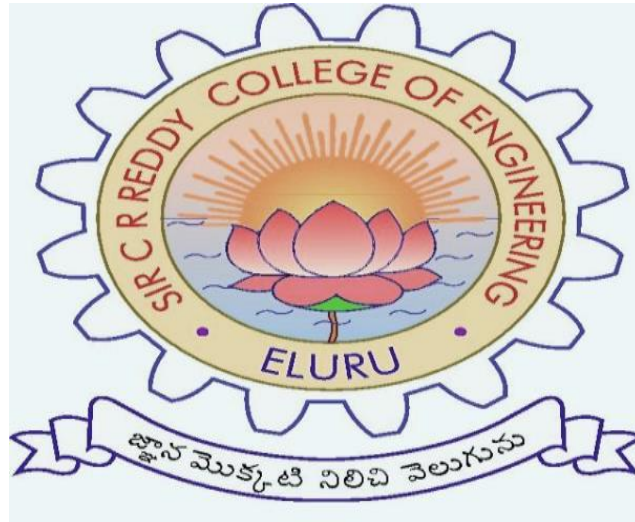# SIR C.R.REDDY COLLEGE OF ENGINEERING
ELURU – 534007

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DIGITAL ELECTRONICS & MICROPROCESSOR
LAB MANUAL

2/4 CSE: II- SEMESTER

Faculty:

B.Homer Benny   (Section-'A')

S.K.Chaitanya.R (Section-'B')

S.Jaya Prakash   (Section-'C')

DIGITAL ELECTRONICS & MICROPROCESSOR LAB

2/4 CSE: II - SEMESTER

INDEX

EXPERIMENT: 1   **STUDY OF BASIC GATES**

AIM: To study and verify the truth table of logic gates

LEARNING OBJECTIVE:
   •   Identify various ICs and their specification.

COMPONENTS REQUIRED:
   •   Logic gates (IC) trainer kit.
   •   Connecting patch chords.
   •   IC 7400, IC 7408, IC 7432, IC 7406, IC 7402, IC 7404, IC 7486

THEORY:

The basic logic gates are the building blocks of more complex logic circuits. These logic gates perform the basic Boolean functions, such as AND, OR, NAND, NOR, Inversion, Exclusive-OR, Exclusive-NOR. Fig. below shows the circuit symbol, Boolean function, and truth. It is seen from the Fig that each gate has one or two binary inputs, A and B, and one binary output, C. The small circle on the output of the circuit symbols designates the logic complement. The AND, OR, NAND, and NOR gates can be extended to have more than two inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

These basic logic gates are implemented as small-scale integrated circuits (SSICs) or as part of more complex medium scale (MSI) or very large-scale (VLSI) integrated circuits. Digital IC gates are classified not only by their logic operation, but also the specific logic-circuit family to which they belong. Each logic family has its own basic electronic circuit upon which more complex digital circuits and functions are developed. The following logic families are the most frequently used.

TTL   Transistor-transistor logic

ECL   Emitter-coupled logic

MOS   Metal-oxide semiconductor
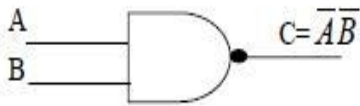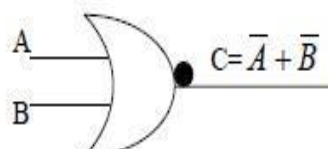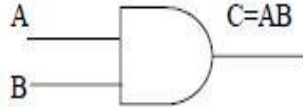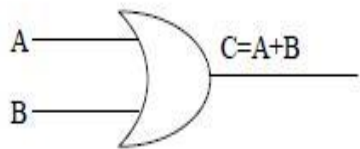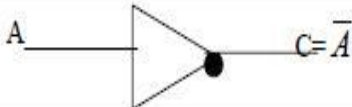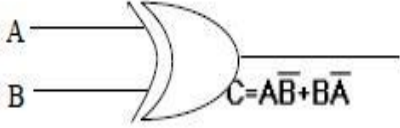
CMOS   Complementary metal-oxide semiconductor

TTL and ECL are based upon bipolar transistors. TTL has a well established popularity among logic families. ECL is used only in systems requiring high-speed operation. MOS and CMOS, are based on field effect transistors. They are widely used in large scale integrated circuits because of their high

component density and relatively low power consumption. CMOS logic consumes far less power than MOS logic. There are various commercial integrated circuit chips available. TTL ICs are usually distinguished by numerical designation as the 5400 and 7400 series.

PROCEDURE:
1. Check the components for their working.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. Provide the input data via the input switches and observe the output on output LEDs

S.NO	GATE	SYMBOL	INPUTS		OUTPUT
			A	B	C
1.	NAND IC 7400	A, B $C=\overline{AB}$	0	0	1
			0	1	1
			1	0	1
			1	1	0
2.	NOR IC 7402	A, B $C=\overline{A}+\overline{B}$	0	0	1
			0	1	0
			1	0	0
			1	1	0
3.	AND IC 7408	A, B $C=AB$	0	0	0
			0	1	0
			1	0	0
			1	1	1
4.	OR IC 7432	A, B $C=A+B$	0	0	0
			0	1	1
			1	0	1
			1	1	1
5.	NOT IC 7404	A $C=\overline{A}$	1	-	0
			0	-	1
6.	EX-OR IC 7486	A, B $C=A\overline{B}+B\overline{A}$	0	0	0
			0	1	1
			1	0	1
			1	1	0

EXPERIMENT: 2   **REALIZATION OF GATES BY USING UNIVERSAL BUILDING BLOCKS**

Aim : To Realize AND,OR,NOT,EX-OR and EX-NOR gates by using only NAND and only NOR gates

Apparatus and Components :

S.No	Name	Quantity
1.	Digital trainer	1
2.	IC 7400	2
3.	IC 7402	2

Procedure :

Using NAND Gates:
1. Derive truth table
2. Realize expression of AND gate by using number of NAND gates
3. Connect the circuit
4. Verify the truth table
5. Repeat above steps for OR, NOT, EX-OR and EX-NOR gates

Using NOR Gates:
1. Derive truth table
2. Realize expression of AND gate by using number of NOR gates
3. Connect the circuit
4. Verify the truth table
5. Repeat above steps for OR, NOT, EX-OR and EX-NOR gates

Result :

Basic gates are realized by using Universal building blocks.

Circuit Diagram :

Realization of AND gate using only NAND gates
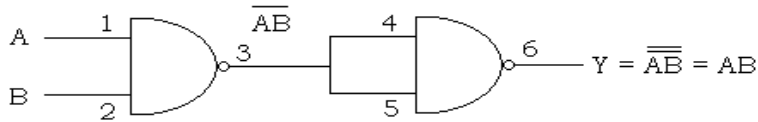
A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

Realization of OR gate using only NAND gates
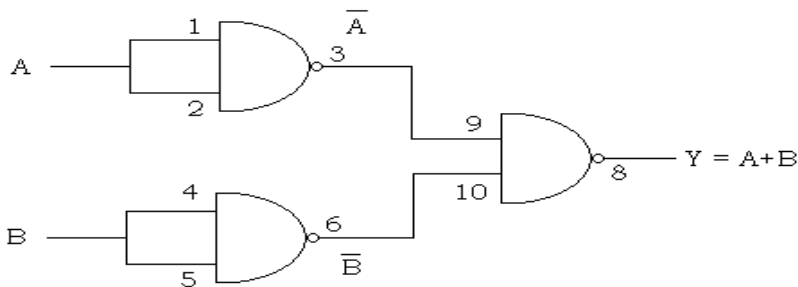
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

Realization of NOT gate using only NAND gates
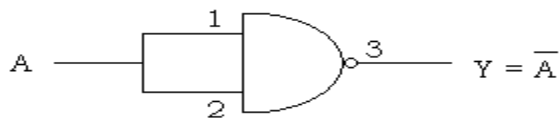
A	Y = $\overline{A}$
0	1
1	0

Realization of EX-OR gate using only NAND gates

A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

6

Realization of EX-NOR gate using only NAND gates

A	B	Y = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

Realization of AND gate using only NOR gates
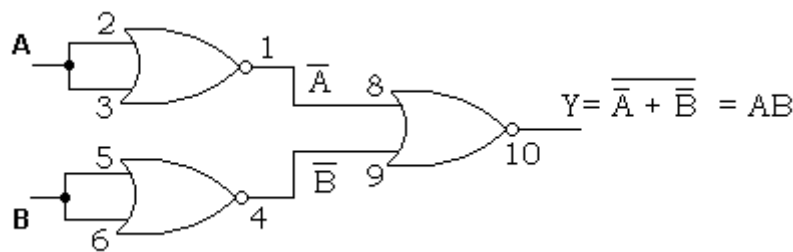
$Y = \overline{\overline{A} + \overline{B}} = AB$

A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

Realization of OR gate using only NOR gates

$Y = A+B$
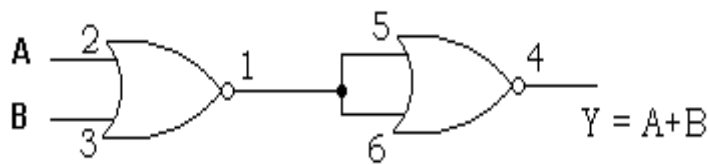
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

Realization of NOT gate using only NOR gates

$Y = \overline{A}$

A	Y = $\overline{A}$
0	1
1	0

7
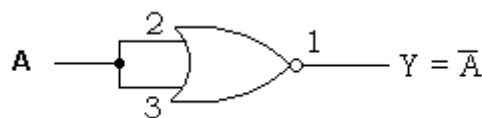
Realization of EX-OR gate using only NOR gates
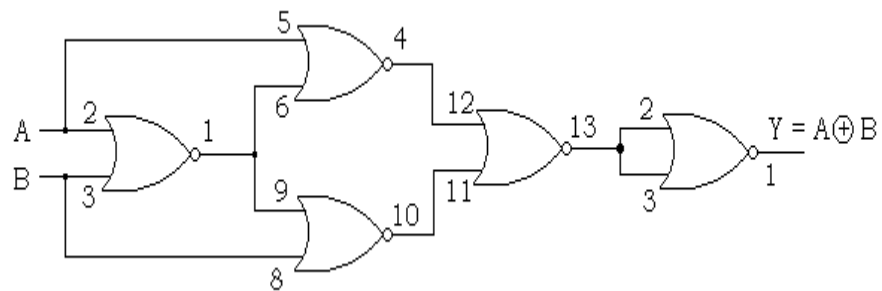
A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Realization of EX-NOR gate using only NOR gates
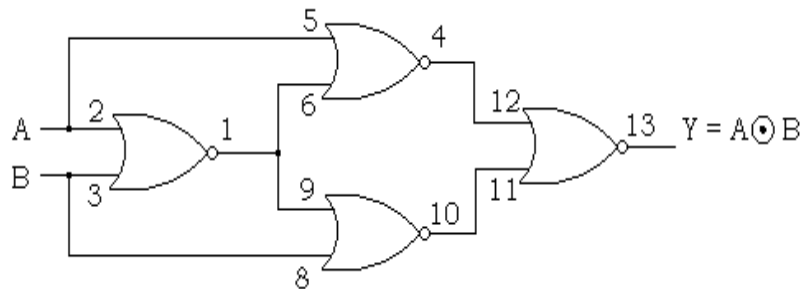
A	B	Y = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

EXPERIMENT: 3   **REALIZATION OF FLIPFLOPS**

Aim : To Construct different types of flip-flops and verify the truth tables.

Apparatus and Components :

S.No	Name	Quantity
1.	Digital trainer	1
2.	IC 7476	1
3.	IC 7400	1
4.	IC 7404	1

Procedure :

1. RS flip-flop is wired as shown in fig and input signals are fed from logic input switches and the output is monitored on the logic level output condition indicators and the truth table is verified.

2. JK flip-flop is wired as shown in fig and the input signals are fed from logic input switches and the output is monitored on the logic level output condition indicators and the truth table is verified.

3. Verify the truth tables of D flip flop and T flip flop in the same procedure.

Result :

Truth tables of RS, JK, D and T flip-flops are verified.

Circuit Diagram:

Realization of RS flip- flop using NAND gates
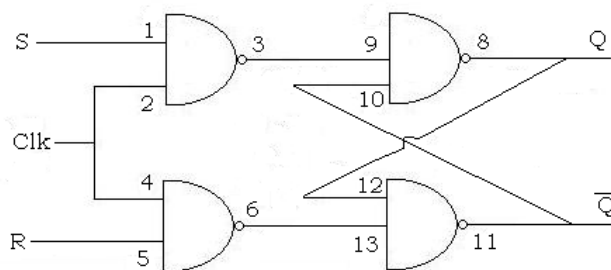
Clk	S	R	Theoretical Q_{n+1}	Practical Q_{n+1}
0	X	X	Q_n	
1	0	0	Q_n	
1	0	1	0	
1	1	0	1	
1	1	1	?	

