



**Valliammai Engineering College**



**SRM Nagar, Kattankulathur - 603203**

Department of Electrical and Electronics Engineering

EE6612 – Microprocessors and Microcontrollers Laboratory Manual

**LAB MANUAL**

III Year- VI Semester - Electrical and Electronics Engineering

Academic Year 2015-2016

(2013 Regulation)

Prepared by,

Dr.G.Udhayakumar, Associate Professor

Mr.S.Rajanbabu, Assistant Professor (Sr.G)

Mrs K.Durgadevi, Assistant Professor

Mrs A.Shanmugapriya, Assistant Professor

## **SYLLABUS**

### **EE6612 - MICROPROCESSORS AND MICRO CONTROLLERS LABORATORY**

#### **OBJECTIVES:**

To provide training on programming of microprocessors and microcontrollers and understand the interface requirements.

#### **LIST OF EXPERIMENTS:**

1. Simple arithmetic operations: addition / subtraction / multiplication / division.
2. Programming with control instructions:
  - (i) Ascending / Descending order, Maximum / Minimum of numbers
  - (ii) Programs using Rotate instructions
  - (iii) Hex / ASCII / BCD code conversions.
3. Interface Experiments: with 8085
  - (i) A/D Interfacing. & D/A Interfacing.
4. Traffic light controller.
5. I/O Port / Serial communication
6. Programming Practices with Simulators/Emulators/open source
7. Read a key, interface display
8. Demonstration of basic instructions with 8051 Micro controller execution, including:
  - (i) Conditional jumps, looping
  - (ii) Calling subroutines.
9. Programming I/O Port 8051
  - (i) study on interface with A/D & D/A
  - (ii) study on interface with DC & AC motor .
10. Mini project development with processors.

**TOTAL: 45 PERIODS**

#### **OUTCOMES:**

- Ability to understand and analyse, linear and digital electronic circuits.
- To understand and apply computing platform and software for engineering problems.

## **LIST OF EXPERIMENTS**

### **CYCLE I**

#### **8-bit Microprocessor**

1. Simple arithmetic operations:  
(a) Addition (b) subtraction (c) multiplication (d) division.
2. Ascending / Descending order, Maximum / Minimum of numbers :  
(a) Ascending order (b) Descending order (c) Maximum of numbers (d) Minimum of numbers.
3. Rotate instructions:  
(a) Odd and Even Number (b) Positive and Negative Number
4. Code conversion.  
(a) ASCII to hexadecimal number (b) hexadecimal to ASCII (c) hexadecimal to decimal number (d) binary to hexadecimal number and (e) hexadecimal to binary number

### **CYCLE II**

#### **8085 Interfacing Program**

5. Interface Experiments:  
(a) A/D Interfacing. (b) D/A Interfacing
6. Traffic light controller: I/O port - 8255
7. Serial Communication: 8251
8. Read a key, interface display: 8279

### **CYCLE III**

#### **8-bit Microcontroller**

9. Demonstration of basic instructions with 8051 Micro controller execution, including:  
(a) Addition (b) subtraction (c) multiplication (d) division.
10. (a) Conditional jumps, looping : Sum of elements in an array  
(b) Calling subroutines : Check whether given number is Odd or Even using call option
11. Programming I/O Port :  
(a) A/D Interfacing. (b) D/A Interfacing
12. Interface with DC & AC motor
13. Mini project development with processors

### **ADDITIONAL EXPERIMENTS**

14. Interfacing 8253 Timer With 8085
15. Stepper Motor Interfacing With 8051

<b>S.no</b>	<b>Date</b>	<b>List Of Experiments</b>	<b>Signature</b>


# **8085 MICROPROCESSOR**

**Ex. No: 1**

**SIMPLE ARITHMETIC OPERATIONS**

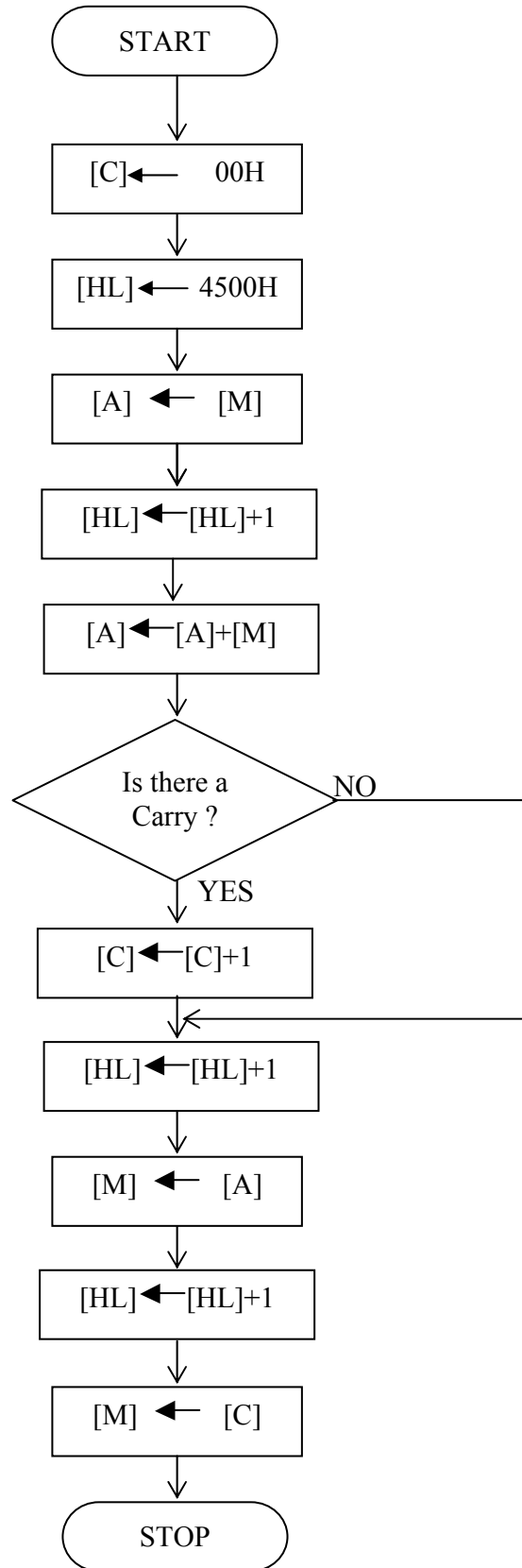
**AIM:**

To write an assembly language program to add, subtract, multiply and divide the given data stored at two consecutive locations using 8085 microprocessor.

**a. 8 BIT DATA ADDITION:**

**ALGORITHM:**

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and add it to the accumulator.
4. Store the answer at another memory location.

