Statement: Write a program to output square wave of 1 kHz frequency on the SOD pinof 8085 for 5 seconds. Operating frequency of 8085 is 2 MHz.

Source program

```
        LXI SP, 27FFH          : Initialize stack pointer
        LXI B, 1388H           : Initialize counter with count 5000.
BACK:   MVI A, COH
        SIM                    : Send high on SOD pin
        CALL DELAY             : Wait for 0.5 msec
        MVI A, 40H             : Send low on SOD pin
        CALL DELAY             : wait for. 5 msec
        DCX B                  : Decrement count by 1
        MOV A, C
        ORA B                  : Check if count = 0
        JNZ BACK               : If not, repeat
        HLT                    : Stop program execution
```

Delay subroutine:

```
Delay:  LXI D, Count
Back:   DCX D
        MOV A, D
        ORA E
        JNZ Back
        RET
```


Start
Initialize counter = 5000
Send high on SOD pin
Call delay of 0.5 msec
Send low on SOD pin
Call delay of 0.5 msec
Count = Count – 1
Is Count = 0 ?
No
Yes
End
Delay
Initialize counter
Decrement counter
Is counter = 0 ?
No
Yes
RET

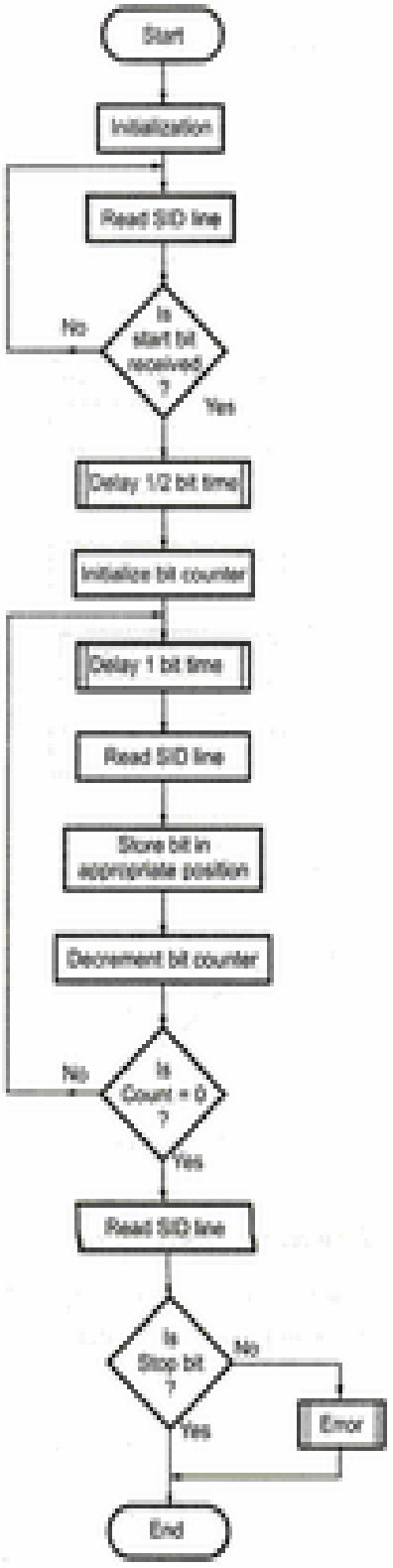
63 Statement: An ASCII character is being received on SID pin of 8085. Write a program in assembly language of 8085 to assemble this character and store it in memory. Write comment for each instruction.

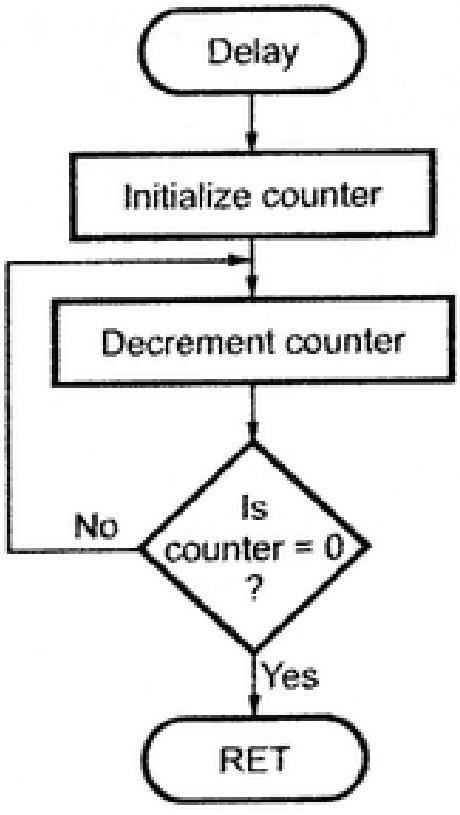
Source program:

```
    LXI SP, 27FFH
    LXI H, 2000H          : Memory pointer
    RIM                   : Read SID
    ANI 80H               : Check D7 bit of Accumulator
    CALL Delay            : 1/2 bit time delay for stop bit
    MVI B, 08H            : Initialize bit counter
    MVI D, 00H            : Clear data register
UP1: ALL Delay            : 1 bit time
    RIM                   : Read SID line
    ANI 80H               : Mask bits B6 - Bo
    ORA D                 : OR data bit with previous bits
    RRC
    MOV D, A              : Store data bit at appropriate position
    DCR B
    JNZ UP1
    RLC                   : Shift left to correct result
    MOV M, A              : Store result
    RIM                   : Read stop bit
    ANI 8OH
    CZ error              : If not stop bit call error
    HLT                   : Terminate program.
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
    MOV A, D
    ORA E
    JNZ Back
    RET
```


Start
Initialization
Read SID line
Is start bit received ?
No
Yes
Delay 1/2 bit time
Initialize bit counter
Delay 1 bit time
Read SID line
Store bit in appropriate position
Decrement bit counter
Is Count = 0 ?
No
Yes
Read SID line
Is Stop bit ?
No
Yes
Error
End


Delay
Initialize counter
Decrement counter
Is counter = 0 ?
No
Yes
RET

64 Statement: Write a assembly program to transmit a message from an 8085 to a CRT terminal for the following requirements and draw the interfacing diagram.

i) A message of 50 characters is stored as ASCII characters (without parity) in memory locations starting at 2200H.

ii) Baud rate x 16

iii) Stop bits 2

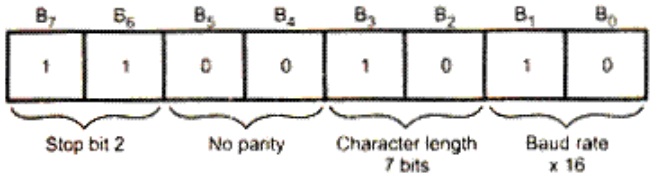
Solution Description:

- **CRT terminal uses normal RS 232C standard serial communication interface. Therefore, to transmit data to CRT it is necessary to have RS 232C interface at the sending end.**
- **Fig. shows the interfacing of 8251 with RS 232C to 8085.**
- **As shown in the Fig. three RS-232C signals (TxD, RxD are Ground) are used for serial communication between the CRT terminal and the 8085 system.**
- **Line drivers and receivers are used to transfer logic levels from TTL logic to RS-232C logic.**
- **For RS-232C the voltage level +3V to +15V is defined as logic 0 and voltage level from -3V to -15V is defined as logic 1.**
- **The line driver, MC 1488, converts logic 1 of TIL to approximately -9V and logic a of TIL to approximately +9V. These levels at the receiving end are again converted by the line receiver, MC1489, into TTL compatible logic.**

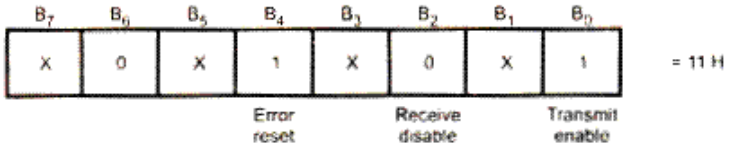
I/O Map :

Register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data Register	1	1	1	1	1	1	1	0	FEH
Control Register	1	1	1	1	1	1	1	1	FFH

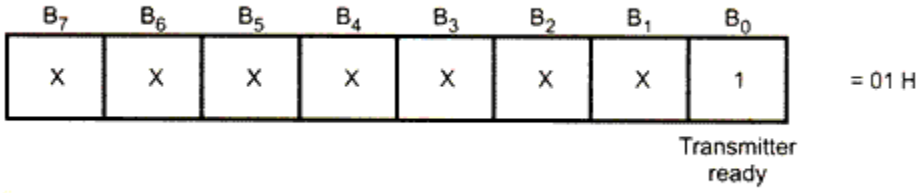
Mode word necessary for the given specification is as follows :



Command word necessary for the given specification is as follows



Status word necessary for the given specification is as follows



If bit 0 of the Status word is logic '1' then transmitter is ready to accept the character

Source program:

```
        LXI H, 2200H      : Initialize memory pointer to pointer the message
        MVI C, 32H        : Initialize counter to send 50 characters
        MVI A, 00H
        OUT FFH
        OUT FFH           : Dummy mode word
        OUT FFH
        MVI A, 40H        : Reset command word
        OUT FFH           : Reset 8251A
        MVI A, CAH        : Mode word initialization
        OUT FFH
        MVI A, 11H        : Command word initialization
        OUT FFH
CHECK:  IN FFH
        ANI 0lH           : Check TxRDY
        JZ CHECK          : Is TxRDY I? if not, check again
        MOV A, M          : Get the character in accumulator
        OUT FEH           : Send character to the transmitter
        INX H             : Increment memory pointer
        DCR C             : Decrement counter
        JNZ CHECK         : if not zero, send next character
        HLT               : Stop program execution
```