JK flip- flop
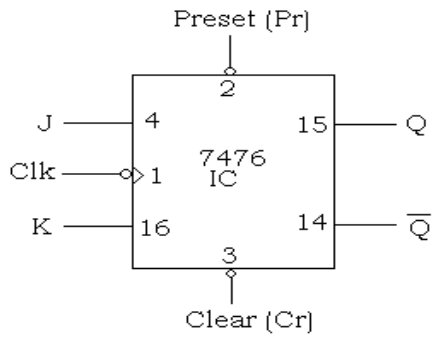
Clk	PR	CR	J	K	Theoretical	Practical
					Q_{n+1}	Q_{n+1}
0	0	1	X	X	1	
0	1	0	X	X	0	
↓	1	1	0	0	Q_n	
↓	1	1	0	1	0	
↓	1	1	1	0	1	
↓	1	1	1	1	$\overline{Q_n}$	

T flip- flop
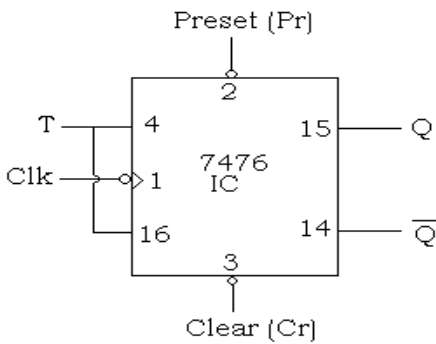
Clk	PR	CR	T	Theoretical	Practical
				Q_{n+1}	Q_{n+1}
0	0	1	X	1	
0	1	0	X	0	
↓	1	1	0	Q_n	
↓	1	1	1	$\overline{Q_n}$	

D flip- flop
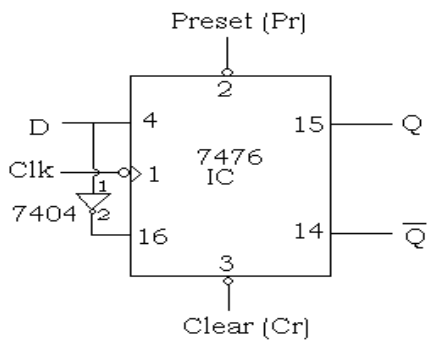
Clk	PR	CR	D	Theoretical	Practical
				Q_{n+1}	Q_{n+1}
0	0	1	X	1	
0	1	0	X	0	
↓	1	1	0	0	
↓	1	1	1	1	

EXPERIMENT: 4   **4-BIT RIPPLE COUNTER**

Aim : To design Asynchronous (Ripple) counter and verify the truth table.

Apparatus and Components :

S.No	Name	Quantity
1.	Digital trainer	1
2.	IC 7476	2

Procedure :

     1. Ripple counter circuit is connected as shown in the circuit diagram.

     2. 1Hz clock pulse is applied to the pin shown.

     3. The outputs $Q_oQ_1Q_2Q_3$ are observed and verify the truth table.

Result :

     Asynchronous counter is constructed and truth table is verified.

4 – bit Ripple Counter Truth Table.

Clock pulses	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Circuit Diagram:
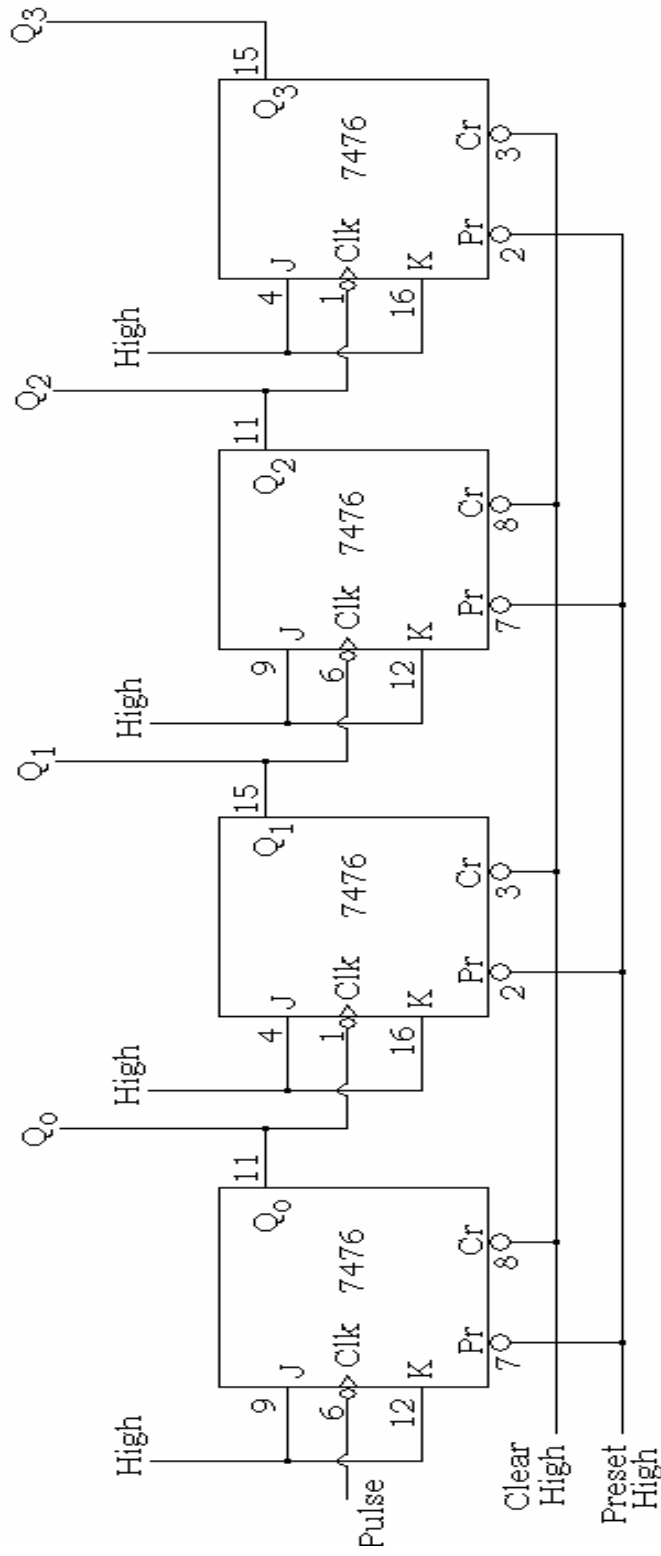
4 - bit Ripple Counter

Vcc = 5
Gnd = 13

EXPERIMENT: 5   <u>**4 - BIT SHIFT REGISTER**</u>

<u>**Aim**</u> **:** To design 4-bit shift register and verify the operation of serial loading
and parallel loading,

<u>**Apparatus and Components**</u> **:**

S.No	Name	Quantity
1.	Digital trainer	1
2.	IC 7474	2
3.	IC 7400	1

<u>**Procedure**</u> **:**

    1. Connect the circuit as shown in fig.

    2. For serial loading keep load low.

    3. First clear all the flip-flops by supplying clear = low

    4. First enter serial input one by one, through clock pulse; we will get
   Parallel output at $Q_3Q_2Q_1Q_o$. After applying 4 clock pulses we will
   get serial output.

    5. For parallel input keep load = high

    6. Directly apply parallel input to $P_{r3},P_{r2},P_{r1},P_{r0}$; we will get parallel
   output at $Q_3Q_2Q_1Q_o$. After applying 4 clock pulses we will get serial
   output.

<u>**Result**</u> **:** The operation of 4-bit shift register for serial loading and parallel loading is observed

4 – bit Shift Register Truth Table :

Serial Input → Serial Output / Parallel Output: The input is 1010.

Load	Clk	Clear	Serial i/p	Parallel output				Serial output Qo
				Q3	**Q2**	**Q1**	**Q0**	
0	X	0	X	0	0	0	0	0
0	1	1	0	**0**	0	0	0	0
0	2	1	1	**1**	**0**	0	0	0
0	3	1	0	**0**	**1**	**0**	0	0
0	4	1	1	**1**	**0**	**1**	**0**	**0**
0	5	1	0	0	1	0	1	**1**
0	6	1	0	0	0	1	0	**0**
0	7	1	0	0	0	0	1	**1**

Parallel input → Parallel Output / Serial Output   : The input is 1010.

Load	Clk	Clear	Parallel i/p				Parallel output				Serial output Qo
			Pr3	**Pr2**	**Pr1**	**Pr0**	**Q3**	**Q2**	**Q1**	**Q0**	
0	X	0	X	X	X	X	0	0	0	0	0
1	X	1	1	0	1	0	**1**	**0**	**1**	**0**	**0**
0	1	1	X	X	X	X	0	**1**	**0**	**1**	**1**
0	2	1	X	X	X	X	0	0	**1**	**0**	**0**
0	3	1	X	X	X	X	0	0	0	**1**	**1**

Circuit Diagram:
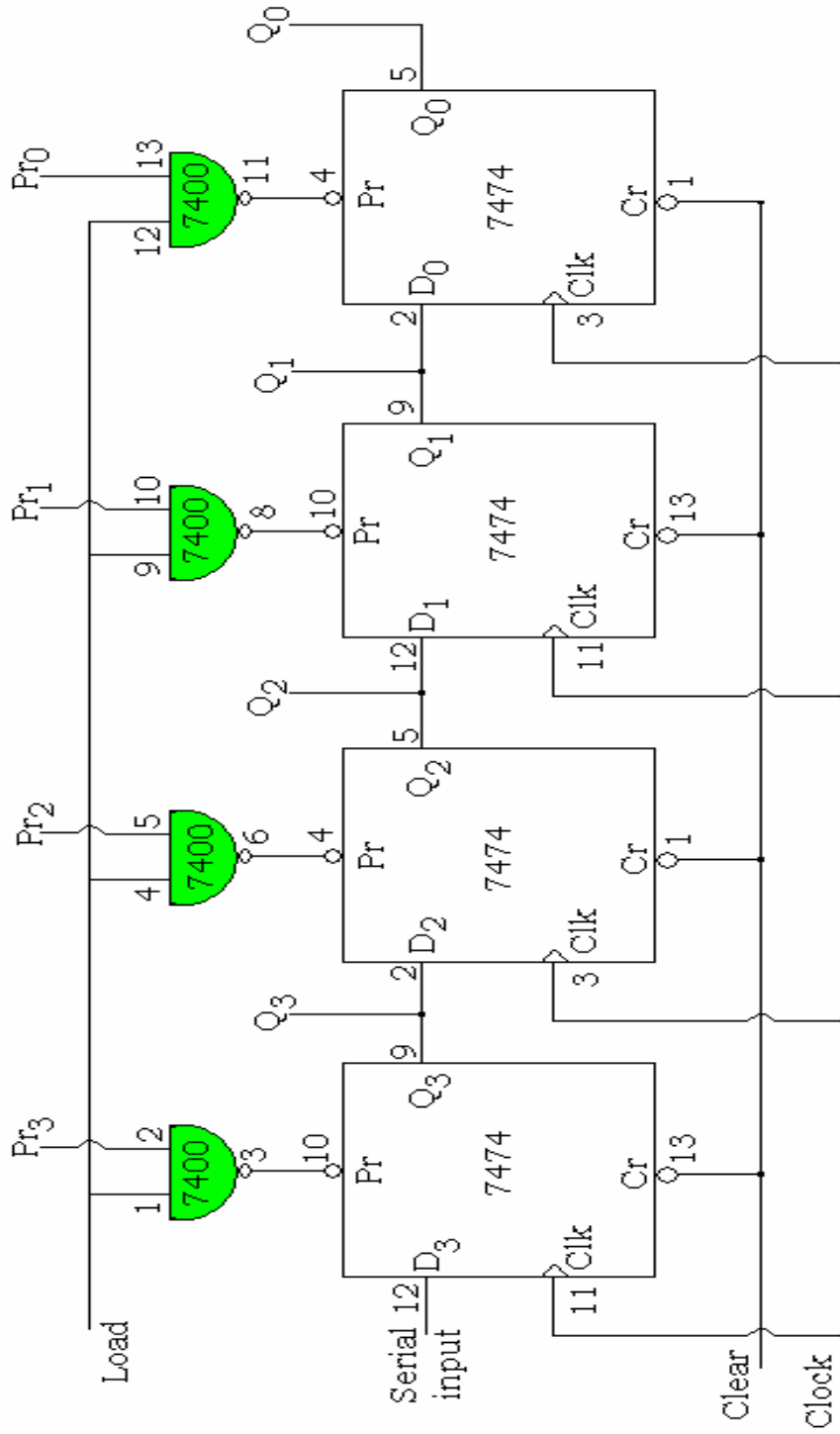
4-bit Shift Register

EXPERIMENT: 6　　　　**4-BIT & 8-BIT BINARY ADDERS**

Aim : To construct and evaluate 4 bit and 8 bit adders

Apparatus and Components :

S.No	Name	Quantity
1.	Digital Trainer	1
2.	IC 7483	2
3.	IC 7486	2
4.	IC 7404	1

Procedure :

Adders:

      1. The circuit is connected as shown in fig.

      2. Apply two 4 bit positive numbers A and B, observe the output.

      3. Verify the truth table.

      4. Repeat above steps for 8 bit adders also.

Result :

      The truth tables for 4 bit & 8 bit full adders are verified.