**FLOW CHART:**

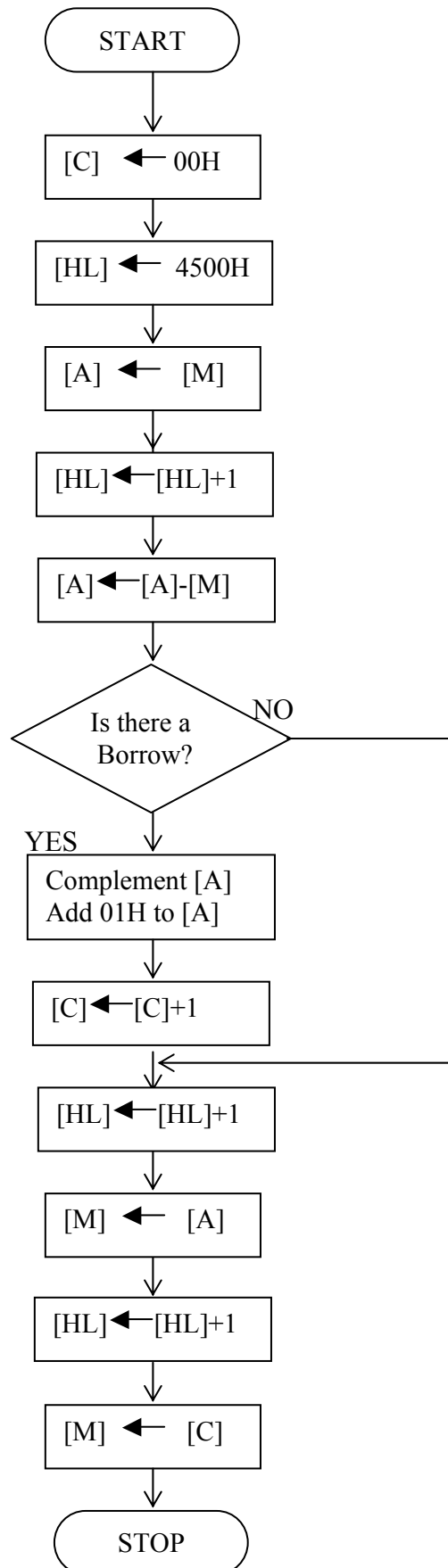
**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	C, 00	Clear C reg.
4101					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	A, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next memory Location.
4107			ADD	M	Add first number to acc. Content.
4108			JNC	L1	Jump to location if result does not yield carry.
4109					
410A					
410B			INR	C	Increment C reg.
410C		L1	INX	H	Increment HL reg. to point next memory Location.
410D			MOV	M, A	Transfer the result from acc. to memory.
410E			INX	H	Increment HL reg. to point next memory Location.
410F			MOV	M, C	Move carry to memory
4110			HLT		Stop the program

**b. 8 BIT DATA SUBTRACTION**

**ALGORITHM:**

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and subtract from the accumulator.
4. If the result yields a borrow, the content of the acc. is complemented and 01H is added to it (2's complement). A register is cleared and the content of that reg. is incremented in case there is a borrow. If there is no borrow the content of the acc. is directly taken as the result.
5. Store the answer at next memory location.

**FLOW CHART:**

**PROGRAM:**

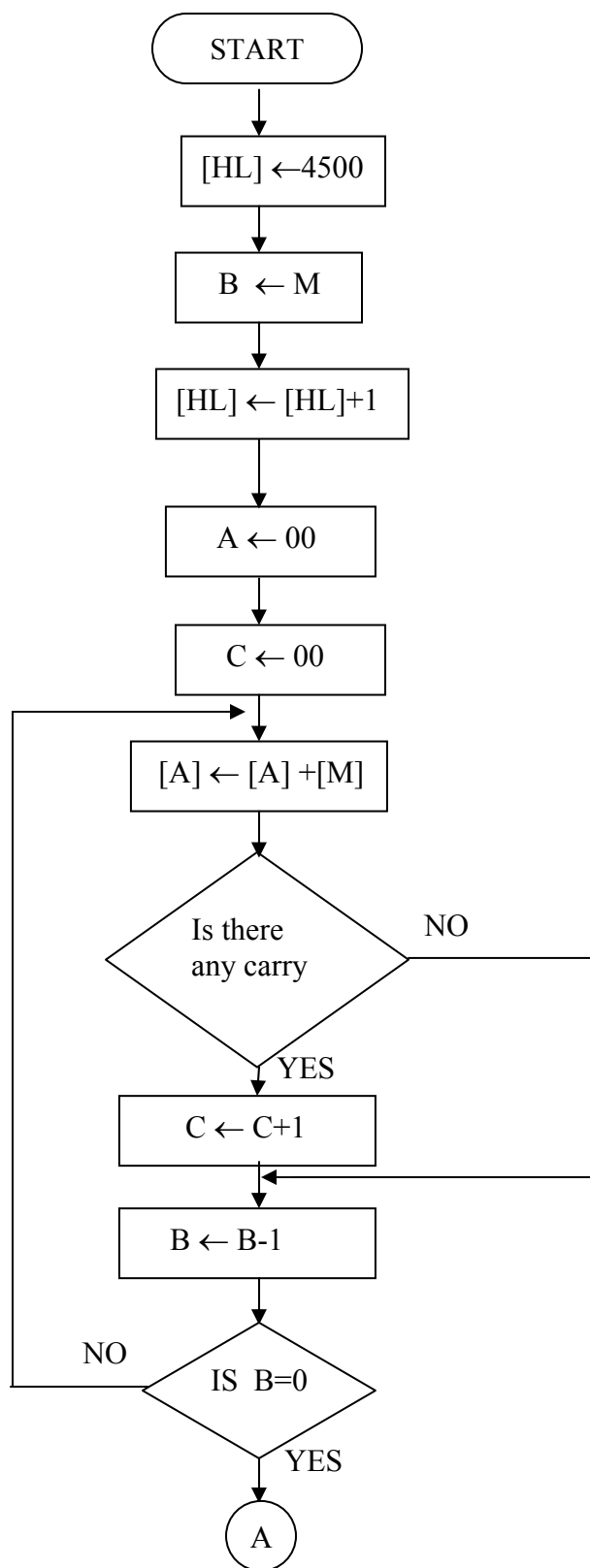
ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	C, 00	Clear C reg.
4101					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	A, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next mem. Location.
4107			SUB	M	Subtract first number from acc. Content.
4108			JNC	L1	Jump to location if result does not yield borrow.
4109					
410A					
410B			INR	C	Increment C reg.
410C			CMA		Complement the Acc. content
410D			ADI	01H	Add 01H to content of acc.
410E					
410F		L1	INX	H	Increment HL reg. to point next mem. Location.
4110			MOV	M, A	Transfer the result from acc. to memory.
4111			INX	H	Increment HL reg. to point next mem. Location.
4112			MOV	M, C	Move carry to mem.
4113			HLT		Stop the program

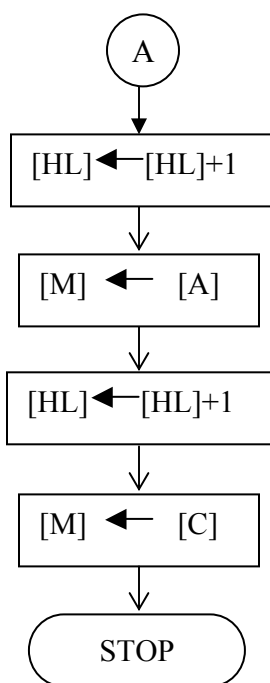
**c. 8 BIT DATA MULTIPLICATION:**

**ALGORITHM:**

**LOGIC:** Multiplication can be done by repeated addition.

1. Initialize memory pointer to data location.
2. Move multiplicand to a register.
3. Move the multiplier to another register.
4. Clear the accumulator.
5. Add multiplicand to accumulator
6. Decrement multiplier
7. Repeat step 5 till multiplier comes to zero.
8. The result, which is in the accumulator, is stored in a memory location.