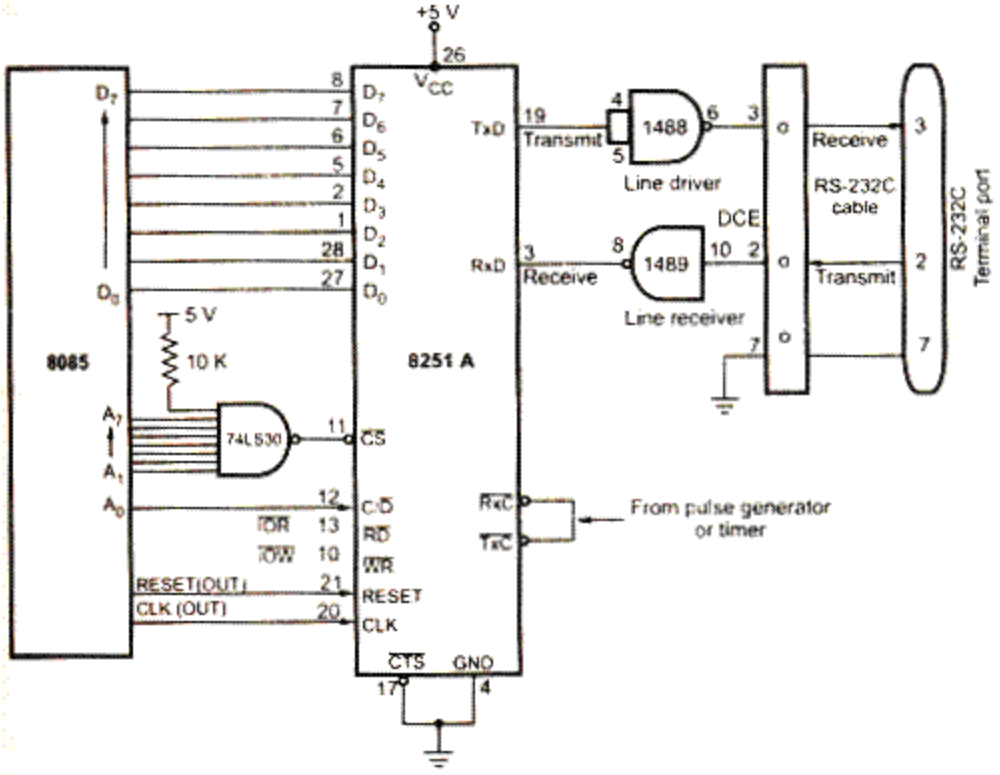
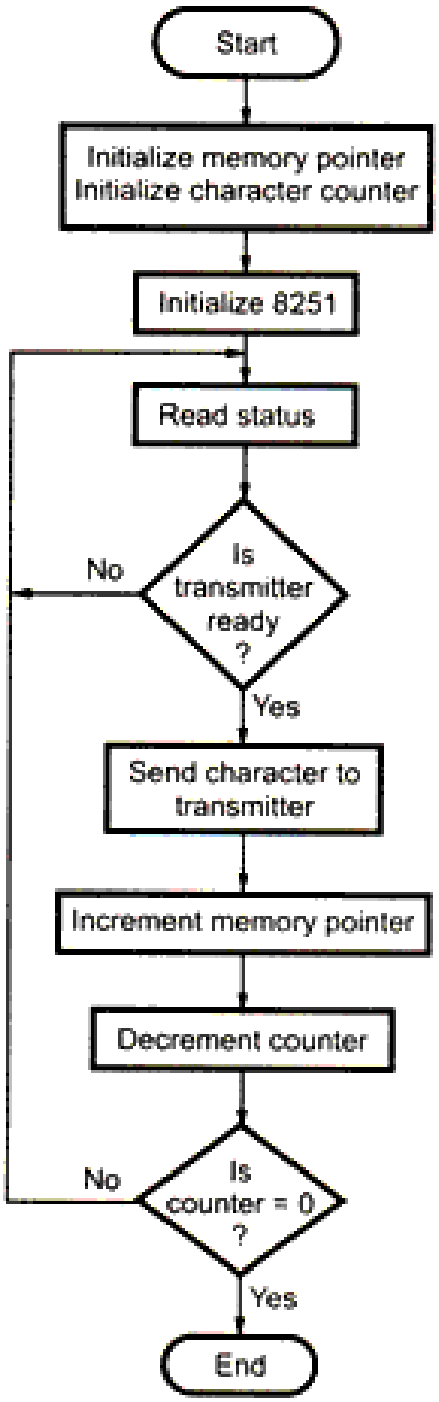
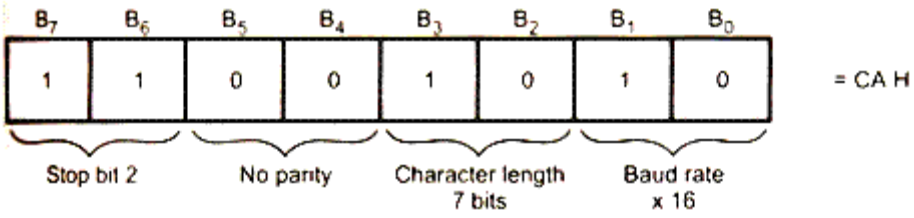


Fig. Schematic of interfacing an RS-232C terminal with an 8085 system using the 8251A

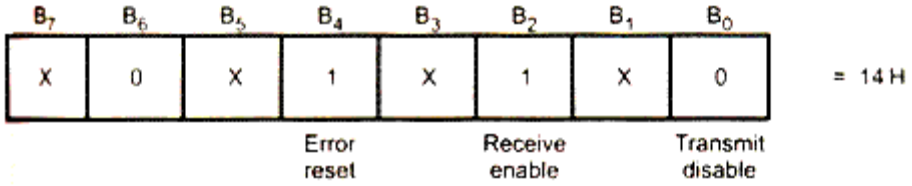

Start
Initialize memory pointer
Initialize character counter
Initialize 8251
Read status
Is transmitter ready ?
No
Yes
Send character to transmitter
Increment memory pointer
Decrement counter
Is counter = 0 ?
No
Yes
End

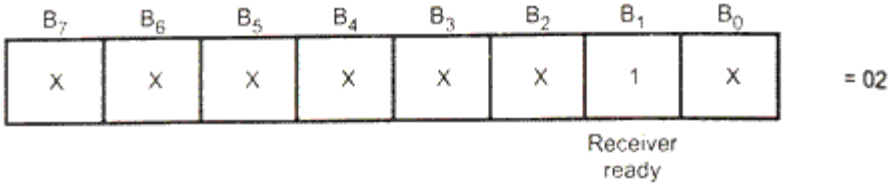
65 Statement: Write a assembly program to receive 25 bytes from an CRT terminal to 8085 for the following requirements.

i) Baud rate x 16
ii) Stop bits 2



Command word necessary for the given specification is as follows





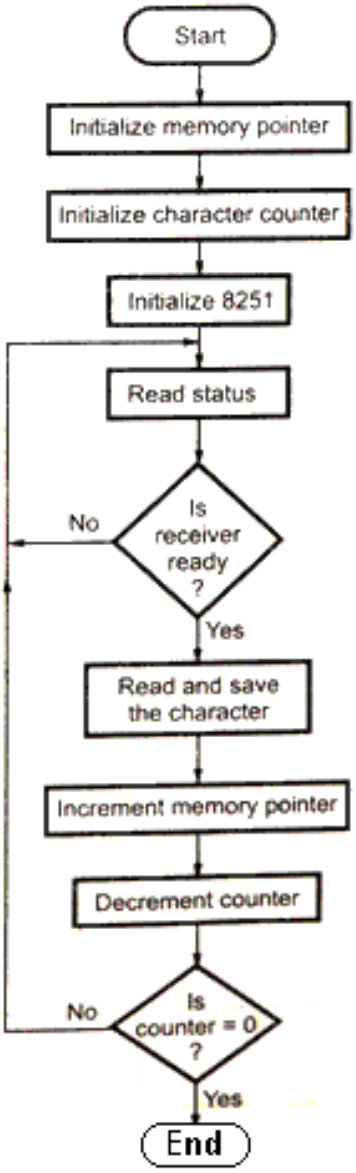
If bit 1 of the status word is logic '1' then receiver is ready to give the character

Note: Reading of status word is necessary for checking the status of RxD line of 8085 that whether receiver is ready to give data or not.

Source program:

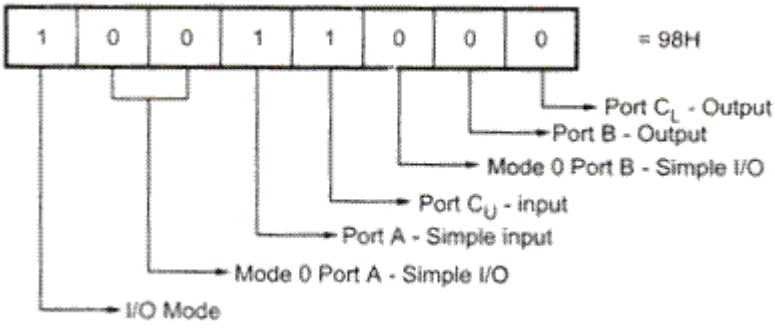
```
        LXI H, 2300 H        : Initialize memory pointer
        MVI C, FFH           : Initialize counter to accept 25 characters
        MVI A, 00H
        OUT FFH
        OUT FFH              : Dummy mode word
        OUT FFH
        MVI A, 40H           : Reset command word
        OUT FFH              : Reset 8251 A
        MVI A, CAH           : Mode word initialization
        OUT FFH
        MVI A, 14 H          : Command word initialization
        OUT FFH
CHECK:  IN FFH
        ANI 02 H             : Check RxRDY
        JZ CHECK             : Is RxRDY ? If not, check again
```

```
IN FEH              : Get the character
MOV M, A              : save the character
INX H                 : Increment memory pointer
DCR C                 : Decrement memory pointer
OUT FEH             : Send character to the transmitter
JNZ CHECK             : If not zero, accept next character
HLT                 : Stop program execution
```



66 Statement:
Write a program to initialize 8255 in the configuration given below
Sample 1:
Write a program to initialize 8255 in the configuration given below:
1. Port A: Simple input
2. Port B: Simple output
3. Port CL: Output
4. Port Cu: Input
Assume address of the control word register of 8255 as 83H.

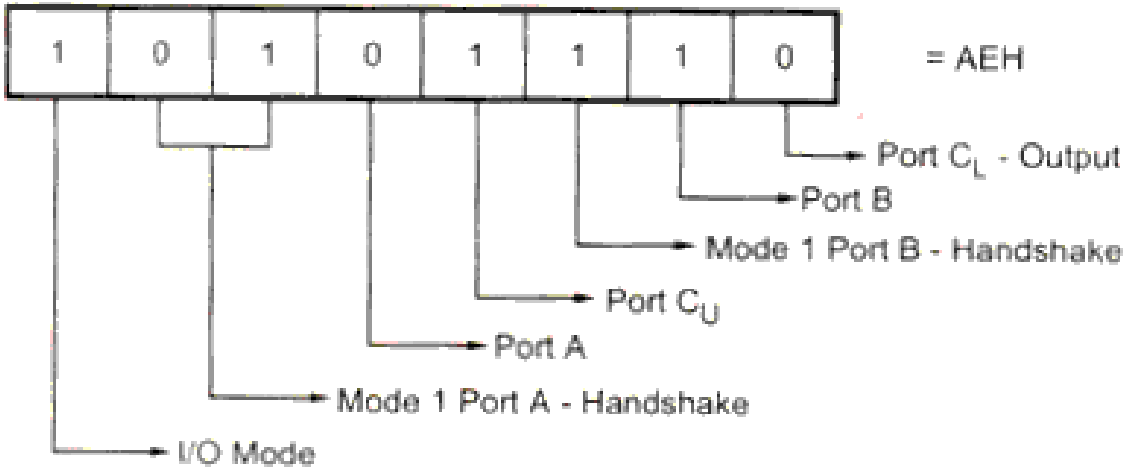
Solution:



SOURCE PROGRAM 1:

```
MVI A, 98H        : Load control word
OUT 83H           : Send control word
```

Sample 2:
Write a program to initialize 8255 in the configuration given below:
1. Port A: Output with handshake
2. Port B: Input with handshake
3. Port CL: Output
4. Port Cu: Input
Assume address of the control word register of 8255 as 23H.
Solution:

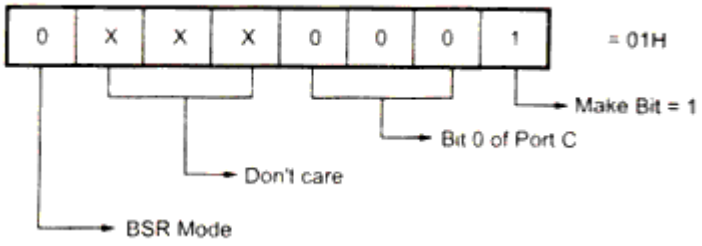


SOURCE PROGRAM 2:

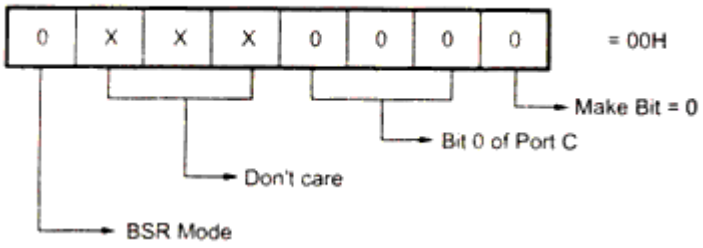
```
MVI A, AEH        : Load control word
OUT 23H           : Send control word
```

67 Statement: Write a program to blink Port C bit 0 of the 8255. Assume address of control word register of 8255 as 83H. Use Bit Set/Reset mode.