Truth Table :

4 – bit Binary Adder :

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	C_{in}	S_3	S_2	S_1	S_0	C_{out}

8 – bit Binary Adder :

A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0	Cin	S7	S6	S5	S4	S3	S2	S1	S0	Cout

Circuit Diagram:

4 - bit Adder

8 - bit Adder

EXPERIMENT: 7   **ADDITION OF TWO 8-BIT NUMBERS**

PROGRAM:

Write an assembly language program to add two binary numbers of 8-bit data stored in memory locations 8C40H and 8C41H and store the result in 8C42H and 8C43H.

PROBLEM ANALYSIS:

To perform addition in 8085 one of the data should be in accumulator and another data can be in any one of the general purpose register or in memory. After addition the sum will be in accumulator. The sum of two 8-bit data can be either 8-bit (sum only) or 9 bits (sum and carry). The accumulator can be accommodated only the sum and if there is a carry, the 8085 will indicate by setting carry flag. Hence one of the register is used to account for carry.

ALGORITHM:

1. Load the address of the data in H L reg. pair.
2. Clear C register.
3. Move the first data from memory to accumulator.
4. Increment the H L register pair (memory pointer).
5. Add the content as memory addressed by H L with accumulator.
6. Check for carry. If carry=1, go to step 7 or if carry =0, go to step 8.
7. Increment the C register.
8. Increment the H L pair and store the sum.
9. Increment the H L pair and store the carry.
10. Stop.

FLOWCHART:

start

load address of data
in HL Register pair

clear C Register

get the first data in
A Register

increment HL register pair

add the content of memory
to A Register

check whether cy = 1 — yes → increment C Register

No

increment HL pair and
store the sum in memory

increment HL pair and
store the carry in memory

stop
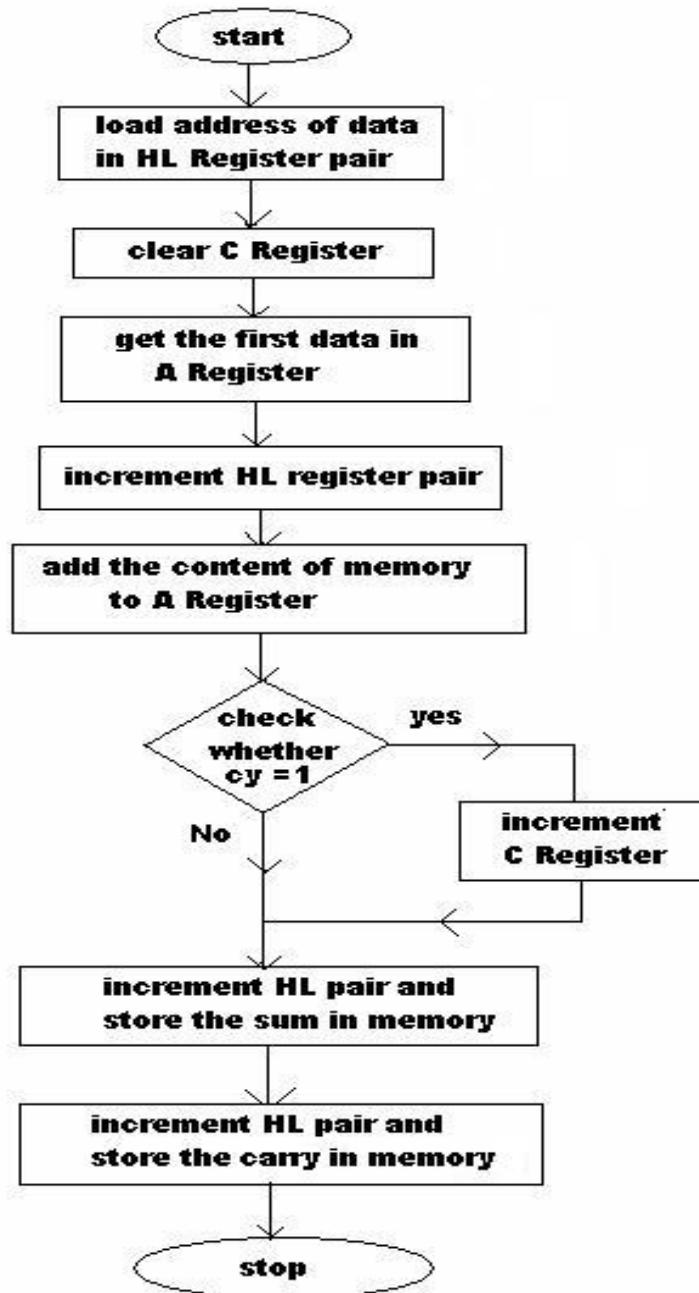
ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC	COMMENT
8C00H	21 40 8C		LXI H, 8C40H	// Set pointer for data.
8C03H	0E 00		MVI C,00H	// Clear C Reg. to account for carry
8C05H	7E		MOV A,M	// Get first data in A reg.
8C06H	23		INX H	// Increment the pointer for second data
8C07H	86		ADD M	// Add second data which is stored in memory to A. sum in A Reg.
8C08H	D2 0C 8C		JNC **L1**	// if CY=0, go to **L1**
8C0BH	0C		INR C	// if CY=1 increment register C
8C0CH	23	**L1:**	INX H	// increment the pointer to store the sum in Memory
8C0DH	77		MOV M,A	// Store the sum in memory.
8C0EH	23		INX H	// Increment the pointer to store the carry in Memory.
8C0FH	71		MOV M,C	// Store the carry in memory
8C10H	76		HLT	// Halt the program

OUTPUT:

8C40::02
8C41::03
8C42::05
8C43::00

EXPERIMENT: 8   <u>**SUBTRACTION OF TWO 8 - BIT NUMBERS**</u>

PROGRAM:

Write an assembly language program to subtract two numbers of 8-bit data stored in memory locations 8C40H and 8C41H. Store the magnitude of the result in 8C42H. If the result is positive store 00 in 8C43H or if the result is negative store 01in 8C43H.

PROGRAM ANALYSIS:

To perform subtraction in 8085 one of the data should be in accumulator and another data can be in anyone of the general purpose register or in memory. After subtraction the result will be in accumulator. The 8085 performs 2's complement subtraction and then complements the carry. Therefore if the result is negative then carry flag is set and accumulator will have 2's complement of the result. Hence one of the register is used to account for sign of the result. To get the magnitude of the result again take 2's complement of the result.

ALGORITHM:

1. Load the subtrahend (the data to be subtracted) from memory to accumulator and move it to B- register.
2. Load the minuend from memory to accumulator.
3. Clear C register to account for sign of the result.
4. Subtract the content of B-register from the content of the accumulator.
5. Check for carry . if carry =1 go to step 6 or if carry=0 , go to step 7.
6. Increment C register , complement the accumulator and add 01H
7. store the difference in memory.
8. Move the content of C register (sign bit) to accumulator and store in memory.
9. Stop.

FLOWCHART:
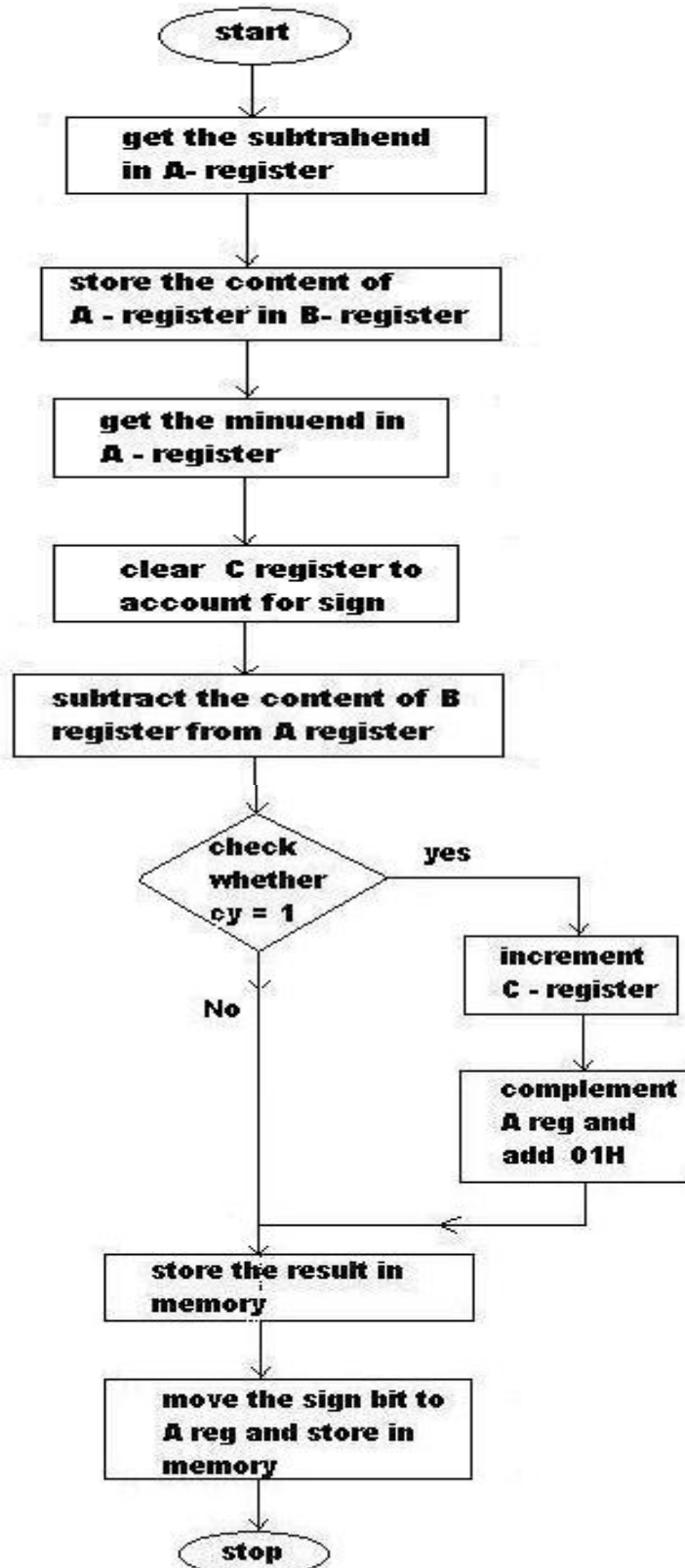
ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC	COMMENT
8C00H	3A 41 8C		LDA 8C41H	; get subtrahend from 8C41H and store in Acc
8C03H	47		MOV B,A	; move the content of the Acc. into B reg.
8C04H	3A 40 8C		LDA 8C40H	; get the minuend in A - reg
8C07H	0E 00		MVI C,00H	; clear C register , to account for sign.
8C09H	90		SUB B	; get the difference in A register
8C0AH	D2 11 8C		JNC **L1**	; if CY = 0 , then go to **L1**
8C0DH	0C		INR C	; if CY = 1 then increment C register.
8C0EH	2F		CMA	; get 2's complement of difference in A-reg
8C0FH	C6 01		ADI 01	; increment A register
8C11H	32 42 8C	**L1**	STA 8C42H	; store the result in memory
8C14H	79		MOV A,C	; move the barrow to accumulator
8C15H	32 43 8C		STA 8C43H	; store the sign bit in memory.
8C18H	76		HLT	; halt the program

OUTPUT:

8C40::02

8C41::03

8C42::01

8C43::01

EXPERIMENT: 9   MULTIPLICATION OF TWO 8-BIT NUMBERS

PROBLEM:

Write an assembly language program to multiply two numbers of 8–bit data stored in memory 8C4OH and 8C41H. Store the product in 8C42H and 8C43H.