**FLOW CHART:**



**PROGRAM:**

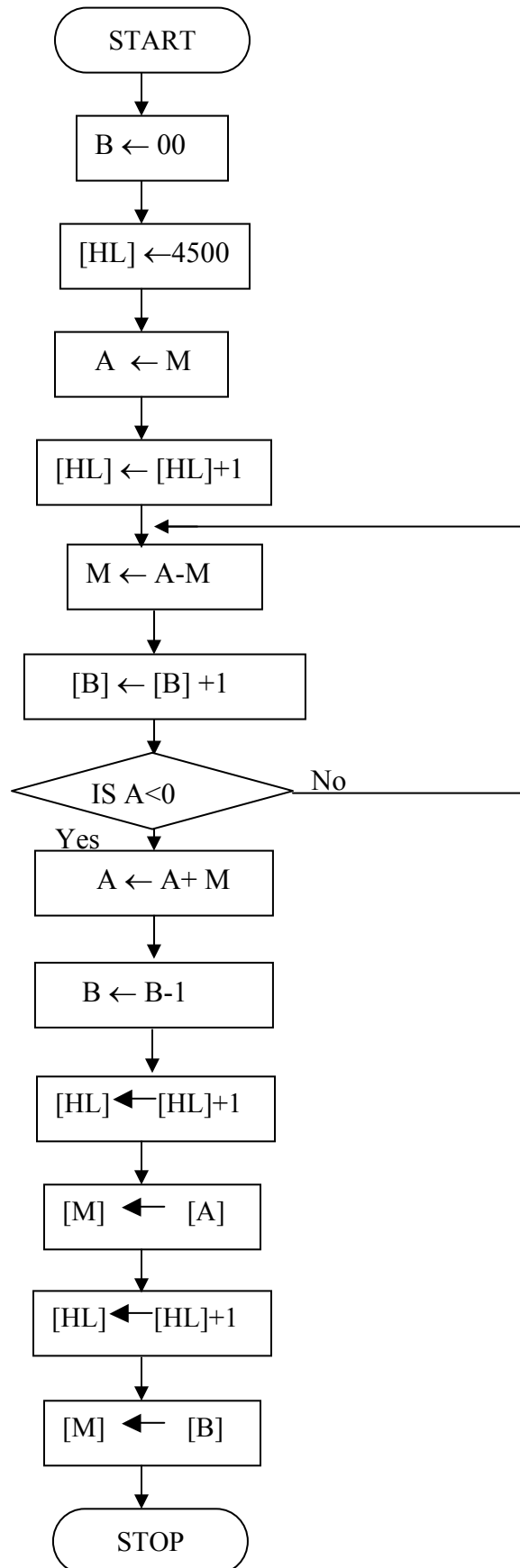
ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	LXI	H, 4500	Initialize HL reg. to 4500
4101					
4102					
4103			MOV	B, M	Transfer first data to reg. B
4104			INX	H	Increment HL reg. to point next mem. Location.
4105			MVI	A, 00H	Clear the acc.
4106					
4107			MVI	C, 00H	Clear C reg for carry
4108					
4109		L1	ADD	M	Add multiplicand multiplier times.
410A			JNC	NEXT	Jump to NEXT if there is no carry
410B					
410C					
410D			INR	C	Increment C reg
410E		NEXT	DCR	B	Decrement B reg
410F			JNZ	L1	Jump to L1 if B is not zero.
4110					
4111					
4112			INX	H	Increment HL reg. to point next mem. Location.
4113			MOV	M, A	Transfer the result from acc. to memory.
4114			INX	H	Increment HL reg. to point next mem. Location.
4115			MOV	M, C	Transfer the result from C reg. to memory.
4116			HLT		Stop the program

#### **d. 8 BIT DIVISION:**

##### **ALGORITHM:**

**LOGIC:** Division is done using the method Repeated subtraction.

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:**

**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100			MVI	B,00	Clear B reg for quotient
4101					
4102			LXI	H,4500	Initialize HL reg. to 4500H
4103					
4104					
4105			MOV	A,M	Transfer dividend to acc.
4106			INX	H	Increment HL reg. to point next mem. Location.
4107		LOOP	SUB	M	Subtract divisor from dividend
4108			INR	B	Increment B reg
4109			JNC	LOOP	Jump to LOOP if result does not yield borrow
410A					
410B					
410C			ADD	M	Add divisor to acc.
410D			DCR	B	Decrement B reg
410E			INX	H	Increment HL reg. to point next mem. Location.
410F			MOV	M,A	Transfer the remainder from acc. to memory.
4110			INX	H	Increment HL reg. to point next mem. Location.
4111			MOV	M,B	Transfer the quotient from B reg. to memory.
4112			HLT		Stop the program

**OBSERVATION:**

**ADDITION:**

S.NO	INPUT		OUTPUT	
	ADDRESS	DATA	ADDRESS	DATA
1	4500		4502	
	4501		4503	
2	4500		4502	
	4501		4503	

**SUBTRACTION:**

S.NO	INPUT		OUTPUT	
	ADDRESS	DATA	ADDRESS	DATA
1	4500		4502	
	4501		4503	
2	4500		4502	
	4501		4503	

**MULTIPLICATION:**

S.NO	INPUT		OUTPUT	
	ADDRESS	DATA	ADDRESS	DATA
1	4500		4502	
	4501		4503	
2	4500		4502	
	4501		4503	

**DIVISION:**

S.NO	INPUT		OUTPUT	
	ADDRESS	DATA	ADDRESS	DATA
1	4500		4502	
	4501		4503	
2	4500		4502	
	4501		4503	



**RESULT:**

Thus the addition, subtraction, multiplication and division of two numbers was performed using the 8085 microprocessor.