Control word to make Bit 0 high



Control word to make Bit 0 low

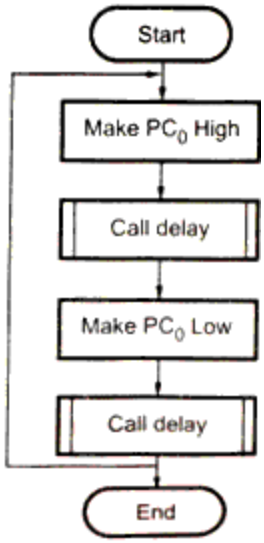


Source program:

```
BACK: MVI A, 0lH        : Load bit pattern to make PCo high
      OUT 83H           : Send it to control word register
      CALL DELAY        : Call Delay subroutine
      MVI A, 00H        : Load bit pattern to make PCo Low
      OUT 83H           : Send it to control word register
      CALL Delay        : Call Delay subroutine
      JMP BACK          : Repeat
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
      MOV A, D
      ORA E
      JNZ Back
      RET
```



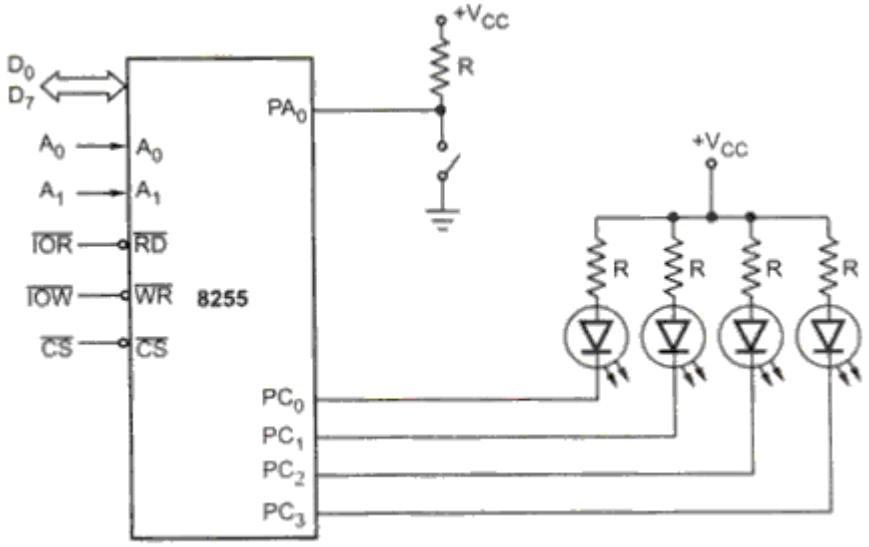
68 Statement: Design a system (both Software and Hardware) that will cause 4 LEDs to flash 10 times when a push button switch is pressed. Use 8255. Assume persistence of vision to be 0.1 seconds.

Source program:

```
      LXI SP, 2000 H          : Initialize stack pointer
      MVI A, 90H
      OUT CR                  : Initialize 8255
BACK: IN PA                   : [Read status
      ANI 01                  : of push
      JNZ BACK                : button]
      MVI B, 0AH              : Initialize counter
AGAIN: MVI A, 00H             : Load data to light LEDs
      OUT PC                  : Send data on port C
      CALL Delay              : Call. Delay of 0.1 sec
      MVI A, FFH              : Load data to switch off LEDs
      OUT PC                  : Send data on port C
      CALL Delay              : Call Delay of 0.1 sec
      DCR B                   : Decrement count
      JNZ AGAIN               : If not zero repeat
      JMP BACK                : Jump back to read status
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
      MOV A, D
      ORA E
      JNZ Back
      RET
```



69 Statement: Design a microprocessor system to control traffic lights. The traffic light arrangement is as shown in Fig. The traffic should be controlled in the following manner.
1) Allow traffic from W to E and E to W transition for 20 seconds. 2) Give transition period of 5 seconds (Yellow bulbs ON) 3) Allow traffic from N to 5 and 5 to N for 20 seconds 4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.
HARDWARE FOR TRAFFIC LIGHT CONTROL

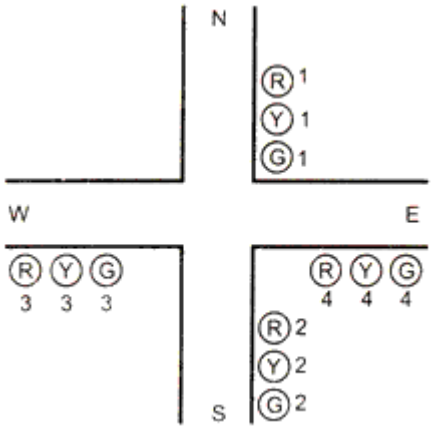


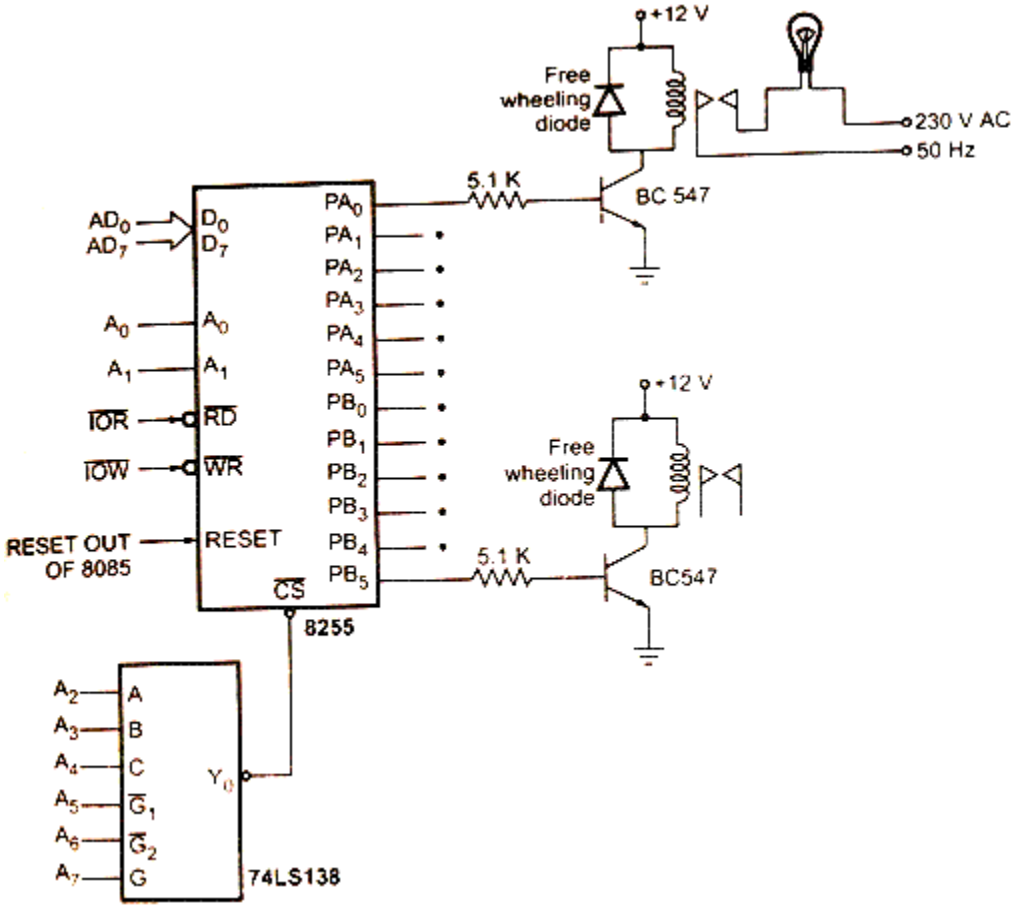
Fig. shows the interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

Pins	Light	Pins	Light
PA_0	R_1	PB_0	R_3
PA_1	Y_1	PB_1	Y_3
PA_2	G_1	PB_2	G_3
PA_3	R_2	PB_3	R_4
PA_4	Y_2	PB_4	Y_4
PA_5	G_2	PB_5	G_4

Table 1

The electric bulbs are controlled by relays. The 8255 pins are used to control relay on-off action with the help of relay driver circuits. The driver circuit includes 12 transistors to drive 12 relays. Fig. also shows the interfacing of 8255 to the system.

INTERFACING DIAGRAM



I/O MAP:

Ports / Control Register	Address lines A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0	Address
Port A	1 0 0 0 0 0 0 0	80H
Port B	1 0 0 0 0 0 0 1	81H
Port C	1 0 0 0 0 0 1 0	82H
Control Register	1 0 0 0 0 0 1 1	83H

Table 2

SOFTWARE FOR TRAFFIC LIGHT CONTROL

Control word : For initialization of 8255.

BSR/IO	$MODE_A$		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

Table shows the data bytes to be sent for specific combinations.

To glow	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Port B Output	Port A Output
R_1,R_2,G_3 and G_4	X	X	1	0	0	1	0	0	X	X	0	0	1	0	0	1	24H	09H
Y_1,Y_2,Y_3 and Y_4	X	X	0	1	0	0	1	0	X	X	0	1	0	0	1	0	12H	12H
R_3,R_4,G_1 and G_2	X	X	0	0	1	0	0	1	X	X	1	0	0	1	0	0	09H	24H

Source program:

```
      MVI A, 80H          : Initialize 8255, port A and port B
      OUT 83H (CR)        : in output mode
START: MVI A, 09H
      OUT 80H (PA)        : Send data on PA to glow R1 and R2
      MVI A, 24H
      OUT 81H (PB)        : Send data on PB to glow G3 and G4
      MVI C, 28H          : Load multiplier count (40ıo) for delay
      CALL DELAY          : Call delay subroutine
      MVI A, 12H
      OUT (81H) PA        : Send data on Port A to glow Y1 and Y2
      OUT (81H) PB        : Send data on port B to glow Y3 and Y4
      MVI C, 0AH          : Load multiplier count (10ıo) for delay
CALL: DELAY               : Call delay subroutine
      MVI A, 24H
      OUT (80H) PA        : Send data on port A to glow G1 and G2
      MVI A, 09H
      OUT (81H) PB        : Send data on port B to glow R3 and R4
      MVI C, 28H          : Load multiplier count (40ıo) for delay
      CALL DELAY          : Call delay subroutine
      MVI A, 12H
      OUT PA              : Send data on port A to glow Y1 and Y2
      OUT PB              : Send data on port B to glow Y3 and Y4
      MVI C, 0AH          : Load multiplier count (10ıo) for delay
      CALL DELAY          : Call delay subroutine
      JMP START
```

Delay Subroutine:

```
DELAY: LXI D, Count       : Load count to give 0.5 sec delay
BACK: DCX D               : Decrement counter
      MOV A, D
      ORA E               : Check whether count is 0
      JNZ BACK            : If not zero, repeat
      DCR C               : Check if multiplier zero, otherwise repeat
      JNZ DELAY
      RET                 : Return to main program
```

Assume Operating Frequency = 2 MHz

$$\text{Time for one T - state} = \frac{1}{2\,\text{MHz}} = 0.5\,\mu\text{sec}$$

$$\text{Count} = \frac{\text{Required Delay}}{\text{Time required for 1 loop}}$$

$$= \frac{0.5\,\text{sec}}{0.5\,\mu\text{sec} \times 24} \because \text{1 loop needs 24 T-states}$$

$$= 41666_{10} = \text{A2C2H}$$

70 Statement: Interface a Stepper Motor to the 8085 microprocessor system and write an 8085 assembly language program to control the Stepper Motor.

HARDWARE FOR STEPPER MOTOR CONTROL

A stepper motor is a digital motor. It can be driven by digital signal. Fig. shows the typical 2 phase motor rated 12V /0.67 A/ph interfaced with the 8085 microprocessor system using 8255. Motor shown in the circuit has two phases, with center-tap winding. The center taps of these windings are connected to the 12V supply. Due to this, motor can be excited by grounding four terminals of the two windings. Motor can be rotated in steps by giving proper excitation sequence to these windings. The lower nibble of port A of the 8255 is used to generate excitation signals in the proper sequence. These excitation signals are buffered using driver transistors. The transistors are selected such that they can source rated current for the windings. Motor is rotated by 1.80 per excitation.