PROBLEM ANALYSIS:

In this method multiplication is performed as repeated additions. The initial value of sum is assumed as zero. One of the data is used as count (N) . For number of additions to be performed. Another data is added to the sum N times where N is the count. The result of the product of two 8-bit data may be 16-bit data. Hence another register is used to account for over flow.

ALGORITHM:

1. Load the address of the first data in H L pair.
2. Clear C register for over flow.
3. Clear the accumulator.
4. Move the first data to B register.
5. Increment the pointer.
6. Move the second data to D register from memory.
7. Add the content of D-register to accumulator.
8. Check for carry. If CY=1 go to step 9 or if CY=0 go to step10.
9. Increment C register.
10. Decrement B register.
11. Check whether count has reached zero. If ZF=0 repeat steps 7 to 11. if ZF=1 got to next step.
12. Increment the pointer and store the LSB of the product in memory.
13. Increment the pointer and store the MSB of the product in memory.
14. Stop.

FLOWCHART:
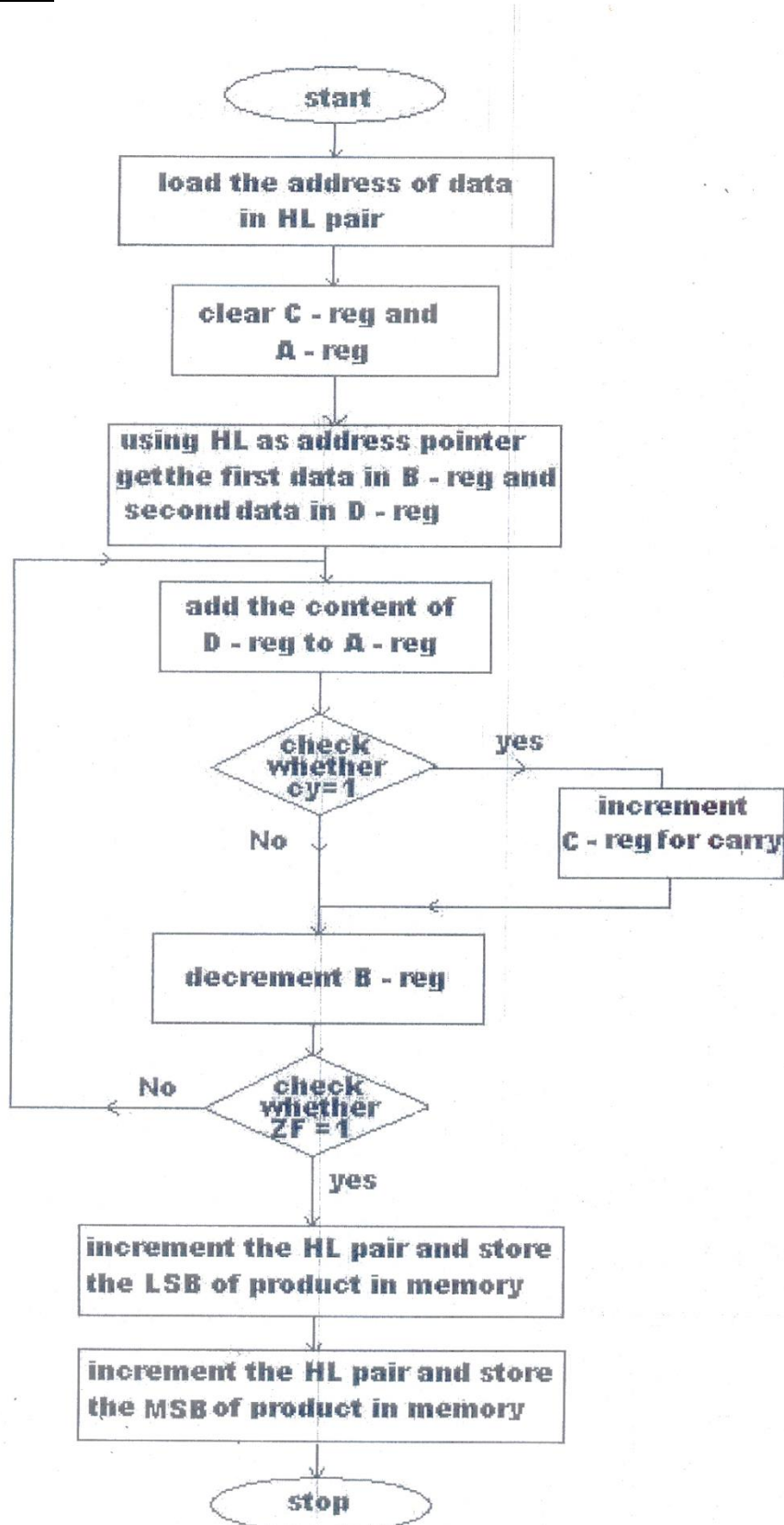
```
                    ┌─────────────┐
                    │    start    │
                    └──────┬──────┘
                           │
              ┌────────────▼────────────┐
              │  load the address of    │
              │    data in HL pair      │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │     clear C - reg and   │
              │         A - reg         │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │ using HL as address     │
              │ pointer get the first   │
              │ data in B - reg and     │
              │ second data in D - reg  │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │    add the content of   │
              │      D - reg to A - reg │
              └────────────┬────────────┘
                           │
                      ╱────▼────╲          yes    ┌──────────────────┐
                     ╱  check    ╲────────────────▶│   increment      │
                     ╲ whether    ╱                │ C - reg for carry│
                      ╲ cy=1     ╱                 └─────────┬────────┘
                       ╲───┬───╱                            │
                        No │                                │
              ┌────────────▼────────────┐◀──────────────────┘
              │     decrement B - reg   │
              └────────────┬────────────┘
                           │
              No      ╱────▼────╲
           ◀─────────╱  check    ╲
                     ╲ whether    ╱
                      ╲ ZF = 1   ╱
                       ╲───┬───╱
                        yes│
              ┌────────────▼────────────┐
              │ increment the HL pair   │
              │ and store the LSB of    │
              │ product in memory       │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │ increment the HL pair   │
              │ and store the MSB of    │
              │ product in memory       │
              └────────────┬────────────┘
                           │
                    ┌──────▼──────┐
                    │    stop     │
                    └─────────────┘
```

ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC	COMMENT
8COOH	21 40 8C		LXI H, 8C40H	; set pointer for data.
8CO3H	0E 00		MVI C, OOH	; clear C register to account for over flow
8CO5H	AF		XRA A	; clear accumulator (initial sum=0).
8CO6H	46		MOV B,M	; get first data in B register.
8C07H	23		INX H	; increment pointer.
8CO8H	56		MOV D,M	; get second data in D register.
8CO9H	82	**L1**	ADD D	; add the content D register to accumulator
8COAH	02 0E 8C		JNC **L2**	; if CY=0, go to L2
8CODH	0C		INR C	; if CY=1 increment C register
8COEH	05	**L2**	DCR B	; if CY=0 decrement B register
8COFH	C2 09 8C		JNZ **L1**	**;** repeat addition until ZF=1.
8C12H	23		INX H	; Increment HL Reg pair
8C13H	77		MOV M,A	; store LSB of product in memory.
8C14H	23		INX H	; Increment HL Reg pair
8C15H	71		MOV M,C	; store MSB of product in memory.
8C16H	76		HLT	; halt the program.

OUTPUT:

8C40::02

8C41::03

8C42::06

8C43::00

28

EXPERIMENT: 10   DIVISION OF TWO 8-BIT NUMBERS

PROBLEM:

Write an assembly language program to divide two numbers of 8 –bit data.

ALGORITHM:

1. Load Divisor and Dividend

2. Subtract divisor from dividend

3. Count the number of times of subtraction which equals the quotient

4. Stop subtraction when the dividend is less than the divisor .The dividend now

   becomes the remainder. Otherwise go to step 2.

5. Stop the program execution.

FLOWCHART:

```
                    ┌─────────────┐
                    │    START    │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │   B ← 00    │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │ [HL] ←4500  │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │   A ← M     │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │[HL] ← [HL]+1│
                    └──────┬──────┘
                           │◄──────────────────────────┐
                    ┌──────▼──────┐                     │
                    │  M ← A-M    │                     │
                    └──────┬──────┘                     │
                           │                            │
                    ┌──────▼──────┐                     │
                    │ [B] ← [B] +1│                     │
                    └──────┬──────┘                     │
                           │                            │
                    ┌──────▼──────┐    NO               │
                    ◄   IS A<0    ►─────────────────────┘
                    └──────┬──────┘
                           │ YES
                    ┌──────▼──────┐
                    │  A ← A+ M   │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │   B ← B-1   │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │[HL]←[HL]+1  │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │ [M] ← [A]   │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │[HL]←[HL]+1  │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │ [M] ← [B]   │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │    STOP     │  30
                    └─────────────┘
```
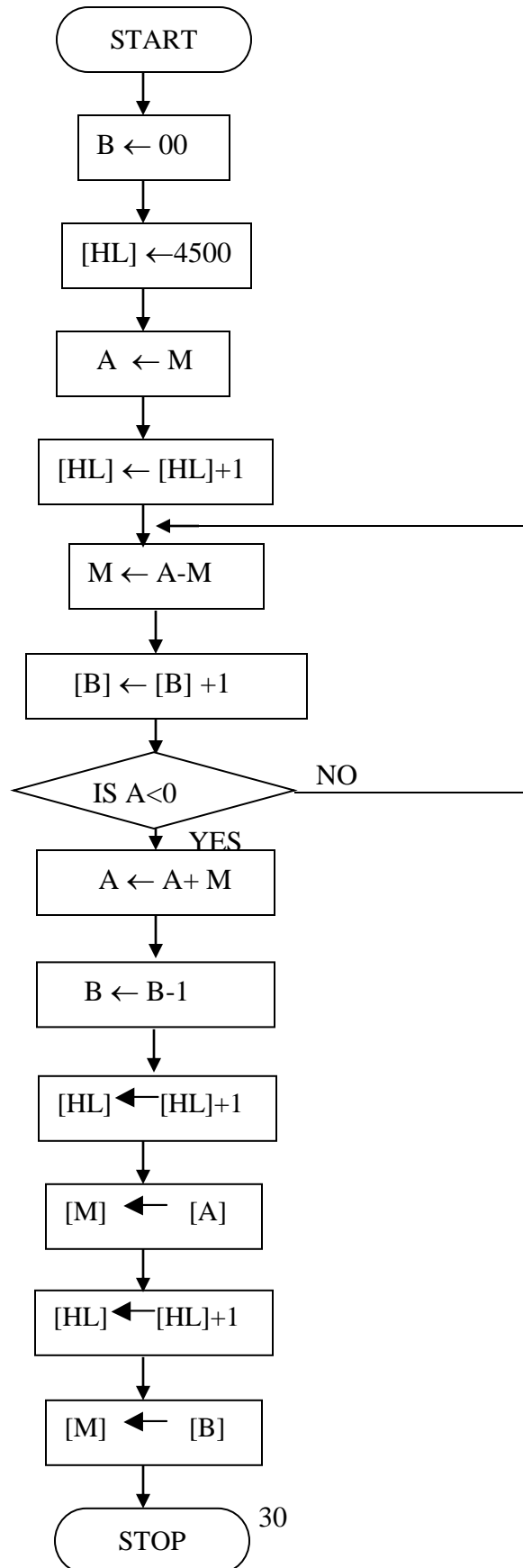
ASSEMBLY LANGUAGE PROGRAM:

ADDRESS	OPCODE	LABEL	MNEM ONICS	OPERA ND	COMMENTS
4100	06		MVI	B,00	Clear B reg for quotient
4101	00				
4102	21		LXI	H,4500	Initialize HL reg. to
4103	00				4500H
4104	45				
4105	7E		MOV	A,M	Transfer dividend to acc.
4106	23		INX	H	Increment HL reg. to point next mem. Location.
4107	96	LOOP	SUB	M	Subtract divisor from dividend
4108	04		INR	B	Increment B reg
4109	D2		JNC	LOOP	Jump to LOOP if result does
410A	07				not yield borrow
410B	41				
410C	86		ADD	M	Add divisor to acc.
410D	05		DCR	B	Decrement B reg
410E	23		INX	H	Increment HL reg. to point next mem. Location.
410F	77		MOV	M,A	Transfer the remainder from acc. to memory.
4110	23		INX	H	Increment HL reg. to point next mem. Location.
4111	70		MOV	M,B	Transfer the quotient from B reg. to memory.
4112	76		HLT		Stop the program

OUTPUT:

4500::06

4501::03

4502::00

4503::02

31

EXPERIMENT: 11   16-BIT ADDITION

PROGRAM:

Write an assembly language program to add two numbers of 16-bit data stored in memory 8C40H, 8C41H and 8C42H, 8C43H. The data are stored such that LSB first and then MSB and store the result from 8C44H to 8C46H

PROBLEM ANALYSIS:

The 16-bit addition can be performed in 8085 microprocessor either in terms of 8-bit addition or by using DAD instruction. In addition using DAD instruction, one of the data should be in H L pair and another data can be another register pair. After addition the sum will be in H L register pair. If there is a carry in addition then that is indicated by setting carry flag. Hence one the register is used to account for carry.