**Ex. No: 2 Ascending / Descending order, Maximum / Minimum of numbers**

**AIM:**

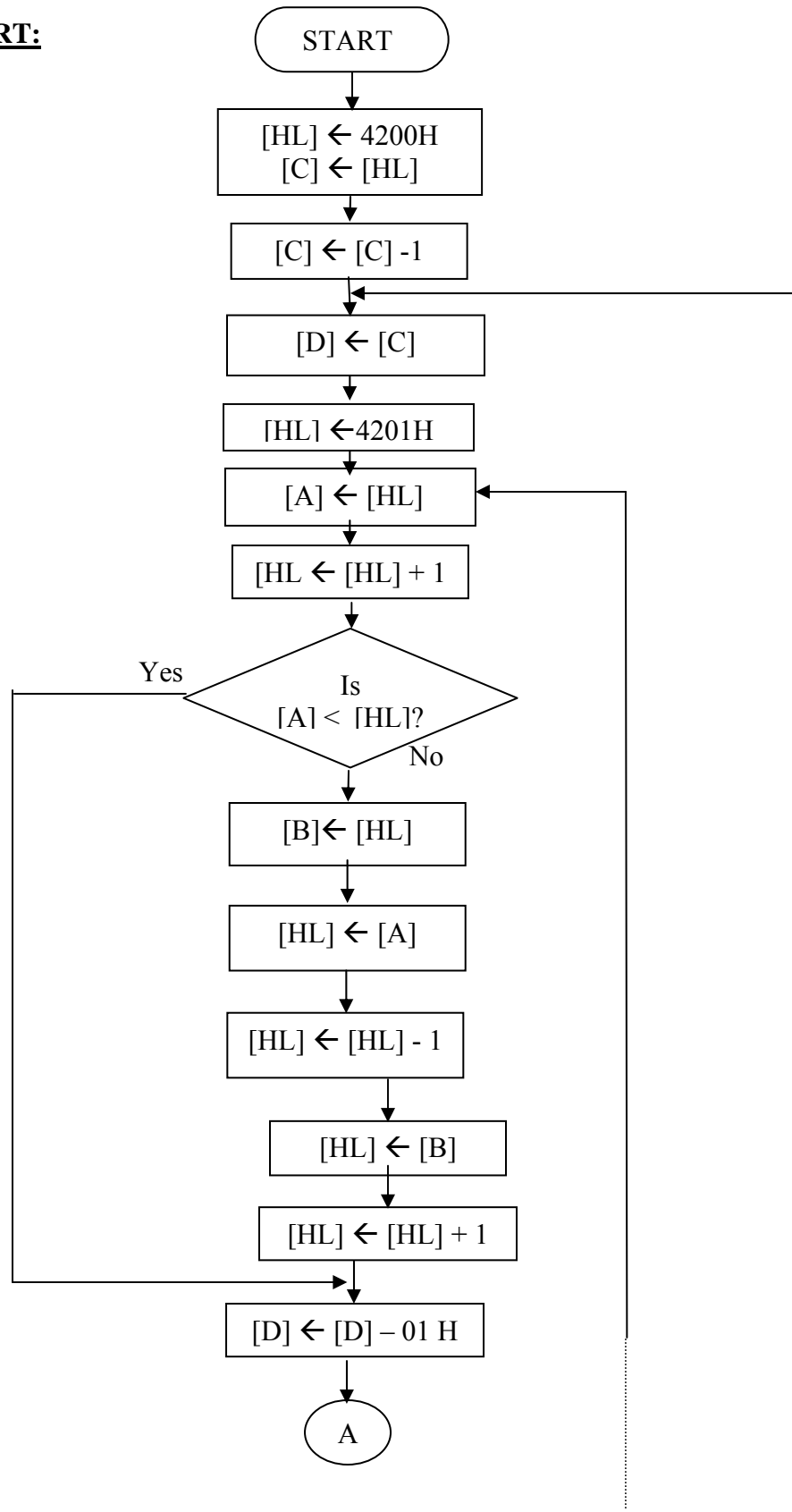
To write an assembly language program to arrange an array of data in ascending and descending order and to find the smallest and largest data among the array.

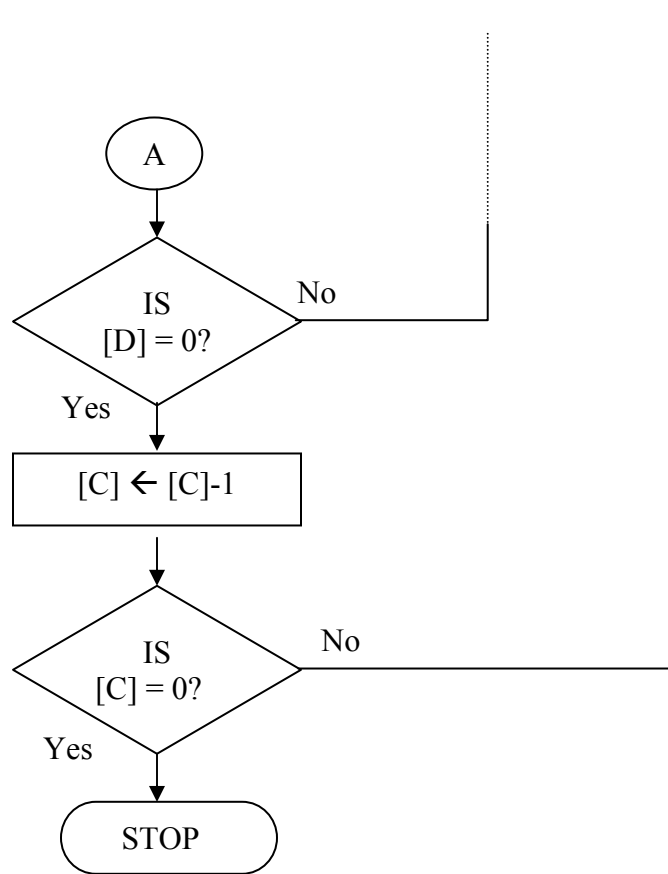
**a. ASCENDING ORDER**

**ALGORITHM:**

1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is larger than second then interchange the number.
3. If the first number is smaller, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