INTERFACING SCHEME

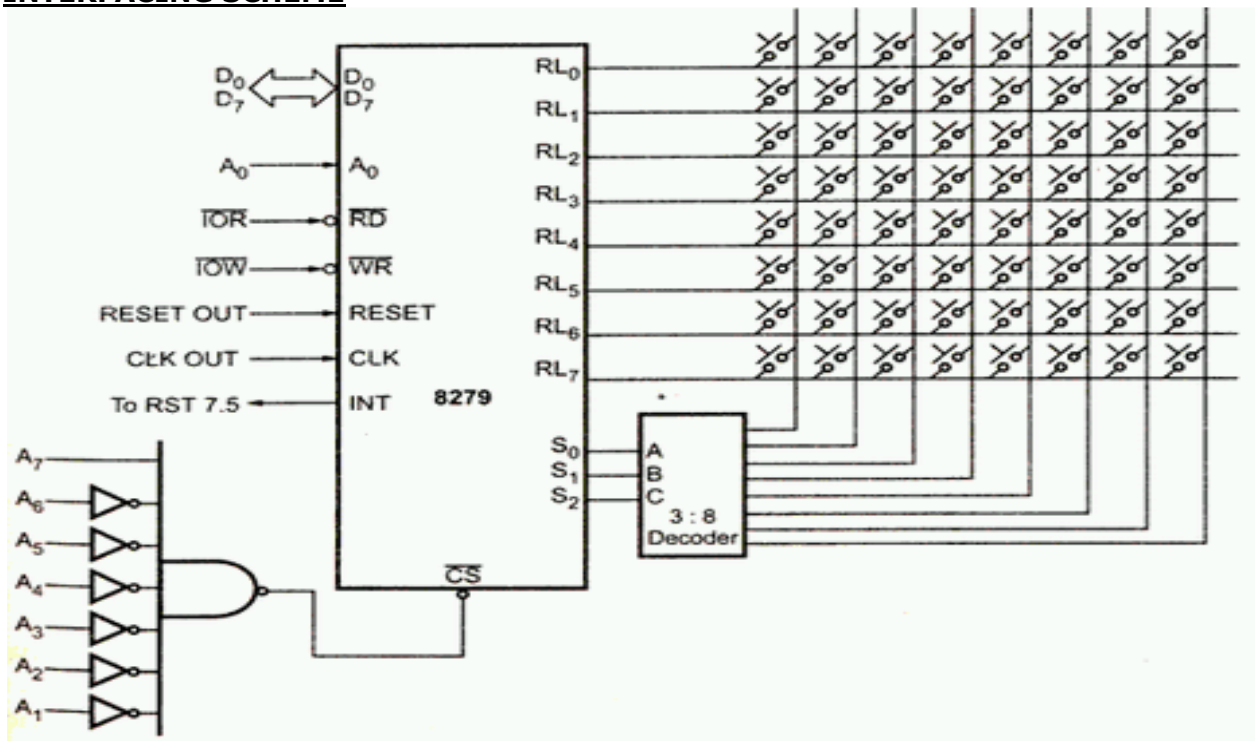


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

SOFTWARE FOR STEPPER MOTOR CONTROL

As port A is used as an output port, control word for 8255 is 80H.

Stepper Motor Control Program:

6000H Excite code DB 03H, 06H,
09H, OCH : This is the code sequence for clockwise rotation

Subroutine to rotate a stepper motor clockwise by 360° - Set the counts:

```
      MVI C, 32H          : Set repetition count to 50ıo
START: MVI B, 04H           : Counts excitation sequence
      LXI H, 6000H        : Initialize pointer
BACK1: MOV A, M           : Get the Excite code
      OUT PORTA           : Send Excite code
      CALL DELAY          : Wait
      INX H               : Increment pointer
      DCR B               : Repeat 4 times
      JNZ BACK l
      DCR C
      JNZ START           : Repeat 50 times
      RET
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
      MOV A, D
      ORA E
      JNZ Back
      RET
```

71 Statement: Interface a 64-key matrix keyboard to the 8085 microprocessor using 8255. Write an 8085 assembly language program to initialize 8255 and to read the key code.

HARDWARE FOR MATRIX KEYBOARD INTERFACE

Fig. shows a matrix keyboard with 64 keys connected to the 8085 microprocessor using 8255. A matrix keyboard reduces the number of connections, thus the number of interfacing lines. In this example, the keyboard with 64 keys, is arranged in 8 x 8 (8 rows and 8 columns) matrix. This requires sixteen lines from the microprocessor to make all the connections instead of 64 lines if the keys are connected individually. The interfacing of matrix keyboard requires two ports: one input port and other output port. Rows are connected to the input port, port A and columns are connected to the output port, port B.

INTERFACING SCHEME

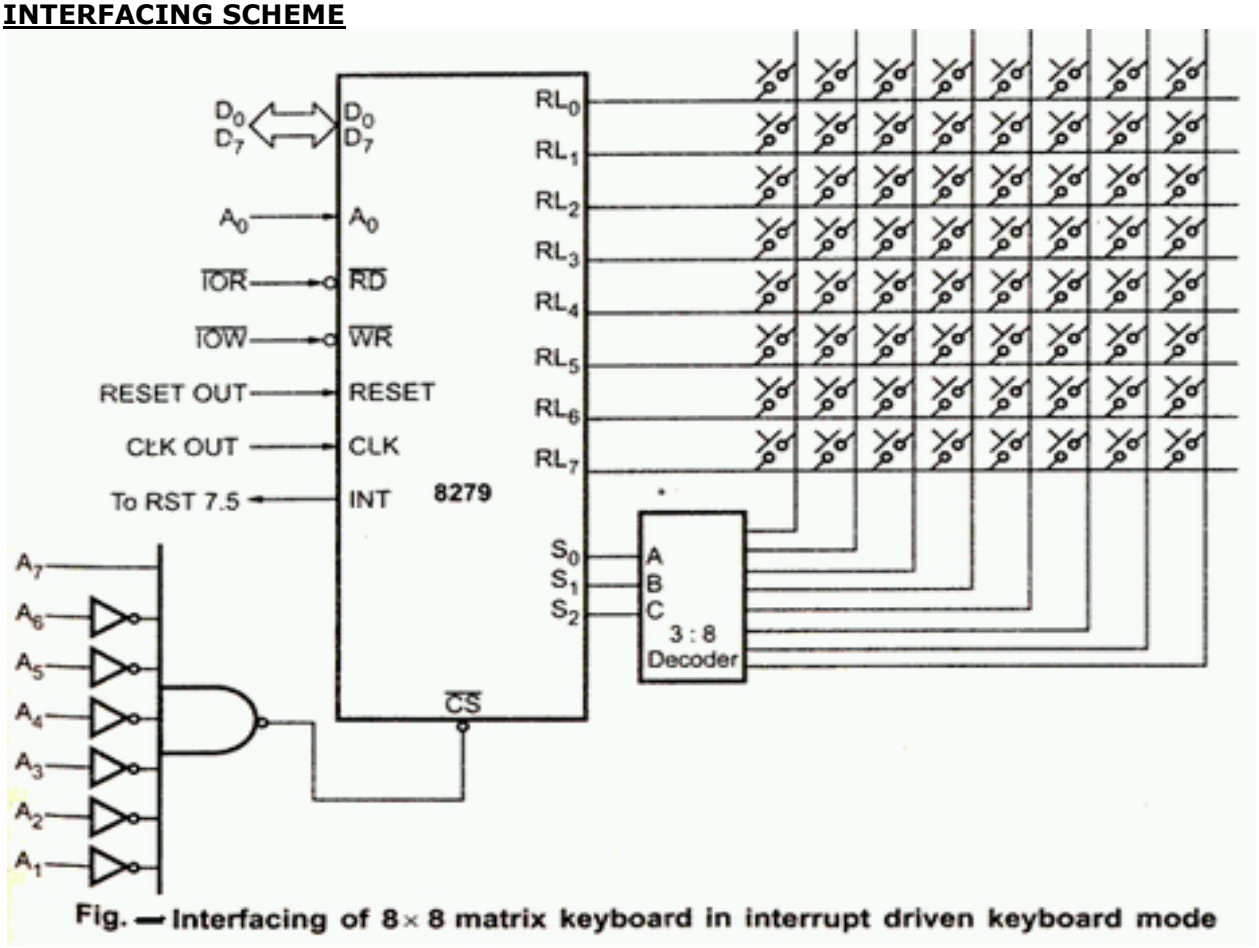


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

SOFTWARE FOR MATRIX KEYBOARD INTERFACE

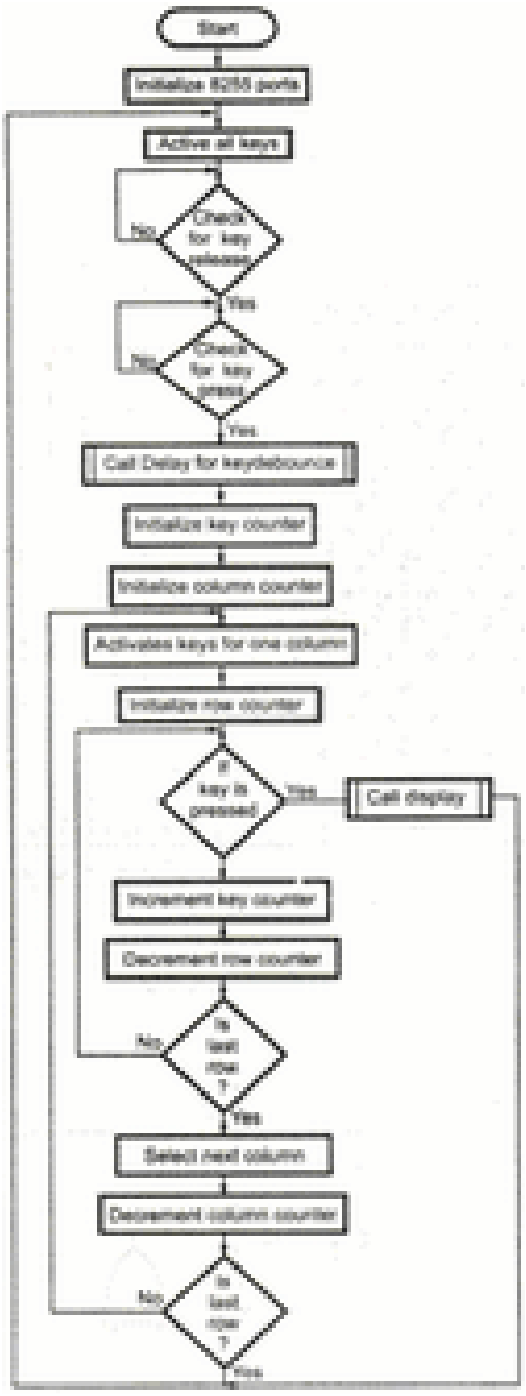
Source program

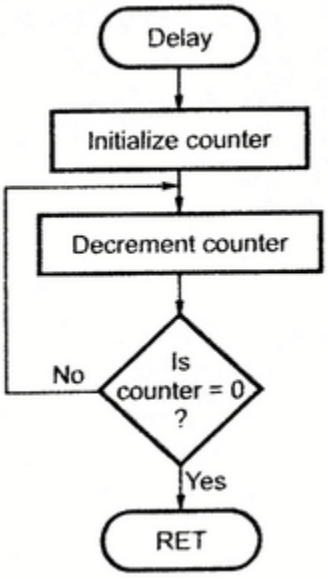
```
        MVI A, 90H          : Initialize Port A as input and
        OUT CR              : Port B as Output
START:  MVI A, 00           : Make all scan lines zero
        OUT PB
BACK:   IN PA
        CPI FF              : Check for key release
        JNZ BACK            : If not, wait for key release
        CALL DELAY          : Wait for key debounce
BACK 1: IN PA
```

```
        CPI FF                  : Check for key press
        JZ BACK 1               : If not, wait for key press
        CALL DELAY              : Wait for key debounce
        MVI L, 00H              : Initialize key counter
        MVI C, 08H
        MVI B, FEH              : Make one column low
NEXTCOL: MOV A, B
        OUT PB
        MVI D, 08H              : Initialize row counter
        IN PA                   : Read return line status
NEXTROW: RRC                    : Check for one row
        JNC DISPLAY             : If zero, goto display else continue
        INR L                   : Increment key counter
        DCR D                   : Decrement row counter
        JNZ NEXTROW             : Check for next row
        MOV A, B
        RLC                     : Select the next column
        MOV B, A
        DCR C                   : Decrement column count
        JNZ NEXTCOL             : Check for last column if not repeat
        JMP START               : Go to start

Delay subroutine:

Delay: LXI D, Count
Back: DCX D
        MOV A, D
        ORA E
        JNZ Back
        RET
```


Start
Initialize 8255 ports
Active all keys
No
Check for key release
Yes
No
Check for key press
Yes
Call Delay for keydebounce
Initialize key counter
Initialize column counter
Activates keys for one column
Initialize row counter
If key is pressed
Yes
Call display
Increment key counter
Decrement row counter
No
Is last row ?
Yes
Select next column
Decrement column counter
No
Is last row ?
Yes


Delay
Initialize counter
Decrement counter
No
Is counter = 0 ?
Yes
RET

72 Statement: Interface an 8-digit 7 segment LED display using 8255 to the 8085 microprocessor system and write an 8085 assembly language routine to display message on the display.