ALGORITHM:

    1 load the first data in H L register pair.

    2. Move the first data to D E register pair.

    3. Load the second data in H L register pair.

    4. Clear A register for carry.

    5. Add the content of D E pair to H L pair.

    6. Check for carry. If carry =1, go to step 7 or if carry=0 go to step 8.

    7. Increment carry register (A) to account for carry.

    8. Store the sum and carry in memory.

    9. Stop.

FLOWCHART:

```
                    ( start )
                        |
                        v
        +-------------------------------+
        |   load the first data in      |
        |   HL pair and save in DE pair |
        +-------------------------------+
                        |
                        v
        +-------------------------------+
        |     get second data           |
        |       in HL pair              |
        +-------------------------------+
                        |
                        v
        +-------------------------------+
        |     clear A - register        |
        +-------------------------------+
                        |
                        v
        +-------------------------------+
        |    add the content of         |
        |    DE pair to HL pair         |
        +-------------------------------+
                        |
                        v
                   /  check  \        yes      +------------------+
                  <  whether  > ------------->  |   increment      |
                   \  cy = 1 /                  |   A - register   |
                      \   /                     +------------------+
                   No   |                              |
                        v                              |
                        +<-----------------------------+
                        |
                        v
        +-------------------------------+
        |      store the sum            |
        |    (HL pair) in memory        |
        +-------------------------------+
                        |
                        v
        +-------------------------------+
        |   store the carry (A-reg)     |
        |        in memory              |
        +-------------------------------+
                        |
                        v
                    ( stop )
```
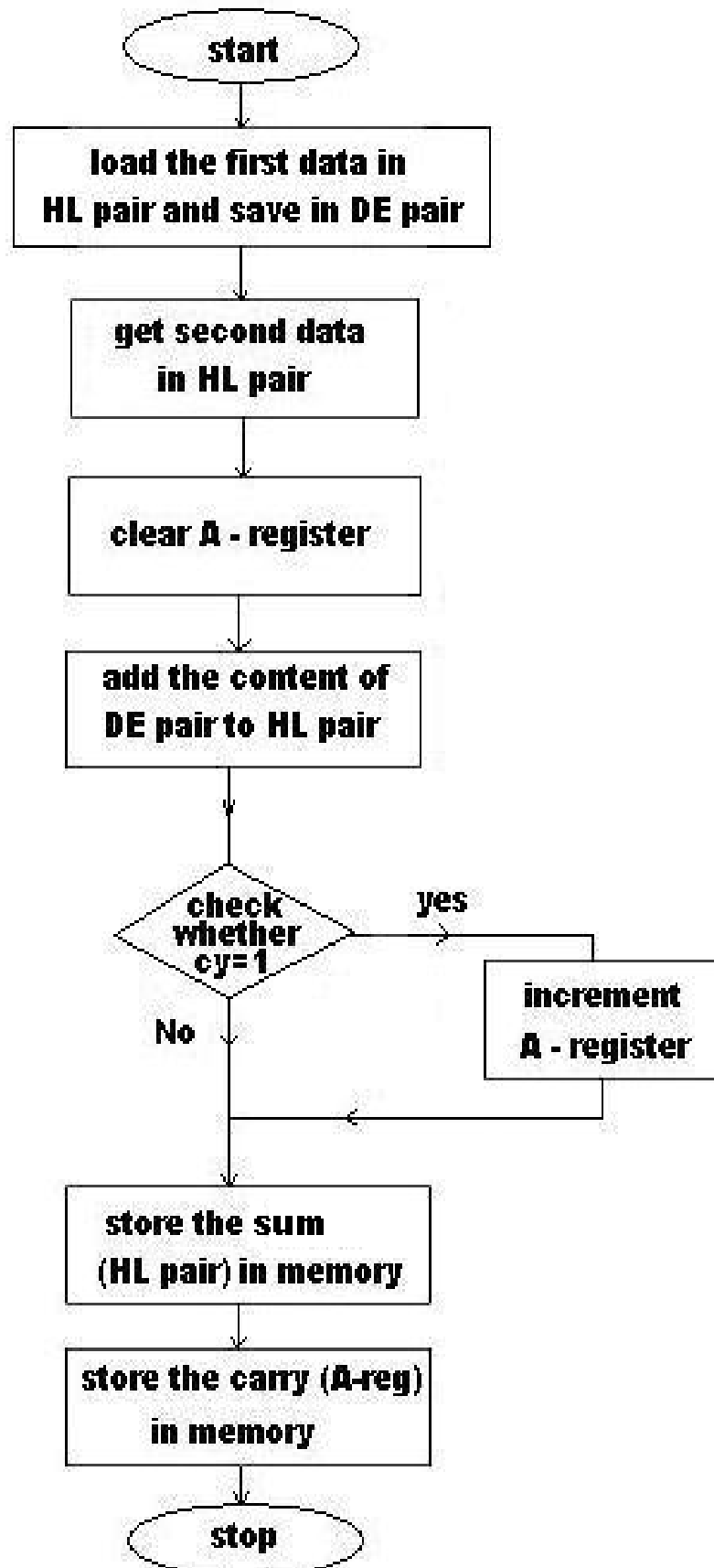
ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC		COMMENT
8C00H	2A 40 8C		LHLD	8C40H	; get first data in HL reg. pair.
8C03H	EB		XCHG		; store first data in DE reg. pair
8C04H	2A 42 8C		LHLD	8C42H	; get second data in HL reg. pair
8C07H	AF		XRA	A	; clear A register for carry
8C08H	19		DAD	D	; get the sum in HL pair.
8C09H	D2 0D 8C		JNC	**L1**	; if CY=0 go to L1
8C0CH	3C		INR	A	; if CY=1, increment A reg.
8C0DH	22 44 8C	**L1**	SHLD	8C44H	; store the sum in memory.
8C10H	32 46 8C		STA	8C46H	; store the carry in memory.
8C13H	76		HLT		; halt the program

OUTPUT:

8C40::02
8C41::03
8C42::05
8C43::00
8C44::07
8C45::03
8C46::00

EXPERIMENT: 12   16-BIT SUBTRACTION

PROGRAM:

Write an assembly language program to subtract two numbers of 16-bit data stored in memory from 8C40H to 8C43H. The data are stored such that LSB first and then MSB. Store the result in 8C44H and 8C45H.

PROBLEM ANALYSIS:

The 16-bit subtraction is performed in terms of 8-bit subtraction. First LSB's of the data are subtracted and the result is stored in memory. Then MSB's of the data are subtracted along with borrow in the previous subtraction and the result is stored in memory.

ALGORITHM:

1. Load the low byte of subtrahend in accumulator from memory and move is to B-register.
2. Load the low byte of minuend in accumulator from memory.
3. Subtract the content of B-register from the content of accumulator.
4. Store the low byte of result in memory.
5. Load the high byte of subtrahend in accumulator from memory and move it to B-register.
6. Load the high byte of minuend in accumulator from memory.
7. Subtract the content of B-register and the carry from the content of accumulator.
8. Store high byte of result in memory.
9. Stop the program.

FLOWCHART:

start

get the low byte ofsubtrahend in A and move to B

get the low byte of minuend in A

perform subtraction of low bytes and store the result in memory

get the high byte of subtrahend in A and move to B

get the high byte of minuend in A

subtract the content of B register and carry from A register

store the result in memory

stop
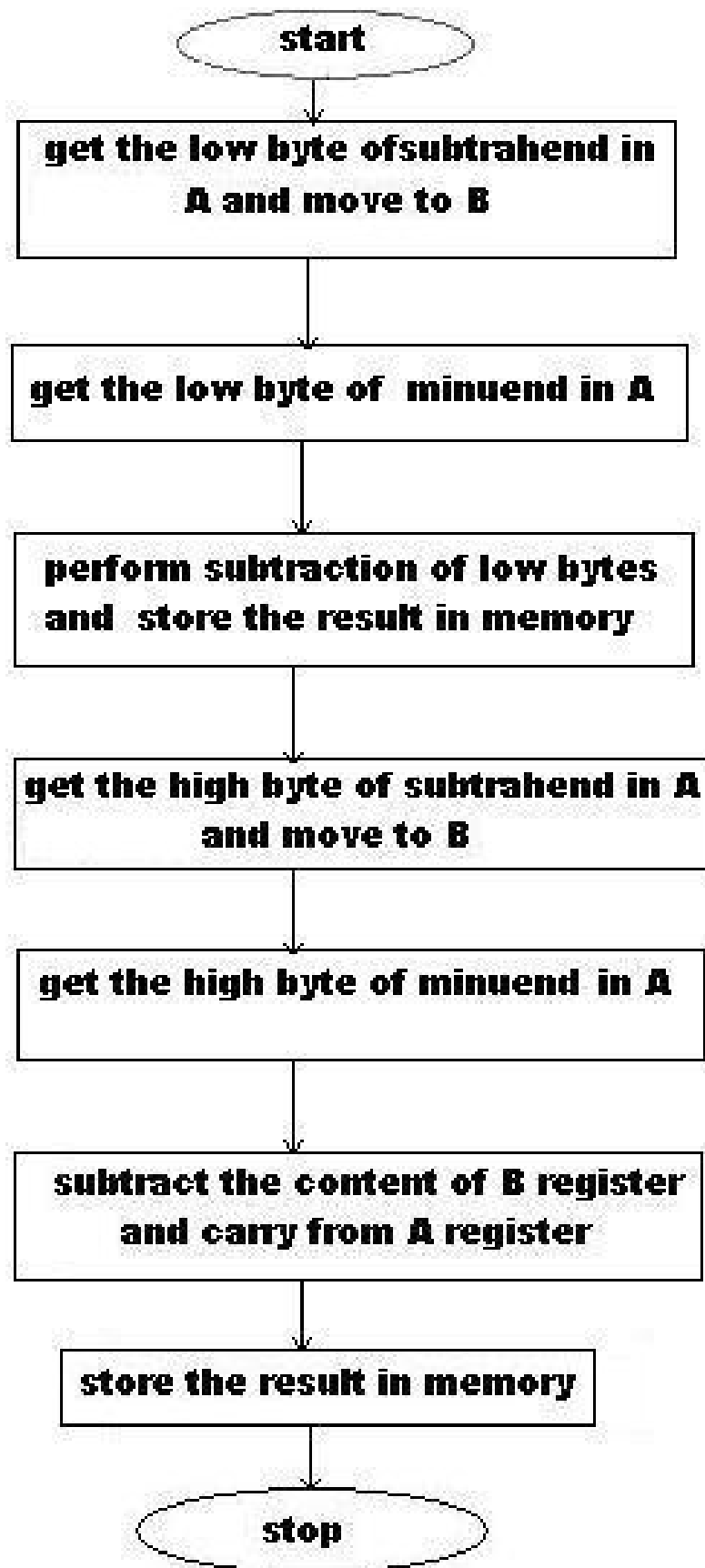
ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	MNEMONIC	COMMENT
8C00H	3A 42 8C	LDA 8C42H	; get the LSB of subtrahend from 8C42H and store in Acc
8C03H	47	MOV B, A	; Move the LSB of subtrahend to B-register
8C04H	3A 40 8C	LDA 8C40H	; get the LSB of minuend in A-register
8C07H	90	SUB B	; get the difference of LSB's in A-register
8C08H	32 44 8C	STA 8C44H	; store the result in memory.
8C0BH	3A 43 8C	LDA 8C43H	; get the MSB of subtrahend from 8C43H and store in Acc
8C0EH	47	MOV B, A	; Move the MSB of subtrahend to B-register.
8C0FH	3A 41 8C	LDA 8C41H	; get the MSB of minuend in A-register.
8C12H	98	SBB B	; get the difference of MSB's in A-register.
8C13H	32 45 8C	STA 8C45H	; store the result.
8C16H	76	HLT	; halt the program

OUTPUT:

8C40::05

8C41::03

8C42::02

8C43::01

8C44::03

8C45::02

8C46::00

EXPERIMENT: 13   **TWO DIGIT BCD ADDITION**

PROGRAM:

Write an assembly language program to add two numbers of two digit (single precession) BCD data stored memory locations 8C40H and 8C41H. Store the result in 8C42H and 8C43H.