**FLOWCHART:**





**PROGRAM:**

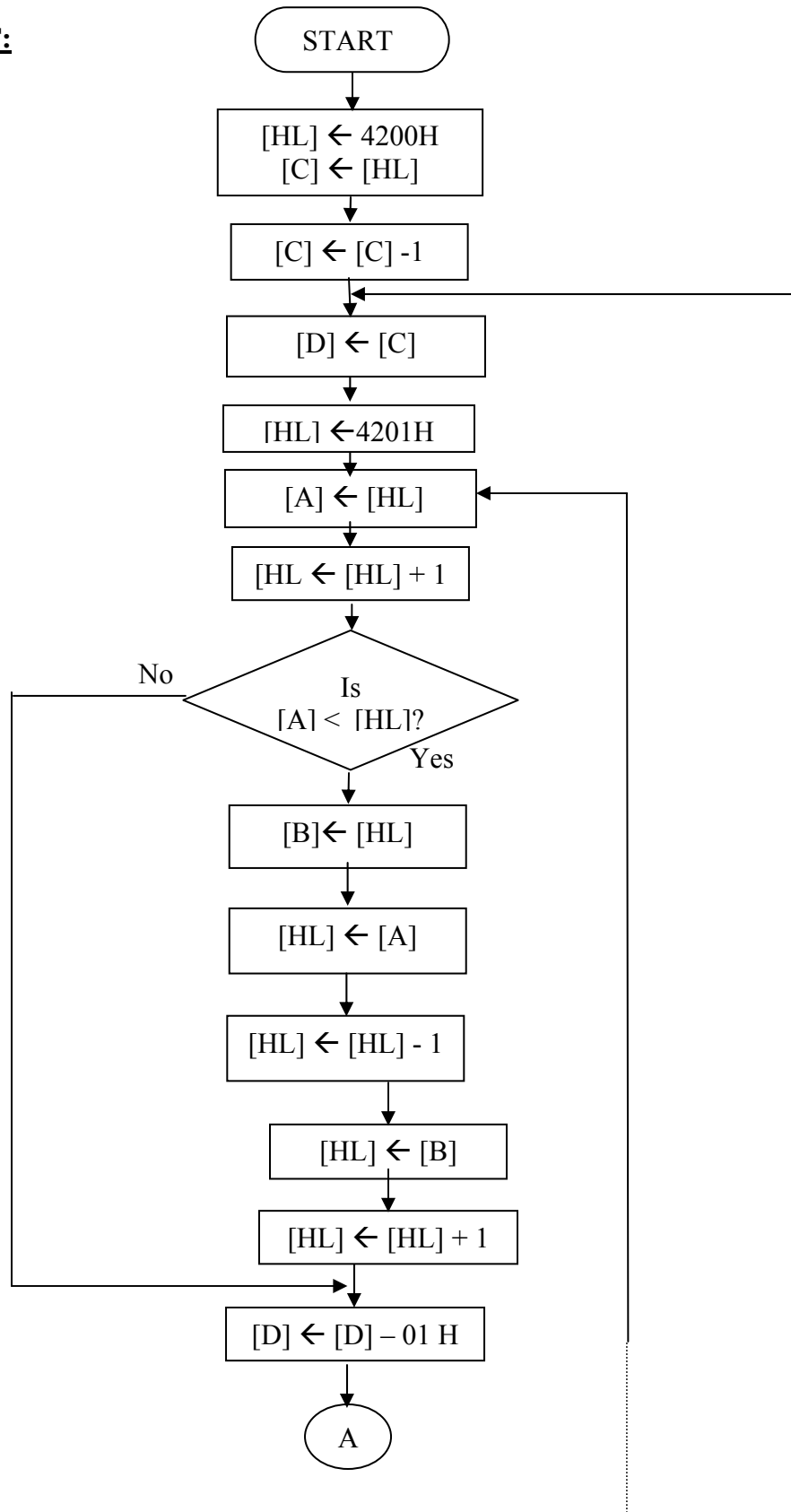
ADDRESS	OPC ODE	LABEL	MNEMONICS	OPERA ND	COMMENTS
4100			LXI	H,4200	Set pointer for array
4101					
4102					
4103			MOV	C,M	Load the Count
4104			DCR	C	Decrement Count
4105		LOOP 3	MOV	D,C	Transfer data from C to D
4106			LXI	H,4201	Load data from 4201
4107					
4108					
4109		LOOP2	MOV	A,M	Copy content of M to A
410A			INX	H	Increment HL reg. to point next memory location
410B			CMP	M	Compare M & A
410C			JC	LOOP1	If A is lesser than M then go to loop1
410D					
410E					
410F			MOV	B,M	Transfer data from M to D reg
4110			MOV	M,A	Transfer data from acc to M
4111			DCX	H	Decrement HL pair
4112			MOV	M,B	Transfer data from B to M
4113			INX	H	Increment HL pair
4114		LOOP1	DCR	D	Decrement D reg
4115			JNZ	LOOP2	If D is not zero go to loop2
4116					
4117					
4118			DCR	C	Decrement C reg
4119			JNZ	LOOP3	If C is not Zero go to loop3
411A					
411B					
411C			HLT		Stop the program

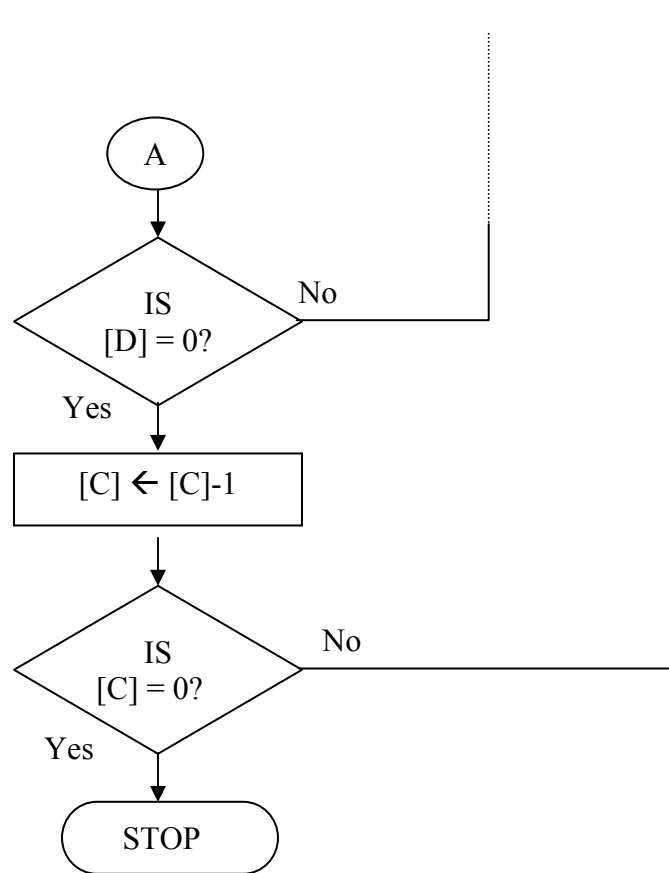
**b. DESCENDING ORDER**

**ALGORITHM:**

1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is smaller than second then I interchange the number.
3. If the first number is larger, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