HARDWARE FOR EIGHT DIGIT SEVEN SEGMENT DISPLAY INTERFACE

Fig. shows the multiplexed eight 7-segment display connected in the 8085 system using 8255. In this circuit port A and port B are used as simple latched output ports. Port A provides the segment data inputs to the display and port B provides a means of selecting a display position at a time for multiplexing the displays. A0-A7 lines are used to decode the addresses for 8255. For this circuit different addresses are:

PA = 00H PB = 01H
PC = 02H CR = 03H.

The register values are chosen in Fig. such that the segment current is 80 mA. This current is required to produce an average of 10 mA per segment as the displays are multiplexed. In this type of display system, only one of the eight display position is 'ON' at any given instant. Only one digit is selected at a time by giving low signal on the corresponding control line. Maximum anode current is 560 mA (7-segments x 80 mA = 560 mA), but the average anode current is 70 mA.

INTERFACING SCHEME

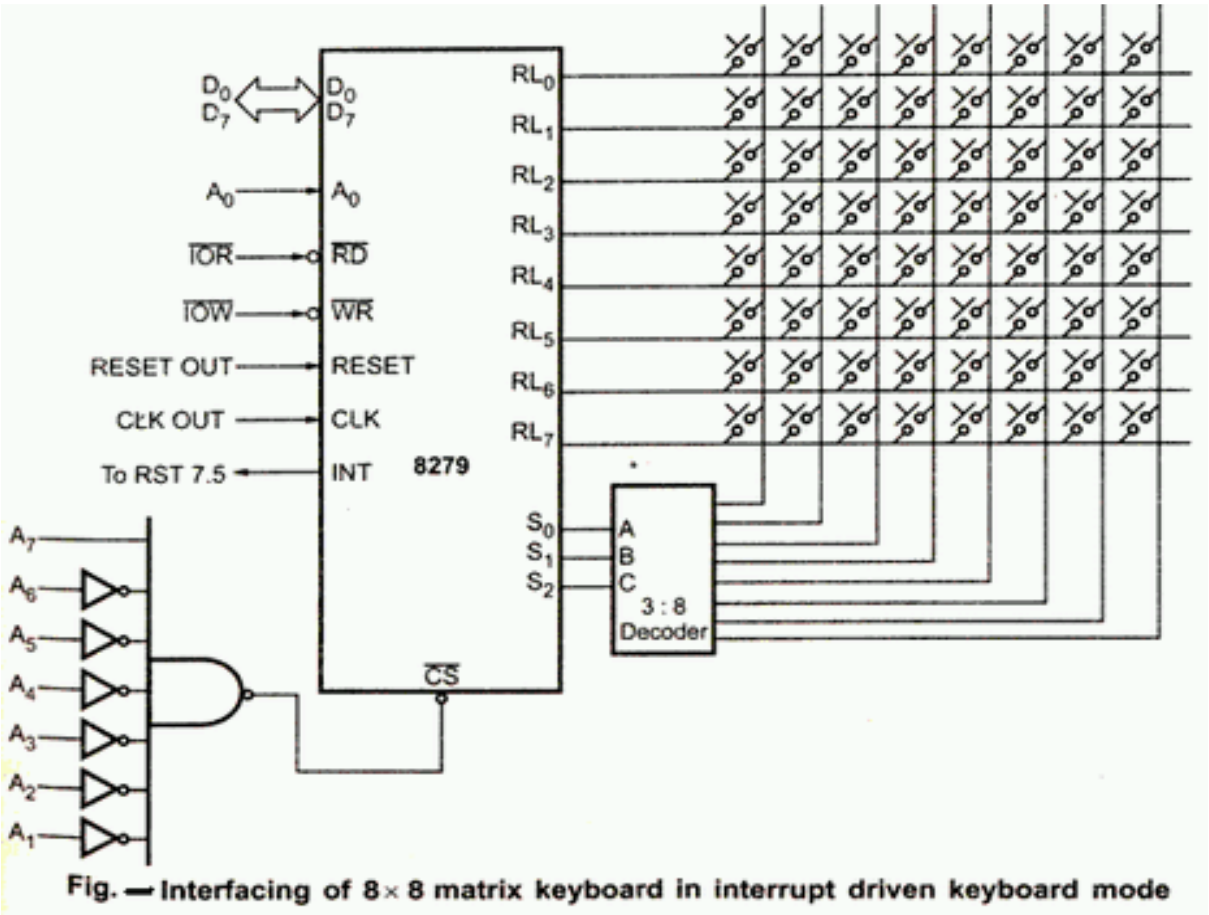


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

SOFTWARE FOR EIGHT DIGIT SEVEN SEGMENT DISPLAY INTERFACE

For 8255, Port A and B are used as output ports. The control word format of 8255 according to hardware connections is:

BSR	Mode A		P_A	P_{CU}	Mode B	P_B	P_{CL}	
1	0	0	0	X	0	0	X	= 80 H

Fig. — Control word format for 8255

Source program:

SOFTWARE TO INITIALIZE 8255:

```
MVI     A, 80H          : Load control word in AL
OUT     CR          : Load control      word in CR
```

SUBROUTINE TO DISPLAY MESSAGE ON MULTIPLEXED LED DISPLAY:

SET UP REGISTERS FOR DISPLAY:

```
MVI B, 08H              : load count
MVI C, 7FH              : load select pattern
LXI H, 6000B              : starting address of message
```

DISPLAY MESSAGE:

```
DISP 1: MOV A, C            : select digit
        OUT PB
        MOV A, M          : get data
        OUT PA          : display data
        CALL DELAY        : wait for some time
DISP 1: MOV A, C
        RRC
        MOV C, A            : adjust selection pattern
        INX H
        DCR B               : Decrement count
        JNZ DISP 1          : repeat 8 times
        RET
```

Note: This "display message subroutine" must be called continuously to display the 7-segment coded message stored in the memory from address 6000H.

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
      MOV A, D
      ORA E
      JNZ Back
      RET
```

73 Statement: Interface an 8 x 8 matrix keyboard to 8085 through 8279 in 2-key lockout mode and write an assembly language program to read keycode of the pressed key. The external clock frequency is 2MHz. Use I/O mapped I/O technique. (Dont use any Interrupts)

HARDWARE FOR 8 x 8 MATRIX KEYBOARD INTERFACE

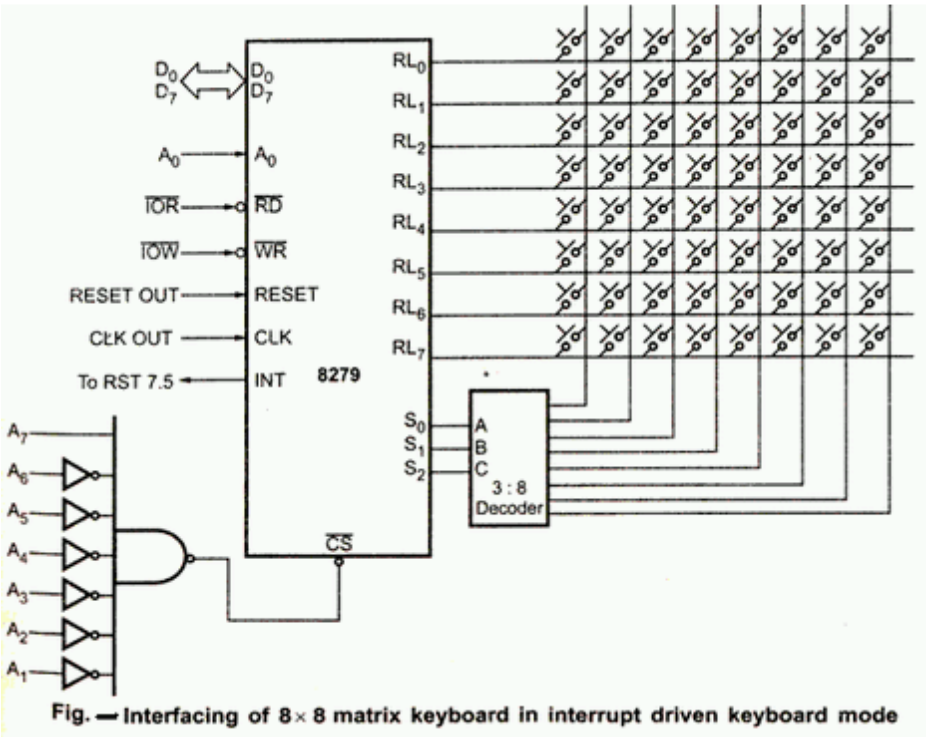


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

SOFTWARE FOR 8 x 8 MATRIX KEYBOARD INTERFACE

I/O Map

Data/Control Register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data Register	1	0	0	0	0	0	0	0	80 H
Control Register	1	0	0	0	0	0	0	1	81 H

SOFTWARE FOR 8 x 8 MATRIX KEYBOARD INTERFACE

The three steps needed to write the software are:

Step 1: Find keyboard/display command word.

0	0	0	D	D	K	K	K	
0	0	0	X	X	0	0	0	= 00H

Step 2: Find program clock command word

			P	P	P	P	P	
0	0	1	1	0	1	0	0	= 34H

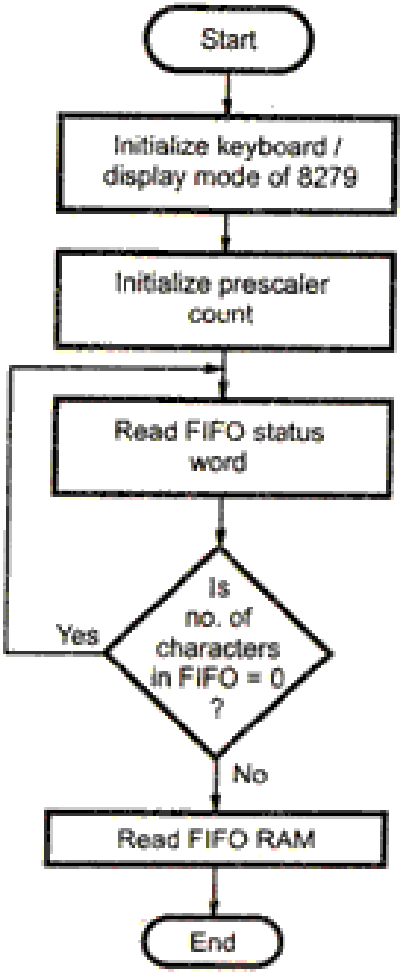
Step 3: Find Read FIFO/sensor RAM command word.

			AI		A	A	A	
0	1	0	0	X	0	0	0	= 40H

Source program:

```
      MVI A, 00H          : Initialize keyboard/display
      OUT 81H           : in encoded scan keyboard-2 keylockout mode
      MVI A, 34H
      OUT 81H           : Initialize prescaler count
BACK: IN 81H            : Read FIFO status word
      ANI 07H           : Mask bit B3 to B7
      JZ BACK           : If 0, key is not pressed wait for key press else read FIFO
RAM
      MVI A, 40H          : Initialize 8279 in read
      OUT 81H           : FI FO RAM mode
      IN 80H          : Read FIFO RAM (keycode)
      HLT               : Stop program execution.
```

FLOWCHART



74 Statement: Interface an 8 x 8 matrix keyboard to 8085 through 8279 in 2-key lockout mode and write an assembly language program to read keycode of the pressed key. The external clock frequency is 2MHz. Use I/O mapped I/O technique.

HARDWARE FOR 8 x 8 MATRIX KEYBOARD INTERFACE(With Interrupt)

Fig. shows the interfacing of 8 x 8 matrix keyboard in interrupt driven keyboard mode. In the interrupt driven mode interrupt line from 8279 is connected to the one of the interrupt input of 8085 except INTR. Here, INT line from 8279 is connected to the interrupt RST 7.5 of 8085. Other signal connections are same as in the non interrupt mode.

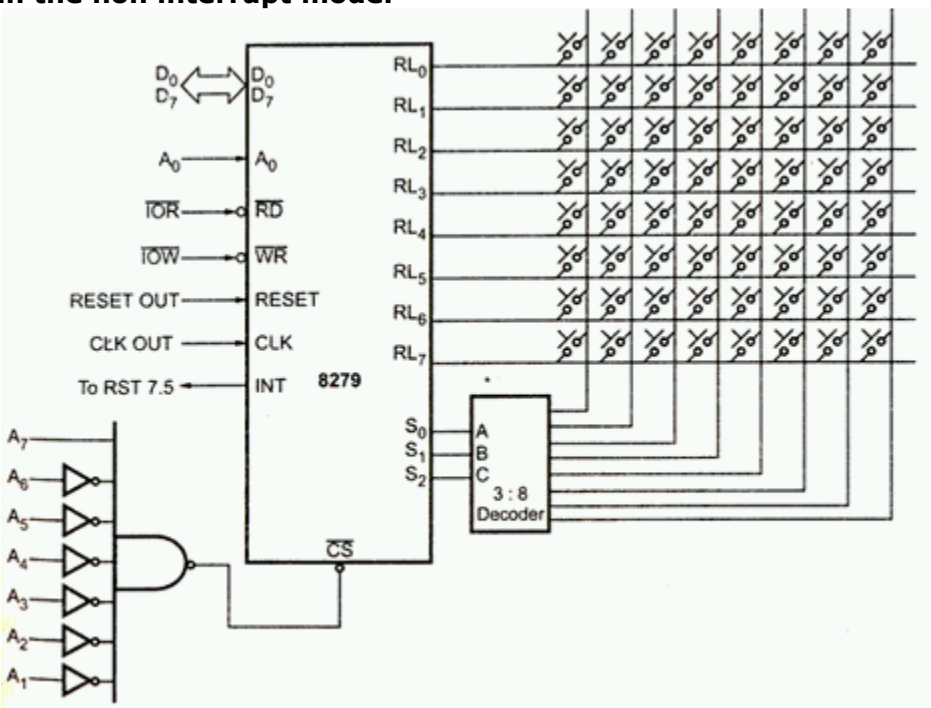


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

I/O Map

Data/Control Register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data Register	1	0	0	0	0	0	0	0	80 H
Control Register	1	0	0	0	0	0	0	1	81 H

SOFTWARE FOR 8 x 8 MATRIX KEYBOARD INTERFACE(With Interrupt)

The three steps needed to write the software are:

Step 1: Find keyboard/display command word.

0	0	0	D	D	K	K	K	
0	0	0	X	X	0	0	0	= 00H

Step 2: Find program clock command word

				P	P	P	P	P
0	0	1	1	0	1	0	0	= 34H

Step 3: Find Read FIFO/sensor RAM command word.

			AI		A	A	A
0	1	0	0	X	0	0	0	= 40H

Source program:

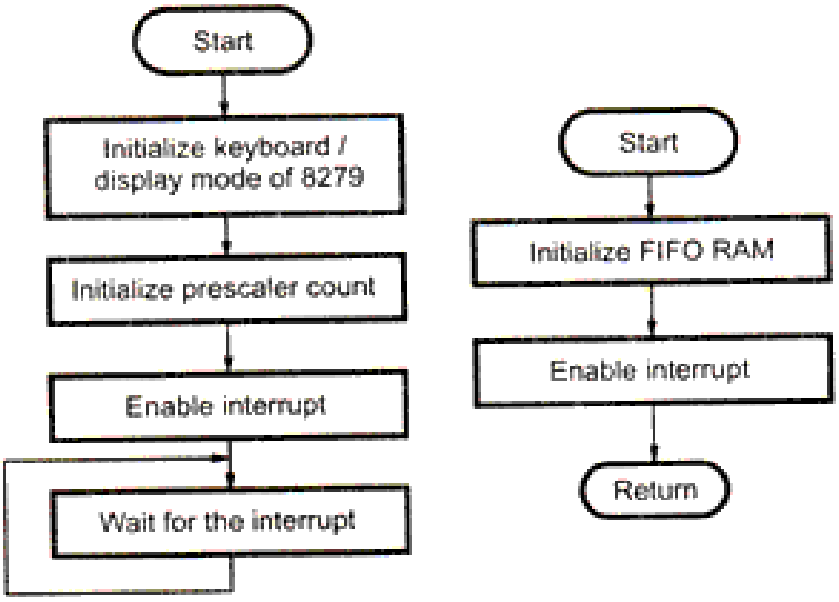
```
    MVI A, 00H          : Initialize keyboard/display in encoded
    OUT 81H             : scan keyboard 2 key lockout mode
    MVI A, 34H
    OUT 81H             : Initialize prescaler count
    MVI A, 0BH          : Load mask pattern to enable RST 7.5
    SIM                 : mask other interrupts
    EI                  : Enable Interrupt
HERE: JMP HERE          : Wait for the interrupt
```

Interrupt Subroutine:

```
    MVI A, 40H          : Initialize 8279 in read FIFO
    OUT    81H          : RAM    mode
    IN 80H              : Read FIFO RAM (keycode)
    EI                  : Enable Interrupt
    RET                 : Return to main program
```

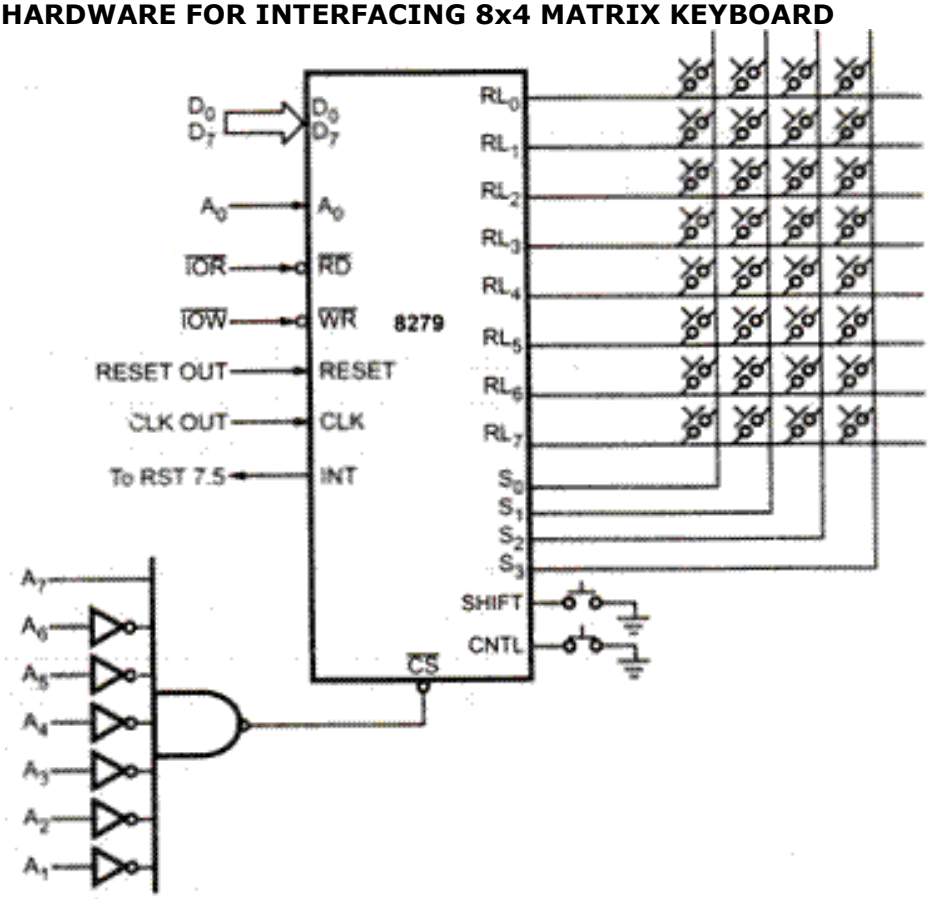
Note: In the interrupt driven keyboard, when key is pressed, key code is loaded into FIFO RAM and interrupt is generated. This interrupt signal is used to tell CPU that there is a keycode in the FIFO RAM. CPU then initiates read command with in the interrupt service routine to read key code from the FIFO RAM.

FLOWCHART



75 Statement:
Interface an 8 x 4 matrix keyboard to 8085 through 8279.

HARDWARE FOR INTERFACING 8x4 MATRIX KEYBOARD



NOTE: As keyboard is having 8 rows and 4 columns, only 4 scan lines are required and we can avoid external decoder to generate scan lines by selecting decoded scan keyboard mode.

SOFTWARE FOR INTERFACING 8x4 MATRIX KEYBOARD
Source program:

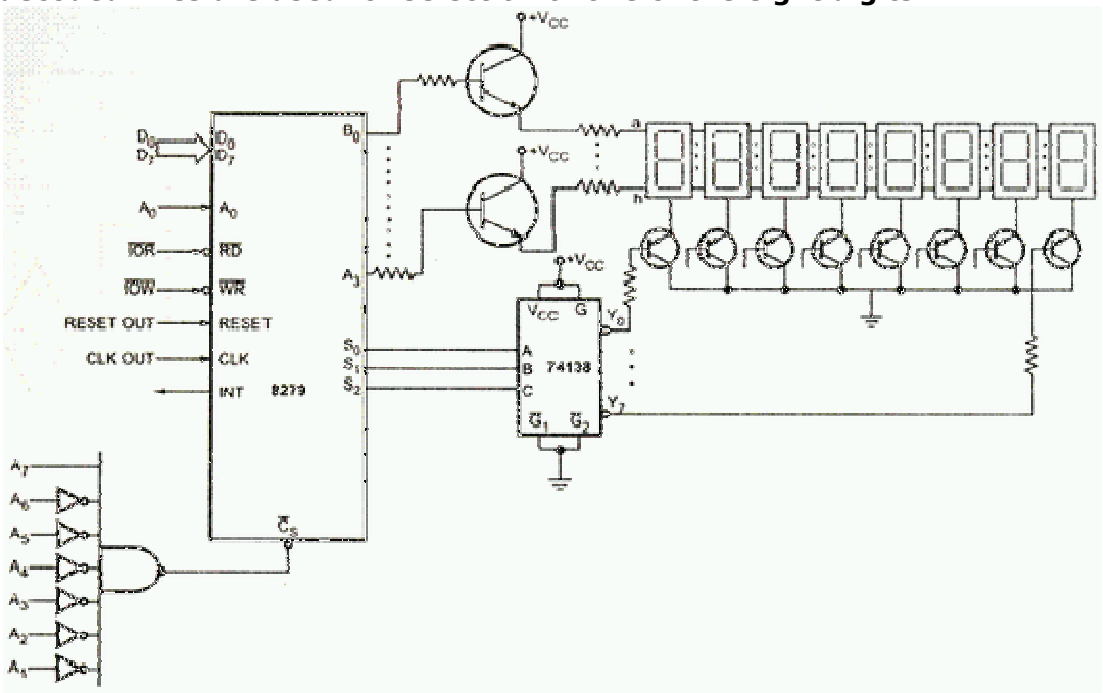
```
    MVI A, 00H          : Initialize keyboard/display in encoded
    OUT 81H             : scan keyboard 2 key lockout mode
    MVI A, 34H
    OUT 81H             : Initialize prescaler count
    MVI A, 0BH          : Load mask pattern to enable RST 7.5
    SIM                 : mask other interrupts
    EI                  : Enable Interrupt
HERE: JMP HERE          : Wait for the interrupt
```

Interrupt Subroutine:

```
    MVI A, 40H          : Initialize 8279 in read FIFO
    OUT    81H          : RAM    mode
    IN 80H              : Read FIFO RAM (keycode)
    EI                  : Enable Interrupt
    RET                 : Return to main program
```

76 Statement:
Interface 8/7-segment digits (common cathode) to 8085 through 8279 and write an 8085 assembly language program to display 1 to 8 on the eight seven segment digits. External clock frequency is 3 MHz.

HARDWARE FOR EIGHT SEVEN SEGMENT DIGITS INTERFACE
Fig. shows the interfacing of eight 7-segment digits to 8085 through 8279. As shown in the figure eight display lines (Bo-B3 and Ao-A3) are buffered with the help of transistor and used to drive display digits. These buffered lines are connected in parallel to all display digits. So, Sl and S2 lines are decoded and decoded lines are used for selection of one of the eight digits.



SOFTWARE FOR EIGHT SEVEN SEGMENT DIGITS INTERFACE
To display 1 to 8 numbers on the eight 7-segment digits we have to load 7-segment codes for 1 to 8 numbers in the corresponding display locations.

Number	h	g	f	e	d	c	b	a	Code
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F

Table — 7-Segment codes for common cathode display

The three steps needed to write the software are:
Step 1: Find keyboard/display command word.

			D	D	K	K	K	
0	0	0	0	0	0	0	0	= 00H

Step 2: Find program clock command word

			P	P	P	P	P	
0	0	1	1	1	1	1	0	= 3EH

Step 3: Find display RAM command word.

			AI	A_3	A_2	A_1	A_0	
1	0	0	1	0	0	0	0	= 90H

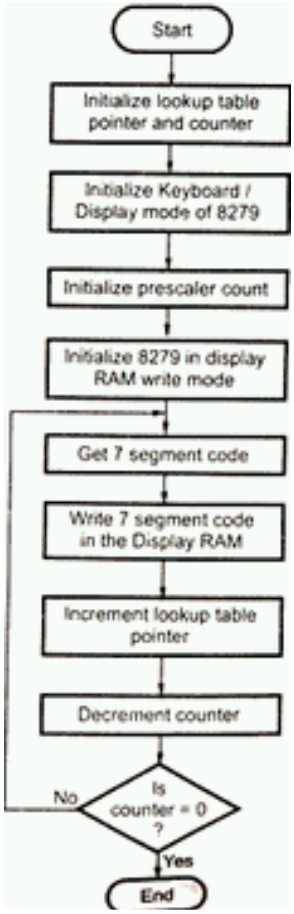
Source program:

```
    LXI B, 6200B        : Initialize lookup table pointer
    MVI C, 08H          : Initialize counter
    MVI A, 00H          : Initialize keyboard/display
    OUT 8IH             : Mode
    MVI A, 3EH          : Initialize prescaler count
    OUT 8IH
    MVI A, 90H          : Initial size 8279 in write Display
    OUT 8IH             : RAM-mode
BACK : MOV A, M         : Get the 7-segment code
    OUT 80H             : Write 7-segment code in display RAM
    INX H               : Increment lookup table pointer
    DCR C               : Decrement counter
    JNZ BACK            : if count = 0 stop, otherwise go to back
    HLT                 : Stop program execution
```

LOOK UP TABLE

Memory Address	Contents
6200	66
6201	5B
6202	4F
6203	66
6204	6D
6205	7D
6206	07
6207	7F

FLOWCHART



77 Statement:
Interface 4 x 4 matrix keyboard and 4 digit 7-segment display and write an tssembly language program to read keycode of the pressed key and display same key on :he 7 segment display.

HARDWARE FOR 4x4 MATRIX KEYBOARD & 4 DIGIT 7 SEGMENT DISPLAY INTERFACE
Fig. shows interfacing diagram. Here, 4 scan lines are sufficient to scan matrix keyboard and to select display digits. Hence decoded mode is used.

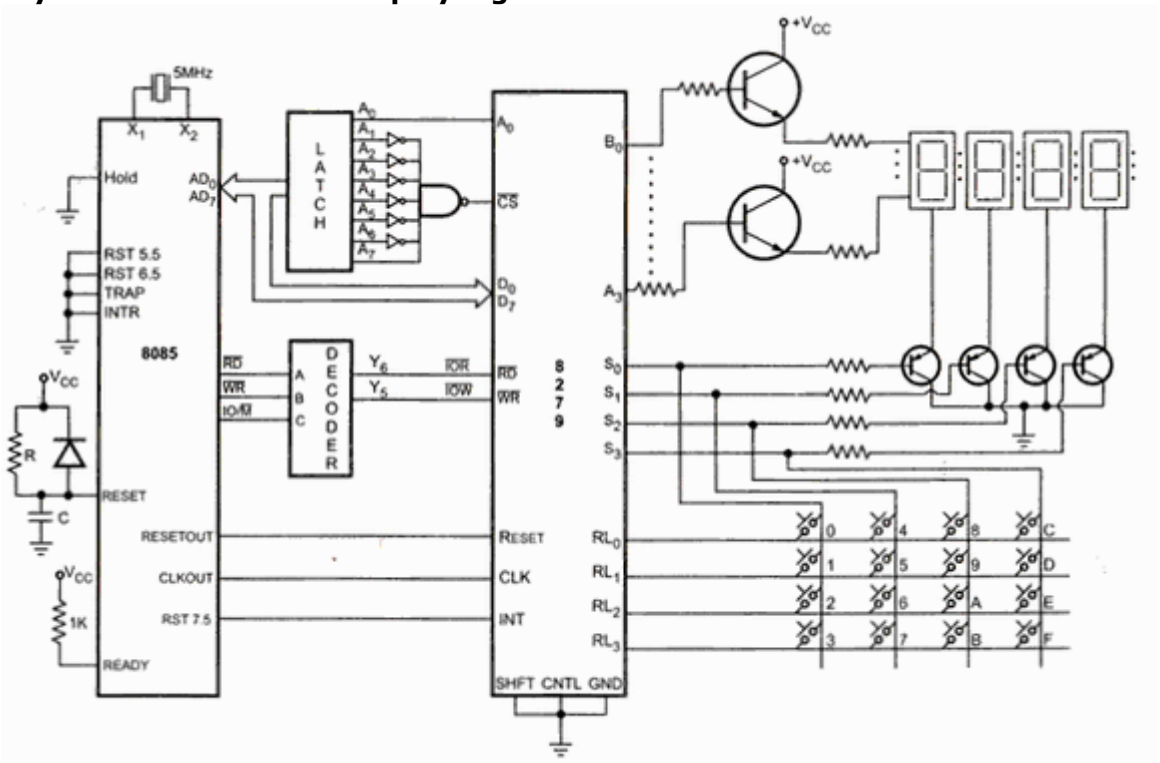


Fig. — Keyboard and display interfacing in decoded mode using 8279

SOFTWARE FOR 4x4 MATRIX KEYBOARD & 4 DIGIT 7 SEGMENT DISPLAY INTERFACE

The three steps needed to write the software are:

Step 1: Find keyboard/display command word.

			D	D	K	K	K	
0	0	0	0	0	0	0	1	= 01H

Step 2: Find program clock command word

			P	P	P	P	P	
0	0	1	1	1	0	0	1	= 39H

Step 3: Find Read FIFO RAM command word.

			A	AI	A	A	A	
0	1	0	0	X	0	0	0	= 40H

Step 3: Find Write FIFO RAM command word.

			AI	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	= 80H

Source program:

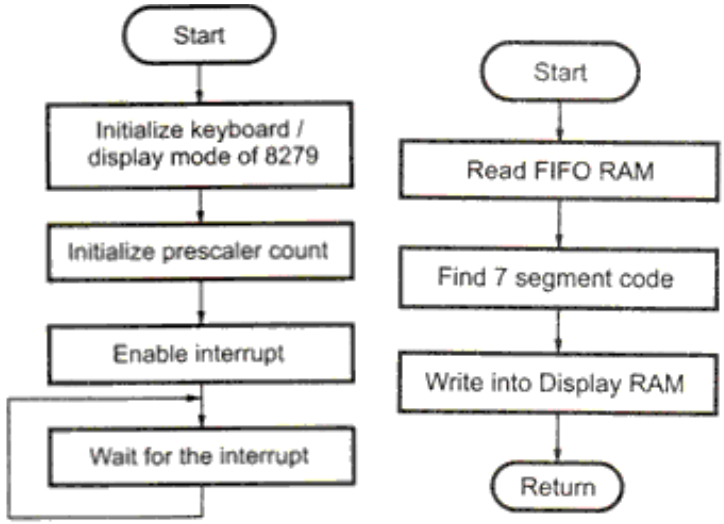
```
      MVI A, 00H          : Initialize keyboard/display in encoded
      OUT 81H             : scan keyboard 2 key lockout mode
      MVI A, 34H
      OUT 81H             : Initialize prescaler count
      MVI A, 0BH          : Load mask pattern to enable RST 7.5
      SIM                 : mask other interrupts
      EI                  : Enable Interrupt
HERE: JMP HERE            : Wait for the interrupt
```

Interrupt service routine