PROBLEM ANALYSIS:

The 8085 microprocessor will perform only binary addition. Hence for BCD addition, the binary addition of BCD data is performed and then the sum is corrected to get result in BCD. After binary addition the following correction should be made to get the result in BCD.

1.   if the sum of lower nibble exceeds 9 or if there is an auxiliary carry then 06 is added to lower nibble.
2.   if the sum of upper nibble exceeds 9 or if there is carry then 06 is added to upper nibble.

The above correction is taken care by DAA instruction. Therefore after binary addition execute DAA instruction to do the above correction in the sum.

ALGORITHM:

1. Load the first data in accumulator and move it to B-register.
2. Load the second data in accumulator.
3. Clear the C register for storing carry.
4. Add the content of B-register to accumulator.
5. Execute DAA instruction.
6. Check for carry. If carry=1, go to step 7 or if carry=0, go to step 8.
7. Increment C register to account for carry.
8. Store the sum in memory.
9. Move the carry (content of C register) to accumulator and store in memory.

10. Stop.

FLOWCHART:
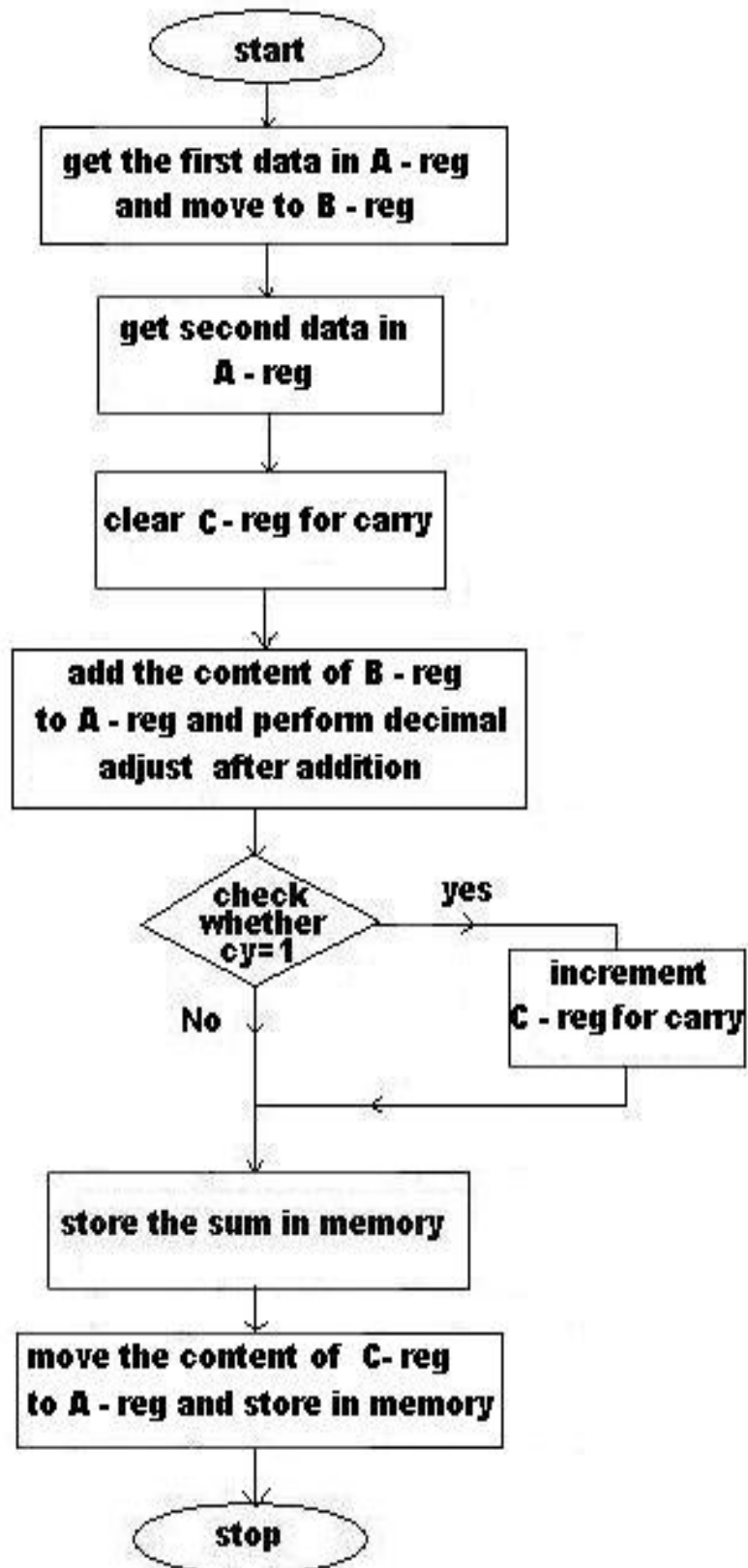
start

get the first data in A - reg
and move to B - reg

get second data in
A - reg

clear C - reg for carry

add the content of B - reg
to A - reg and perform decimal
adjust after addition

check
whether
cy=1

yes

No

increment
C - reg for carry

store the sum in memory

move the content of C- reg
to A - reg and store in memory

stop

ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC		COMMENT
8C00H	3A 40 8C		LDA	8C40H	; get first data in accumulator.
8C03H	47		MOV	B, A	; transfer accumulator data to B-register.
8C04H	3A 41 8C		LDA	8C41H	; get second data in A-register.
8C07H	0E 00		MVI	C, 00H	; clear C register for accounting carry.
8C09H	80		ADD	B	; add the content of B-register to A-register.
8C0AH	27		DAA		; get the sum of BCD data in A- reg.
8C0BH	D2 0E 8C		JNC	**L1**	; if CY=0, go to **L1.**
8COEH	0C		INR	C	; if CY=1, increment C- reg.
8C0FH	32 42 8C	**L1**	STA	8C42H	; store the sum in memory.
8C12H	79		MOV	A, C	; move the carry to A- reg.
8C13H	32 43 8C		STA	8C43H	; store the carry in memory.
8C16H	76		HLT		; halt the program.

OUTPUT:

8C40::80

8C41::80

8C42::60

8C43::01

EXPERIMENT 14   TWO DIGIT BCD SUBTRACTION

PROGRAM:

Write an assembly language program to subtract BCD numbers of 2 digit BCD data stored in memory 8C40H and 8C41H. store the result in 8C42H.

PROBLEM ANALYSIS:

The 8085 microprocessor will perform only binary subtraction. Hence for BCD subtraction 10's complement subtraction is performed. First the 10's complement of the subtrahend is obtained and then added to minuend. The DAA instruction is executed to get the result in BCD.

ALGORITHM:

1. Load the subtrahend in A-register and move to B-register.
2. Move 99 to A-register and subtract the content of B-register from A-register.
3. Increment the A-register.
4. Move the content of A-register to B-register.
5. Load the minuend in A-register.
6. Add the content of B-register to a A-register.
7. Execute DAA instruction.
8. Store the result in memory.
9. Stop.

FLOWCHART:
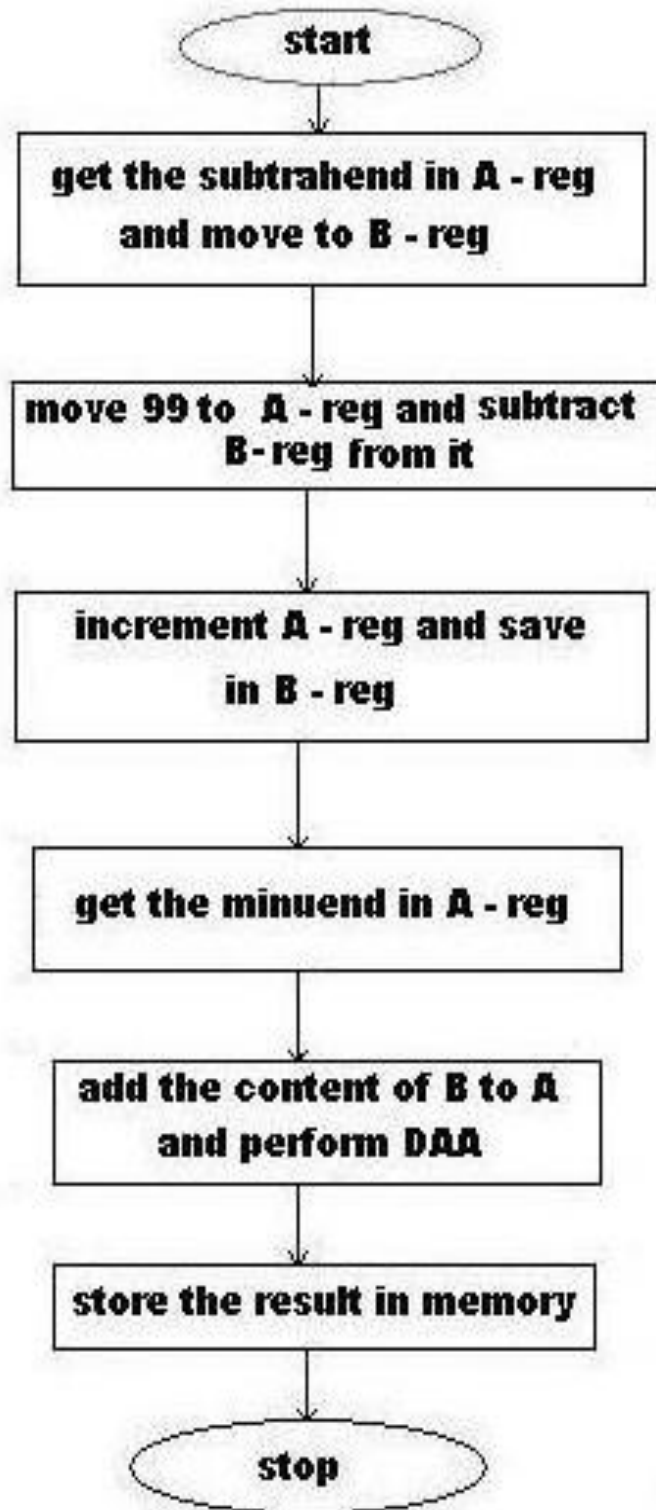
```
                    ┌──────────────┐
                    │    start     │
                    └──────┬───────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │    get the subtrahend in A - reg      │
        │         and move to B - reg           │
        └──────────────────┬───────────────────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │   move 99 to  A - reg and subtract    │
        │          B - reg from it              │
        └──────────────────┬───────────────────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │    increment A - reg and save         │
        │            in B - reg                 │
        └──────────────────┬───────────────────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │      get the minuend in A - reg       │
        └──────────────────┬───────────────────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │     add the content of B to A         │
        │          and perform DAA              │
        └──────────────────┬───────────────────┘
                           │
                           ▼
        ┌──────────────────────────────────────┐
        │     store the result in memory        │
        └──────────────────┬───────────────────┘
                           │
                           ▼
                    ┌──────────────┐
                    │     stop     │
                    └──────────────┘
```

ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE	LABEL	MNEMONIC	COMMENT
8COOH	3A 41 8C		LDA 8C41H	; get the subtrahend in to accumulator.
8CO3H	47		MOV B,A	; move the data into B-register from A-reg.
8CO4H	3E 99		MVI A,99	; move the 99 to A-reg.
8C06H	90		SUB B	; subtract the subtrahend from 99.
8C07H	3C		INR A	; 10's complement of subtrahend.
8C08H	47		MOV B,A	; store the 10's complement of subtrahend in B
8CO9H	3A 40 8C		LDA 8C40H	; get the minuend in A-register
8COCH	80		ADD B	; Get the BCD sum of minuend and 10'complemnt of subtrahend.
8CODH	27		DAA	; the sum is the difference between given BCD data.
8COEH	32 42 8C		STA 8C42H	; store the result in memory.
8C11H	76		HLT	; halt the program.