**FLOWCHART:**





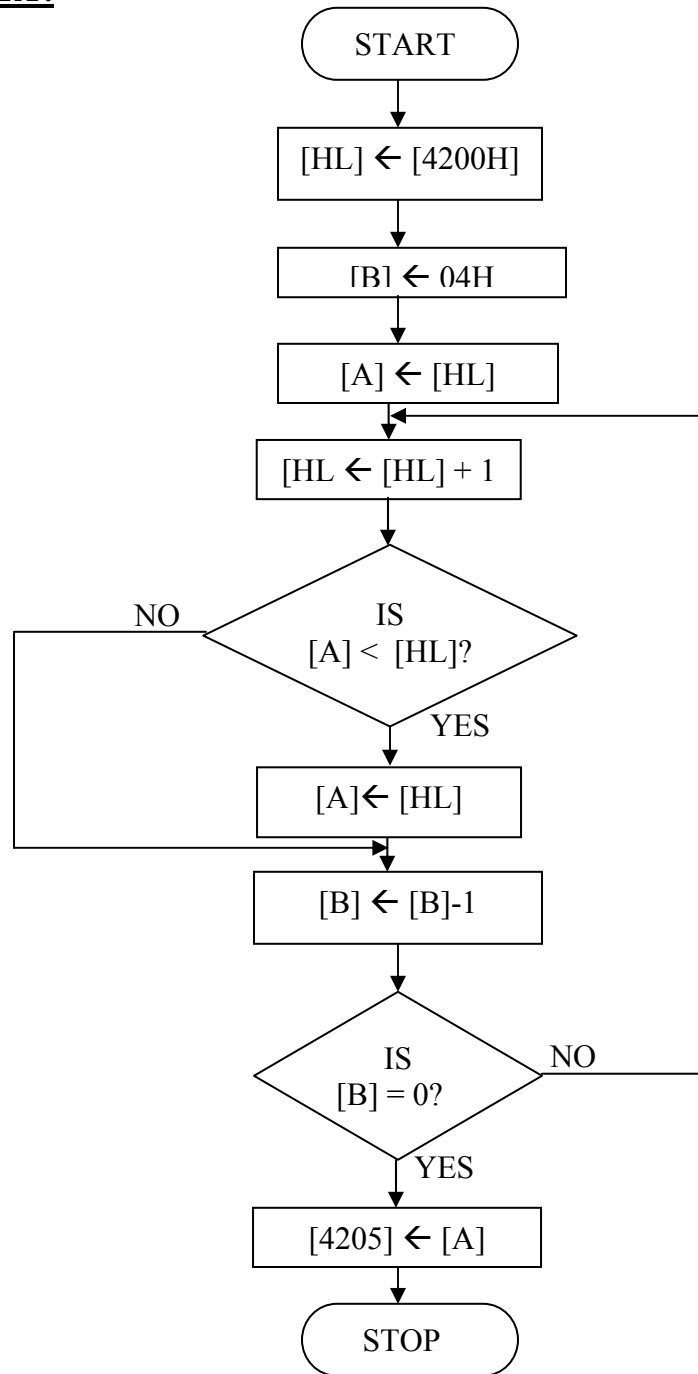
**PROGRAM:**

ADDRESS	OPC ODE	LABEL	MNEMONICS	OPERA ND	COMMENTS
4100			LXI	H,4200	Set pointer for array
4101					
4102					
4103			MOV	C,M	Load the Count
4104			DCR	C	Decrement Count
4105		LOOP 3	MOV	D,C	Transfer data from C to D
4106			LXI	H,4201	Load data from 4201
4107					
4108					
4109		LOOP2	MOV	A,M	Copy content of M to A
410A			INX	H	Increment HL reg. to point next memory location
410B			CMP	M	Compare M & A
410C			JNC	LOOP1	If A is lesser than M then go to loop1
410D					
410E					
410F			MOV	B,M	Transfer data from M to D reg
4110			MOV	M,A	Transfer data from acc to M
4111			DCX	H	Decrement HL pair
4112			MOV	M,B	Transfer data from B to M
4113			INX	H	Increment HL pair
4114		LOOP1	DCR	D	Decrement D reg
4115			JNZ	LOOP2	If D is not zero go to loop2
4116					
4117					
4118			DCR	C	Decrement C reg
4119			JNZ	LOOP3	If C is not Zero go to loop3
411A					
411B					
411C			HLT		Stop the program

c. LARGEST ELEMENT IN AN ARRAY

**ALGORITHM:**

1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero
10. Store the result (accumulator content) in the specified memory location.

**FLOW CHART:**

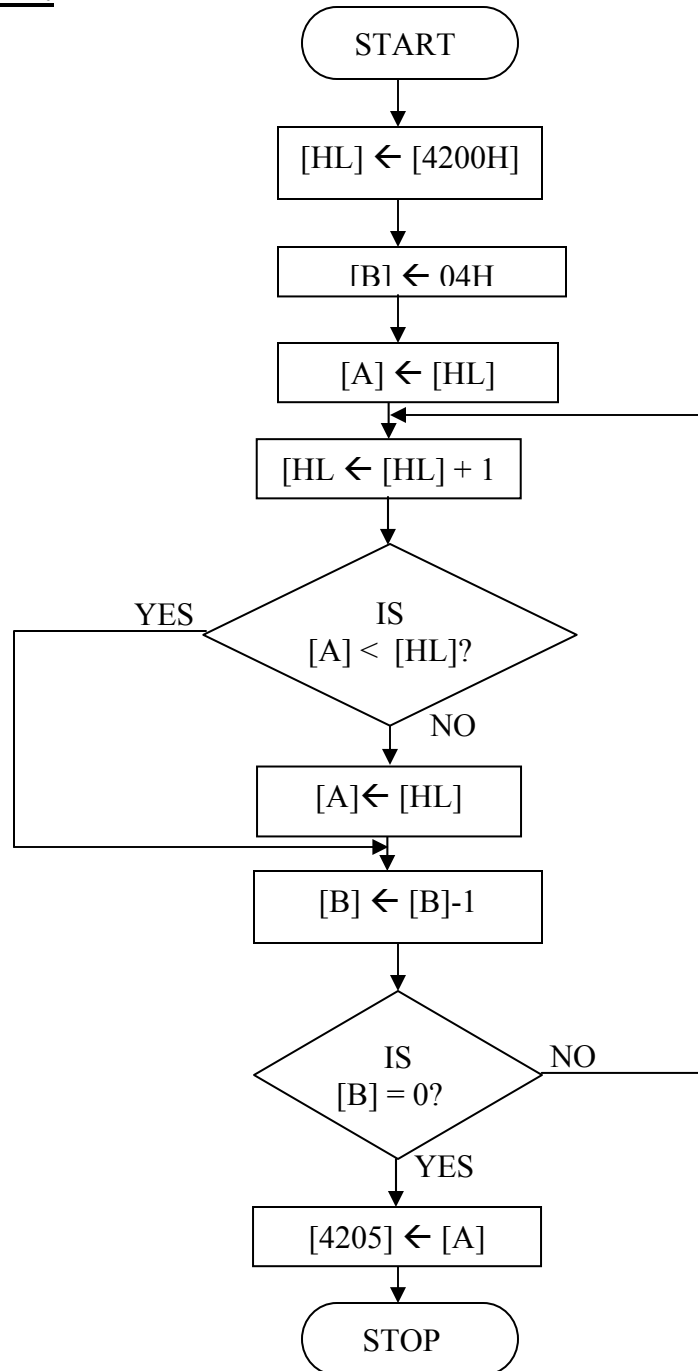
**PROGRAM:**

ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPER AND	COMMENTS
4101			LXI	H,4200	Initialize HL reg. to 4200H
4102					
4103					
4104			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
4105					
4106			MOV	A,M	Transfer first data to acc.
4107		LOOP1	INX	H	Increment HL reg. to point next memory location
4108			CMP	M	Compare M & A
4109			JNC	LOOP	If A is greater than M then go to loop
410A					
410B					
410C			MOV	A,M	Transfer data from M to A reg
410D		LOOP	DCR	B	Decrement B reg
410E			JNZ	LOOP1	If B is not Zero go to loop1
410F					
4110					
4111			STA	4205	Store the result in a memory location.
4112					
4113					
4114			HLT		Stop the program

d. SMALLEST ELEMENT IN AN ARRAY

**ALGORITHM:**

1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero
10. Store the result (accumulator content) in the specified memory location.