```
      MVI A, 40H          : Initialize 8279 in read FIFO RAM mode
      OUT 81H
      IN 80H              : Get keycode
      MVI H, 62H          : Initialize memory pointer to point
      MOV L, A            : 7-Segment code
      MVI A, 80H          : Initialize 8279 in write display RAM mode
      OUT 81H
      MOV A, M            : Get the 7 segment code
      OUT 80H             : Write 7-segment code in display RAM
      EI                  : Enable interrupt
      RET                 : Return to main program
```

FLOWCHARTS

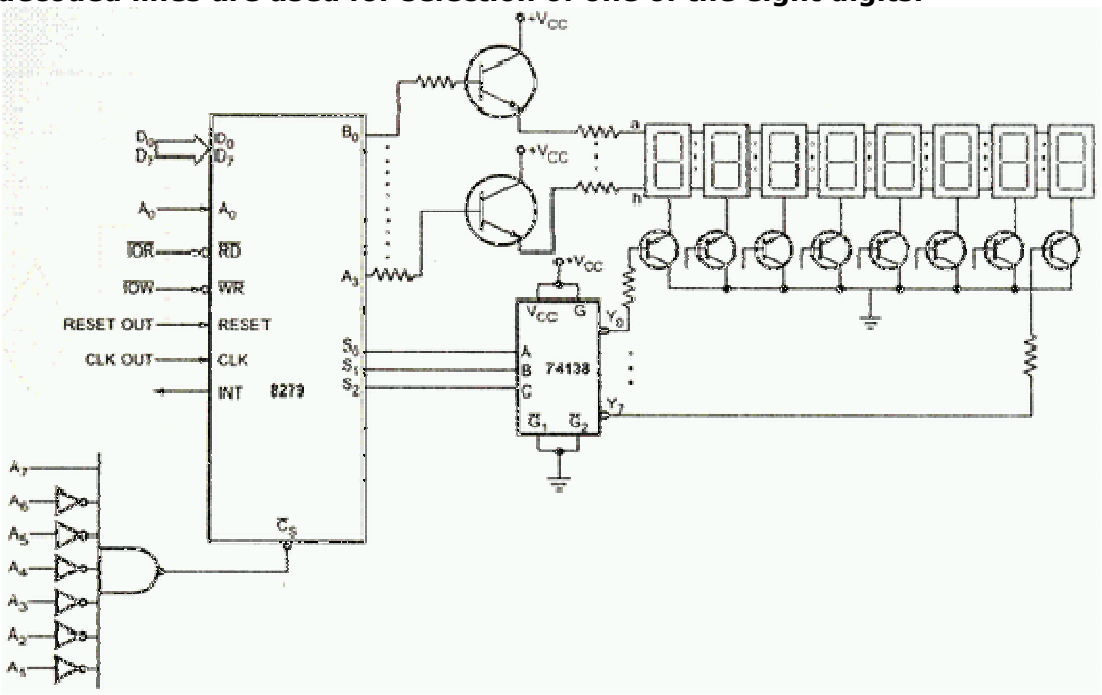
Source Program and Interrupt Service Routine



78 Statement: Write an assembly language program to roll message 'HELL0123' from right to left

HARDWARE FOR ROLLING HELLO123

Fig. shows the interfacing of eight 7-segment digits to 8085 through 8279. As shown in the figure eight display lines (Bo-B3 and Ao-A3) are buffered with the help of transistor and used to drive display digits. These buffered lines are connected in parallel to all display digits. So, Sl and S2 lines are decoded and decoded lines are used for selection of one of the eight digits.



SOFTWARE FOR ROLLING HELLO123

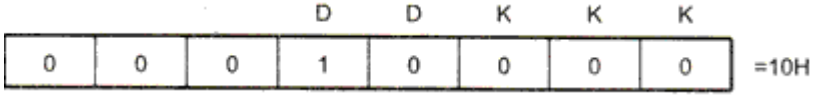
To roll above message we have to load 7-segment codes for characters within the message and it is necessary to configure 8279 in right entry mode

Character	h	g	f	e	d	c	b	a	Code
H	0	1	1	1	0	1	1	0	76H
E	0	1	1	1	1	0	0	1	79H
L	0	0	1	1	1	0	0	0	38H
L	0	0	1	1	1	0	0	0	38H
0	0	0	1	1	1	1	1	1	3FH
1	0	0	0	0	0	1	1	0	06H
2	0	1	0	1	1	0	1	1	5BH
3	0	1	0	0	1	1	1	1	4FH

Table – 7-segment codes for given message

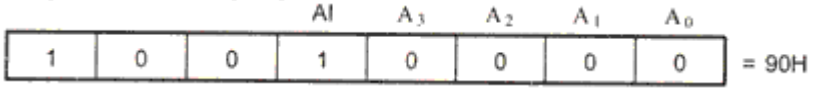
The three steps needed to write the software are:

Step 1: Find keyboard/display command word.

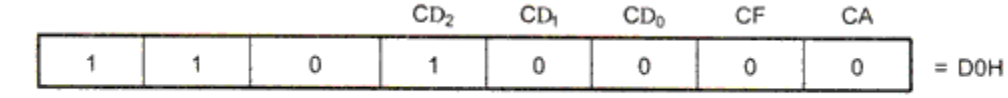


Step 2: Find program clock command word

Step 3: Find display RAM command word.



Clear command word.



CD_0-CD_1 = 00 (Blanking code)

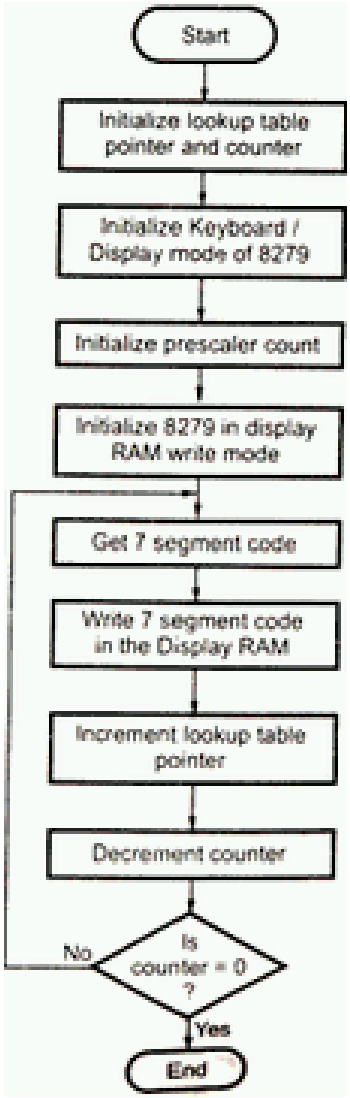
Source program:

```
    LXI B, 6200B        : Initialize lookup table pointer
    MVI C, 08H          : Initialize counter
    MVI A, 10H          : Initialize keyboard/display in right entry mode
    OUT 8IH             : Mode
    MVI A, 3EH          : Initialize prescaler count
    OUT 8IH
    MVI A, D0H          : Clear Display
    OUT 8IH
    MVI A, 90H          : Initialize 8279 in write display
    OUT 81H             : RAM mode
BACK : MOV A, M         : Get the 7-segment code
    OUT 80H             : Write 7-segment code in display RAM
    INX H               : Increment lookup table pointer
    DCR C               : Decrement counter
    JNZ BACK            : if count = 0 stop, otherwise go to back
    HLT                 : Stop program execution
```

LOOK UP TABLE

Memory address	Contents
6200H	76H
6201H	79H
6202H	38H
6203H	38H
6204H	3FH
6205H	06H
6206H	58H
6207H	4FH

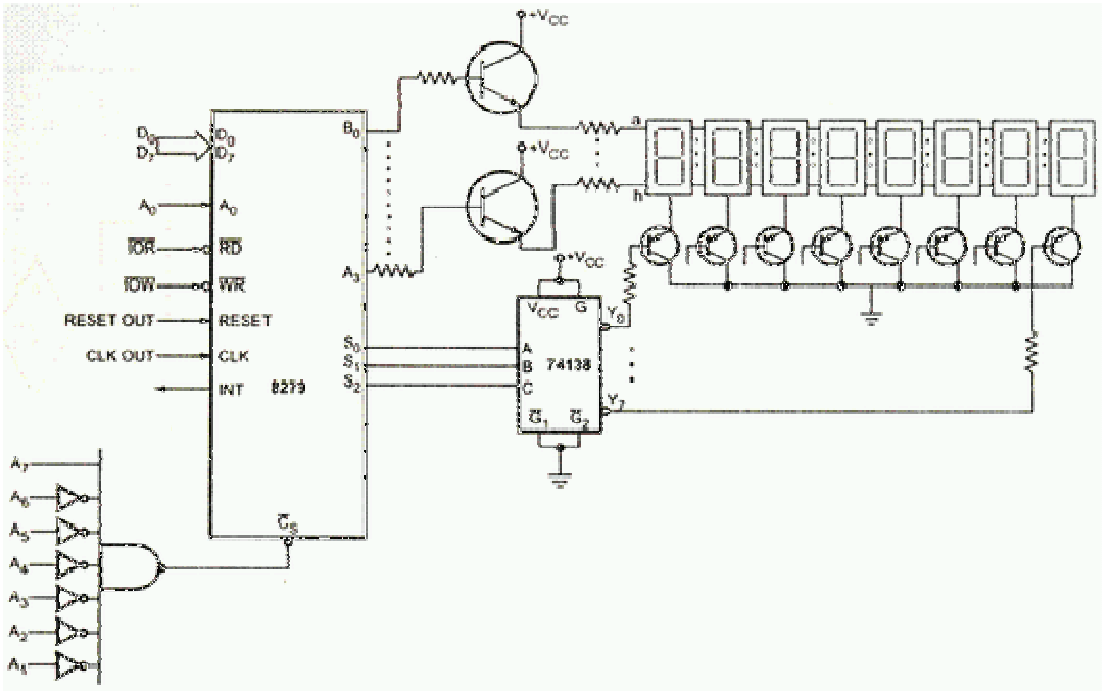
FLOWCHART



79 Statement: Write an assembly language program to your name from right to left

HARDWARE FOR ROLLING HELLO123

Fig. shows the interfacing of eight 7-segment digits to 8085 through 8279. As shown in the figure eight display lines (Bo-B3 and Ao-A3) are buffered with the help of transistor and used to drive display digits. These buffered lines are connected in parallel to all display digits. So, Sl and S2 lines are decoded and decoded lines are used for selection of one of the eight digits



SOFTWARE FOR ROLLING THE NAME - J.BINU

To roll the above namewe have to load 7-segment codes for characters within the message and it is necessary to configure 8279 in right entry mode
The three steps needed to write the software are:

Step 1: Find keyboard/display command word.

			D	D	K	K	K	
0	0	0	1	0	0	0	0	=10H

Step 2: Find program clock command word

			P	P	P	P	P	
0	0	1	1	1	1	1	0	= 3EH

Step 3: Find display RAM command word.

			AI	A_3	A_2	A_1	A_0	
1	0	0	1	0	0	0	0	= 90H

Clear command word.

			CD_2	CD_1	CD_0	CF	CA	
1	1	0	1	0	0	0	0	= D0H

CD_0-CD_1 = 00 (Blanking code)

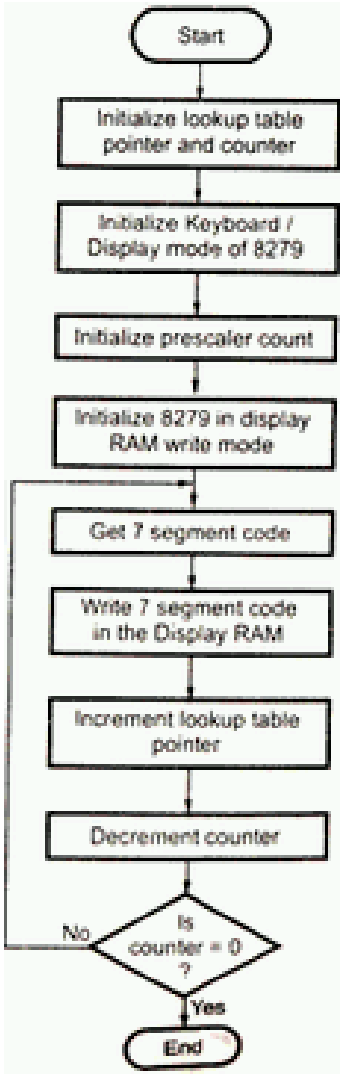
Source program:

```
    LXI B, 6200B        : Initialize lookup table pointer
    MVI C, 08H          : Initialize counter
    MVI A, 10H          : Initialize keyboard/display in right entry mode
    OUT 8IH             : Mode
    MVI A, 3EH          : Initialize prescaler count
    OUT 8IH
    MVI A, D0H          : Clear Display
    OUT 8IH
    MVI A, 90H          : Initialize 8279 in write display
    OUT 81H             : RAM mode
BACK : MOV A, M         : Get the 7-segment code
    OUT 80H             : Write 7-segment code in display RAM
    INX H               : Increment lookup table pointer
    DCR C               : Decrement counter
    JNZ BACK            : if count = 0 stop, otherwise go to back
    HLT                 : Stop program execution
```

LOOK UP TABLE

Memory address	Contents
6200H	1FH
6201H	80H
6202H	BFH
6203H	06H
6204H	B7H
6205H	BEH
6206H	OOH
6207H	OOH

FLOWCHART


Start
Initialize lookup table pointer and counter
Initialize Keyboard / Display mode of 8279
Initialize prescaler count
Initialize 8279 in display RAM write mode
Get 7 segment code
Write 7 segment code in the Display RAM
Increment lookup table pointer
Decrement counter
Is counter = 0 ?
No
Yes
End