OUTPUT:

8C40::80

8C41::60

8C42::20

8C43::00

EXPERIMENT 15   **SORTING OF DATA IN ASCENDING ORDER AND**
FINDING LARGEST NUMBER IN THE ARRAY

PROGRAM:

Write an assembly language program to sort an array of data in ascending order and find the largest number and display it in the data field. The array is stored in memory starting from 8C40H. The first element of the array gives the count value for the number of elements in the array.
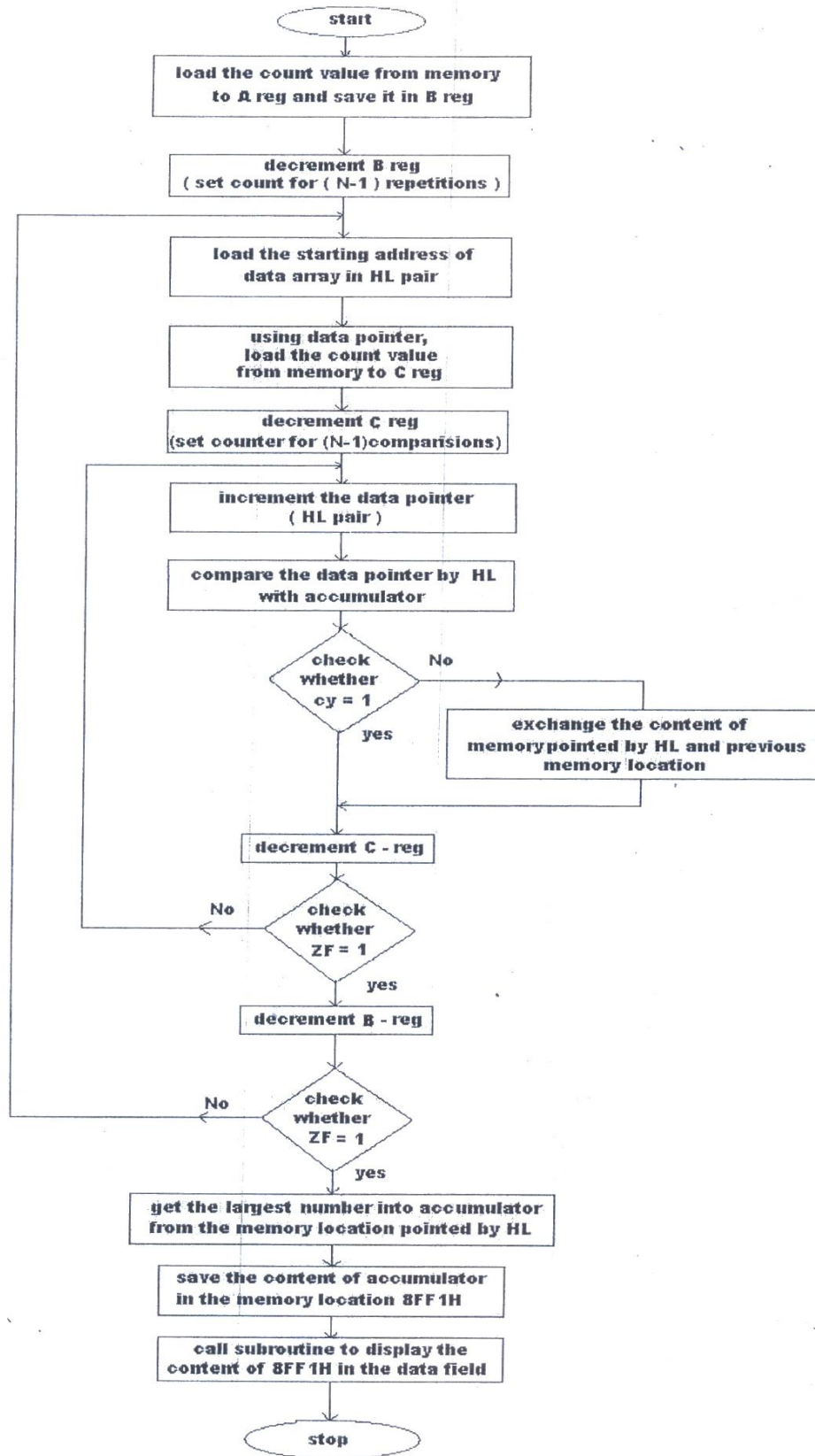
PROBLEM ANALYSIS:

The algorithm for bubble sorting is given below. In bubble sorting of N-data,

(N-1) comparisons are carried by taking two consecutive data at a time. After each comparison, the data are rearranged such that smallest among the two is in first memory location and largest in the next memory location. When we perform (N-1) comparisons as mentioned above, for (N-1) times then the array consisting of N-data will be sorted in the ascending order.

ALGORITHM:

1.   Load the count value from memory to A-reg. and save it in B-reg.
2.   Decrement B-reg . (B is a count for N-1 repetitions)
3.   Set H L pair as data address pointer.
4.   Set C-register as counter for (N-1) comparisons.
5.   Load a data of the array in accumulator using the data address pointer.
6.   Increment the H L pair (data address pointer).
7.   Compare the data pointed by H L with accumulator.
8.   if carry flag is set (if the content of the accumulator is smaller than memory) then go to step 10, otherwise go to next step.
9.   Exchange the content of memory pointed by H L and the accumulator.
10.   Decrement C-register. if zero flag is reset go to the step 6 otherwise go to next step.
11.   Decrement B-register. If zero flag is reset go to step 3 otherwise go to next step.
12.   Load the largest value from memory into accumulator.
13.   Store the content of accumulator in memory location 8FF1H.
14.   Call subroutine to display the content of memory location 8FF1H into the data field.

15.   Stop.

FLOWCHART:

start

load the count value from memory to A reg and save it in B reg

decrement B reg
(set count for (N-1) repetitions)

load the starting address of data array in HL pair

using data pointer, load the count value from memory to C reg

decrement C reg
(set counter for (N-1)comparisions)

increment the data pointer
(HL pair)

compare the data pointer by HL with accumulator

check whether cy = 1

No

yes

exchange the content of memory pointed by HL and previous memory location

decrement C - reg

check whether ZF = 1

No

yes

decrement B - reg

check whether ZF = 1

No

yes

get the largest number into accumulator from the memory location pointed by HL

save the content of accumulator in the memory location 8FF1H

call subroutine to display the content of 8FF1H in the data field

stop

ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE			LABEL	MNEMONIC		COMMENT
8CO0H	3A	40	8C		LDA	8C4OH	; load the count value in A-reg.
8CO3H	47				MOV	B,A	; set counter for (N-1)repetitions of
8CO4H	05				DCR	B	N-1 comparisons.
8CO5H	21	40	8C	**L2**	LXI	H,8C4OH	; set pointer for array.
8CO8H	4E				MOV	C,M	; set counter for (N-1) comparisons.
8CO9H	OD				DCR	C	;
8COAH	23				INX	H	; increment pointer
8COBH	7E			**L1**	MOV	A,M	; get one data of array in A-reg.
8COCH	23				INX	H	; increment pointer.
8CODH	BE				CMP	M	; compare next data with A-reg.
8COEH	DA	16	8C		JC	**L3**	; if content of A is less than memory then go to **L3**
8C11H	56				MOV	D,M	; if the content of A is greater than
8C12H	77				MOV	M,A	the content of memory then exchange
8C13H	2B				DCX	H	the content of memory pointed by H L
8C14H	72				MOV	M,D	and previous location.
8C15H	23				INX	H	;
8C16H	OD			**L3**	DCR	C	; decrement C-register.
8C17H	C2	0B	8C		JNZ	**L1**	; repeat comparisons until C reg. count is zero.
8C1AH	05				DCR	B	; decrement B-register.
8C1BH	C2	05	8C		JNZ	**L2**	; repeat until B count is zero.
8C1EH	7E				MOV	A,M	; get the largest number into accumulator.
8C1FH	32	F1	8F		STA	8FF1H	; store the content of accumulator in memory location 8FF1H.
8C22H	CD	4C	04		CALL	044CH	; call subroutine to display the content of the memory location 8FF1H in data field.
8C25H	76				HLT		; halt the program.

OUTPUT:

8C40::03 8C43::03

8C41::01 8FF1::03

8C42::02

EXPERIMENT 16   <u>**SORTING OF DATA IN DESCENDING ORDER**</u>
<u>**AND FINDING SMALLEST NUMBER IN THE ARRAY**</u>

<u>PROGRAM:</u>

Write an assembly language program to sort an array of data in descending order and find the smallest number and display it in the data field. The array is stored in memory starting from 8C40H. The first element of the array gives the count value for the number of elements in the array.

<u>PROBLEM ANALYSIS:</u>

The algorithm for bubble sorting is given below. In bubble sorting of N-data , (N-1) comparisons are carried by taking two consecutive data at a time. After each comparison, the data are rearranged such that largest among the two is in first memory location and smallest in the next memory location. When we perform (N-1) comparisons as mentioned above, for N times then the array consisting of N-data will be sorted in the descending order.
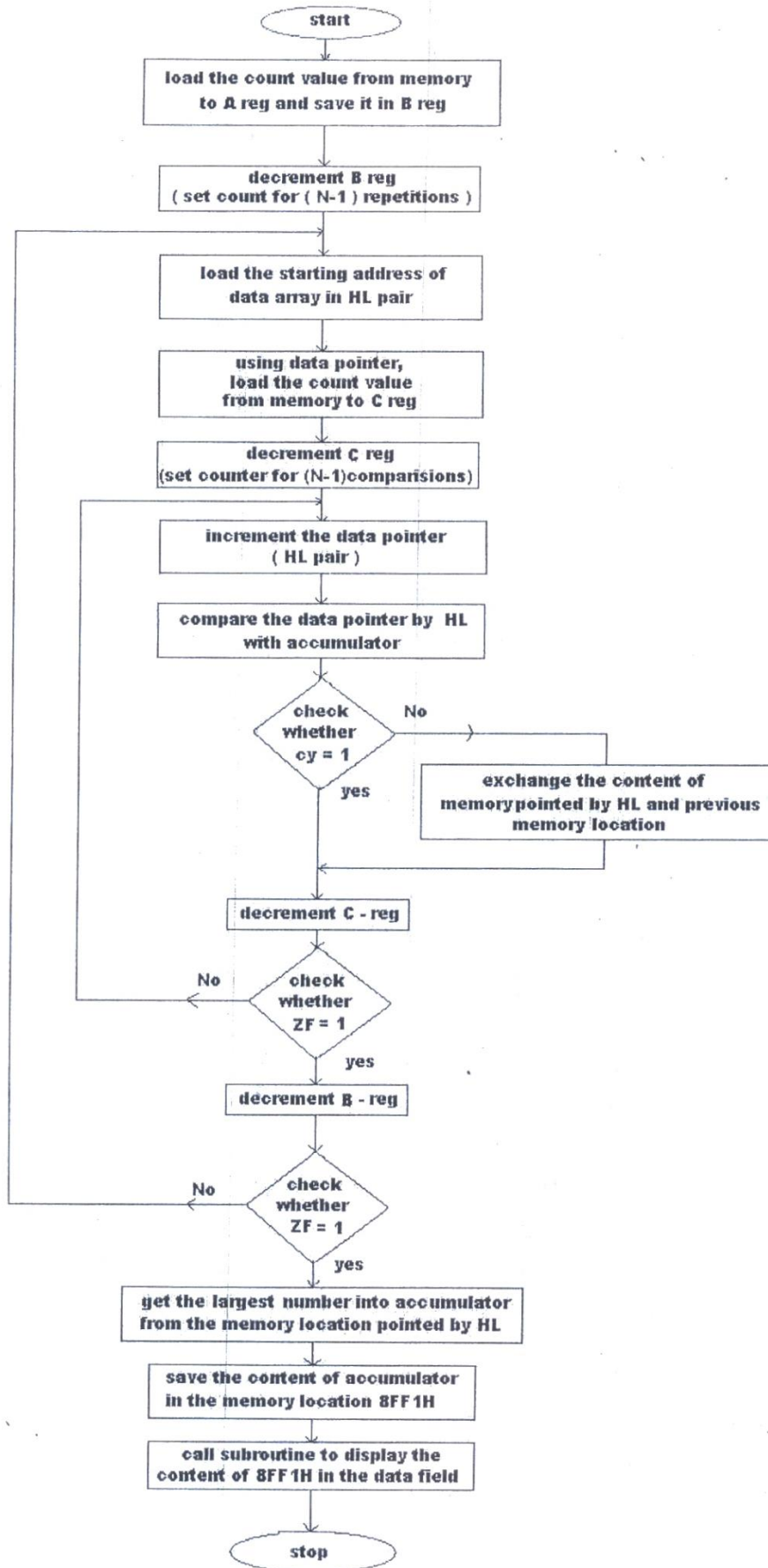
<u>ALGORITHM:</u>

1. Load the count value from memory to A-reg. and save it in B-reg.
2. Decrement B-reg (B is a count for N-1 repetitions).
3. Set H L pair as data address pointer.
4. Set C-register as counter for (N-1) comparisons.
5. Load a data of the array in accumulator using the data address pointer.
6. Increment the H L pair (data address pointer).
7. Compare the data pointed by H L with accumulator.
8. If carry flag is reset (if the content of the accumulator is larger than memory) then go to step 10, otherwise go to next step.
9. Exchange the content of memory pointed by H L and the accumulator.
10. Decrement C-register. if zero flag is reset go to the step 6 otherwise go to next step.
11. Decrement B-register. If zero flag is reset go to step 3 otherwise go to next   step.
12. Load the smallest value from memory into accumulator.
13. Store the content of accumulator in memory location 8FF1H.
14. Call subroutine to display the content of memory location 8FF1H into the

49

data field.