**FLOW CHART:**

**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4101			LXI	H,4200	Initialize HL reg. to 4200H
4102					
4103					
4104			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
4105					
4106			MOV	A,M	Transfer first data to acc.
4107		LOOP1	INX	H	Increment HL reg. to point next memory location
4108			CMP	M	Compare M & A
4109			JC	LOOP	If A is lesser than M then go to loop
410A					
410B					
410C			MOV	A,M	Transfer data from M to A reg
410D		LOOP	DCR	B	Decrement B reg
410E			JNZ	LOOP1	If B is not Zero go to loop1
410F					
4110					
4111			STA	4205	Store the result in a memory location.
4112					
4113					
4114			HLT		Stop the program

**OBSERVATION:****A. ASCENDING ORDER**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4200		4200	
4201		4201	
4202		4202	
4203		4203	
4204		4204	

**B. DESCENDING ORDER**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4200		4200	
4201		4201	
4202		4202	
4203		4203	
4204		4204	

**C. SMALLEST ELEMENT**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4200		4205	
4201			
4202			
4203			
4204			

**D. LARGEST ELEMENT**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4200		4205	
4201			
4202			
4203			
4204			

**RESULT:**

Thus the sorting operations of arranging an array in ascending, descending order and the largest and smallest element were found using the 8085 microprocessor.

**Ex. No: 3**

**FIND THE NUMBER OF EVEN AND ODD NUMBER IN  
A BLOCK OF DATA**

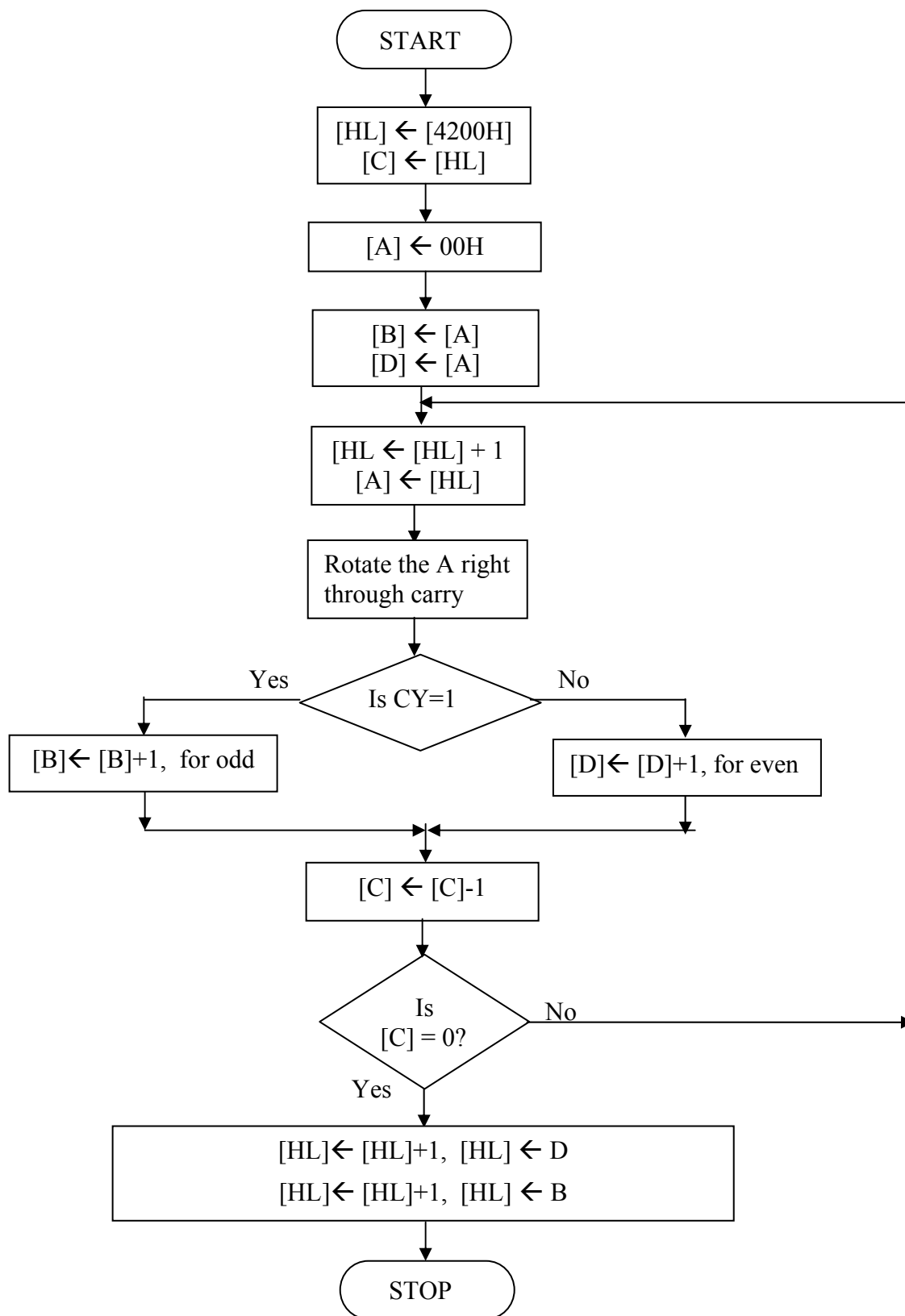
**AIM:**

To find the number of even and odd number, and number of positive and negative numbers in a block of data.

**a. Odd and Even Number**

**ALGORITHM:**

1. Start.
2. Load the given data.
3. Clear the data in accumulator.
4. Move the content of accumulator to D and B register.
5. Increment HL register pair and move the content of memory to accumulator.
6. Rotate the obtain data in the form of RAR.
7. If carry occur increment B else increment D.
8. Decrement C, if  $C \neq 0$  go to step 5. Else increment HL pair.
9. Move the content of D to memory and increment HL pair and move the content B to memory.
10. Stop.