15. Stop.

FLOWCHART:

start

load the count value from memory
to A reg and save it in B reg

decrement B reg
(set count for (N-1) repetitions)

load the starting address of
data array in HL pair

using data pointer,
load the count value
from memory to C reg

decrement C reg
(set counter for (N-1)comparisons)

increment the data pointer
(HL pair)

compare the data pointer by HL
with accumulator

check
whether
cy = 1

No → exchange the content of
memory pointed by HL and previous
memory location

yes

decrement C - reg

check
whether
ZF = 1

No ←

yes

decrement B - reg

check
whether
ZF = 1

No ←

yes

get the largest number into accumulator
from the memory location pointed by HL

save the content of accumulator
in the memory location 8FF1H

call subroutine to display the
content of 8FF1H in the data field

stop

ASSEMBLY LANGUAGE PROGRAM:

MEMORY LOCATION	MACHINE CODE			LABEL	MNEMONIC		COMMENT
8CO0H	3A	40	8C		LDA	8C4OH	; load the count value in A-reg.
8CO3H	47				MOV	B,A	; set counter for (N-1) repetitions of N-1 comparisons.
8CO4H	05				DCR	B	
8CO5H	21	40	8C	**L2**	LXI	H,8C4OH	; set pointer for array.
8CO8H	4E				MOV	C,M	; set counter for (N-1) comparisons.
8CO9H	OD				DCR	C	;
8COAH	23				INX	H	; increment pointer
8COBH	7E			**L1**	MOV	A,M	; get one data of array in A-reg.
8COCH	23				INX	H	; increment pointer.
8CODH	BE				CMP	M	; compare next data with A-reg.
8COEH	DA	16	8C		JNC	**L3**	; if content of A is less than memory then go to **L3**
8C11H	56				MOV	D,M	; if the content of A is greater than the content of memory then exchange the content of memory pointed by H L and previous location.
8C12H	77				MOV	M, A	
8C13H	2B				DCX	H	
8C14H	72				MOV	M, D	
8C15H	23				INX	H	
8C16H	OD			**L3**	DCR	C	; decrement C-register.
8C17H	C2	0B	8C		JNZ	**L1**	; repeat comparisons until C reg. count is zero.
8C1AH	05				DCR	B	; decrement B -register.
8C1BH	C2	05	8C		JNZ	**L2**	; repeat until B count is zero.
8C1EH	7E				MOV	A,M	; get the smallest number into accumulator.
8C1FH	32	F1	8F		STA	8FF1H	; store the content of accumulator in memory location 8FF1H.
8C22H	CD	4C	04		CALL	044CH	; call subroutine to display the content of the memory location 8FF1H in data field.
8C25H	76				HLT		; halt the program

OUTPUT: 8C40::03 8C41::03 8C42::02 8C43:01 8FF1:01

EXPERIMENT 17   <u>**DIGITAL TO ANALOG CONVERSION**</u>

a) To generate square wave at the DAC2 output

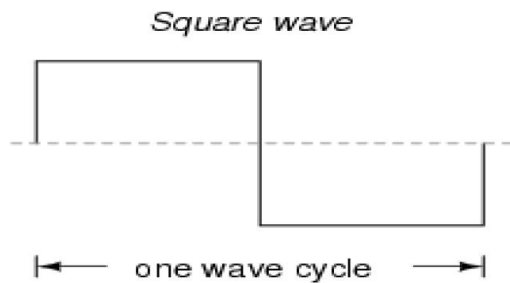**Source code:**

```
                    ORG   4100
START        :      MVI        A, 00
                    OUT        0C8H
                    CALL DELAY
                    MVI        A, 0FF
                    OUT        0C8H
                    CALL DELAY
                    JMP        START
DELAY        :      MVI        B, 05
L1           :      MVI        C, 0FF
L2           :      DCR        C
                    JNZ        L2
                    DCR        B
                    JNZ        L1
                    RET
```

WAVEFORM:

Square wave

|← one wave cycle →|

CALCULATION:

Amplitude:

Time Period:

RESULT: Hence the Square wave is generated.

b) To generate sine-wave at DAC1 output.

Source code:

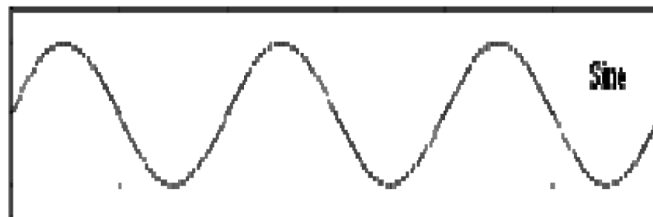```
                        ORG   4100H
START   :   LXI   H, 4110H
                        MVI   C, 46
LOOP   :   MOV A, M
                        OUT   0C0H
                        INX   H
                        DCR   C
                        JNZ   LOOP
                        JMP   START
```

**LOOK-UP TABLE   : (4110)**

| | | | |
|----|----|----|----|
| 7F | 8A | 95 | A0 |
| AA | B5 | BF | C8 |
| D1 | D9 | E0 | E7 |
| ED | F2 | F7 | FA |
| FC | FE | FF | FE |
| FC | FA | F7 | F2 |
| ED | E7 | E0 | D9 |
| D1 | C8 | BF | B5 |
| AA | A0 | 95 | 8A |
| 7F | 74 | 69 | 5F |
| 53 | 49 | 3F | 36 |
| 2D | 25 | 1D | 17 |
| 10 | 0B | 07 | 04 |
| 01 | 00 | 01 | 04 |
| 07 | 0B | 10 | 17 |
| 1D | 25 | 2D | 36 |
| 3F | 49 | 53 | 57 |
| 69 | 74 | | |

WAVEFORM:



Sine

CALCULATION:   Amplitude:                              Time Period:

**RESULT:** Hence the Sine wave is generated.

54

**c) To generate triangular waveform at DAC2 output**

**Source code:**

```
                ORG  4100H
START      :    MVI        L, 00
L1         :    MOV  A, L
                OUT        0C8H
                INR        L
                JNZ        L1
                MVI        L, 0FFH
L2         :    MOV  A, L
                OUT        0C8H
                DCR        L
                JNZ        L2
                JMP        START
```

CALCULATION:

Amplitude:

Time Period:

RESULT: Hence the Triangular wave is generated.

d) To create a saw-tooth wave at the output of DAC1.

Source code:

```
                    ORG  4100H
START        :      MVI  A, 00H
L1           :      OUT         0C0H
                    INR         A
                    JNZ         L1
                    JMP         START
```

CALCULATION:
Amplitude:

Time Period:

RESULT: Hence the Saw-tooth wave is generated.

EXPERIMENT 18   <u>**STEPPER MOTOR CONTROLLER**</u>

a. Stepper motor at different speeds

Aim: To write an ALP for run a stepper motor at different speeds in two directions and observe

the actions which takes place.

Apparatus:

1. Micro-85EB 8085 µP kit

2. Stepper motor Interface Module

3. Bus card

Source Code:

```
            START:      LXI    H, LOOK UP
                        MVI    B,04
            REPT:       MOV    A,M
                        OUT    0C0H
                        LXI    D, 0303H
            DELAY       NOP
                        DCX    D
                        MOV    A,E
                        ORA    D
                        JNZ    DELAY
                        INX    H
                        DCR    B
                        JNZ    REPT
                        JMP    START
            LOOK UP:
                DB:   09     05     06     0A
```

Procedure:

1. Enter the above program starting from 4100h. Connect the stepper motor in port1 and
   execute.

2. The stepper motor can be rotates. Speed can be varied by varying the count at DE pair.

3. Direction can be varied by entering the data in the LOOK UP table in the reverse order.

b. Stepper motor at different angles

Aim: To write an ALP for run a stepper motor for required angle within 360^0, which is equivalent to 256 steps.

Apparatus:

     1. Micro-85EB 8085 µP kit

     2. Stepper motor Interface Module

     3. Bus card

Source Code:

```
                MVI   C, HEX DATA
    START:      LXI   H, LOOK UP
                MVI   B,04
    REPT:       MOV A,M
                OUT   C0
                DCR   C
                JZ    END
                LXI   D, COUNT
    DELAY:      NOP
                DCX   D
                MOV A,E
                ORA   D
                JNZ   DELAY
                INX   H
                DCR   B
                JNZ   REPT
                JMP   START
LOOK UP:
    DB    :09    05    06    0A
    END:        HLT
```

Procedure:

   1. Enter the above program. Connect the stepper motor in port1 and execute.

   2. By converting the required steps in decimal to hex and entering the hex data at 4101h.

   3. The motor rotates for so much steps and then stops.

C. Stepper motor at both directions

Aim: To write an ALP for run stepper motor in both forward and reverse directions with delay.

Apparatus:

    1. Micro-85EB 8085 µP kit

    2. Stepper motor Interface Module

    3. Bus card

Source Code:

```
START:      MVI   C,20H
FORWD:      LXI   H, FORLOOK
            CALL  ROTATE
            DCR   C
            JNZ   FORWD
            CALL  STOP
            MVI   C,20H
REVES:      LXI   H,REVLOOK
            CALL  ROTATE
            DCR   C
            JNZ   REVES
            CALL  STOP
            JMP   START
ROTATE:     MVI   B,04H
REPT:       MOV   A,M
            OUT   C0H
            LXI   D,0303H
LOOP1:      DCX   D
            MOV   A,E
            ORA   D
            JNZ   LOOP1
            INX   H
            DCR   B
            JNZ   REPT
            RET

STOP:       LXI   D,FFFFH
LOOP2:      DCX   D
            MOV   A,E
            ORA   D
            JNZ   LOOP2
            RET
FORLOOK
```

```
        DB      09H     05H     06H     0AH
REVLOOK
        DB      0AH     06H     05H     09H
                END
```

Procedure:

1. Enter the above program starting from 4100h.

2. Connect the stepper motor in port1 and execute.

3. Observe that the stepper motor runs in forward direction and reverse direction continuously with a delay.

RESULT:

Hence the stepper is rotated in different directions and different angles and different speeds.

EXPERIMENT 19   8279- PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

Aim: To display the rolling message 'HELP US" in the display.

Apparatus:

     1. Micro-85EB 8085 µP kit

     2. 8279 Interface Module (Key board & Display)

     3. Bus card

Equivalent:

CNT	EQU	C2H
DAT	EQU	C0H
POINTER	EQU	412CH

Source Code:

```
        START:      LXI    H, POINTER
                    MVI    D, 0FH
                    MVI    A, 10H
                    OUT    CNT
                    MVI    A, CCH
                    OUT    CNT
                    MVI    A, 90H
                    OUT    CNT
        LOP:        MOV    A,M
                    OUT    DAT
                    CALL   DELAY
                    INX    H
                    DCR    D
                    JNZ    LOP
                    JMP    START
        DELAY:      MVI    B, A0H
        LOP1:       MVI    C, FFH
        LOP2:       DCR    C
                    JNZ    LOP2
                    DCR    B
```

JNZ   LOP1

                            RET

POINTER: FF   FF   FF   FF
            FF   FF   FF   FF
            98   68   7C   C8
            1C   29   FF   FF

Procedure:

    1. Enter the above program starting from 4100h.

    2. The data fetched from address 412Ch and display in the first digit of the display.

    3. The next data is displayed in the second digit of the display.

    4. A time delay is given between successive digits for a lively display.

RESULT:

Hence the message HELPUS is displayed.