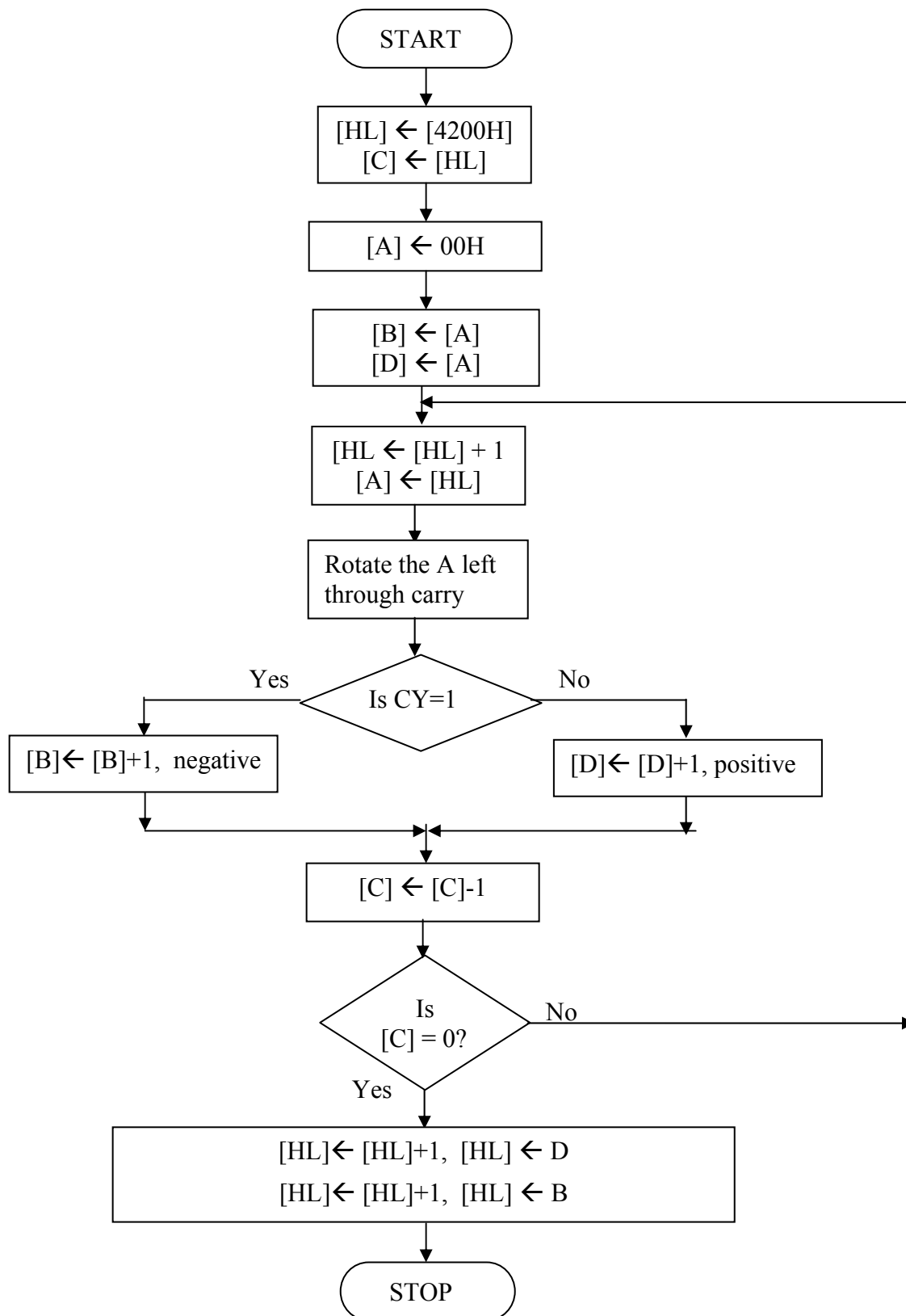


ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100			LXI	H,4200	Load the number of data
4101					
4102					
4103			MOV	C, M	Move M to C
4104			XRA	A	Exclusive OR reg A
4105			MOV	D, A	Move A to D
4106			MOV	B, A	Move A to B
4107		L3	INX H	H	Increment H
4108			MOV	A,M	Move M to A
4109			RAR		Rotate right through carry
410A			JNC	L1	Jump no carry
410B					
410C					
410D			INR	D	Increment D
410E			JMP	L2	Jump to L2
410F					
4110					
4111		L1	INR	B	Increment B
4112		L2	DCR	C	Decrement C
4113			JNZ	L3	Jump non Zero
4114					
4115					
4116			INX	H	Increment H
4117		LOOP	MOV	M, D	Move D to M
4118			INX H	H	Increment H
4119			MOV	M, B	Move B to M
411A			HLT		Halt the program

**b. Positive and Negative Number**

**ALGORITHM:**

1. Start.
2. Load the given data.
3. Clear the data in accumulator.
4. Move the content of accumulator to D and B register.
5. Increment HL register pair and move the content of memory to accumulator.
6. Rotate the obtain data in the form of RAL.
7. If carry occur increment B else increment D.
8. Decrement C, if  $C \neq 0$  go to step 5. Else increment HL pair.
9. Move the content of D to memory and increment HL pair and move the content B to memory.
10. Stop.



ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPER AND	COMMENTS
4100			LXI	H,4200	Load the number of data
4101					
4102					
4103			MOV	C, M	Move M to C
4104			XRA	A	Exclusive OR reg A
4105			MOV	D, A	Move A to D
4106			MOV	B, A	Move A to B
4107		L3	INX H	H	Increment H
4108			MOV	A,M	Move M to A
4109			RAL		Rotate left through carry
410A			JNC	L1	Jump no carry
410B					
410C					
410D			INR	D	Increment D
410E			JMP	L2	Jump to L2
410F					
4110					
4111		L1	INR	B	Increment B
4112		L2	DCR	C	Decrement C
4113			JNZ	L3	Jump non Zero
4114					
4115					
4116			INX	H	Increment H
4117		LOOP	MOV	M, D	Move D to M
4118			INX H	H	Increment H
4119			MOV	M, B	Move B to M
411A			HLT		Halt the program

## **Observation**

### **a. Odd and even number**

#### **INPUT:**

4200 = 04

4201 = 01

4202 = 05

4203 = 03

4204 = 04

#### **OUTPUT:**

4205=03 (Odd numbers)

4206=01 (Even numbers)

### **b. Positive and negative number**

#### **INPUT:**

4200 = 04

4201 = 01

4202 = F5

4203 = 03

4204 = 74

#### **OUTPUT:**

4205=01 (Negative numbers)

4206=03 (Positive numbers)

#### **RESULT:**

Thus the number of even and odd number, and positive and negative number has been found from the given number of list.

**Ex. No: 4**

**CODE CONVERSIONS**

**AIM:**

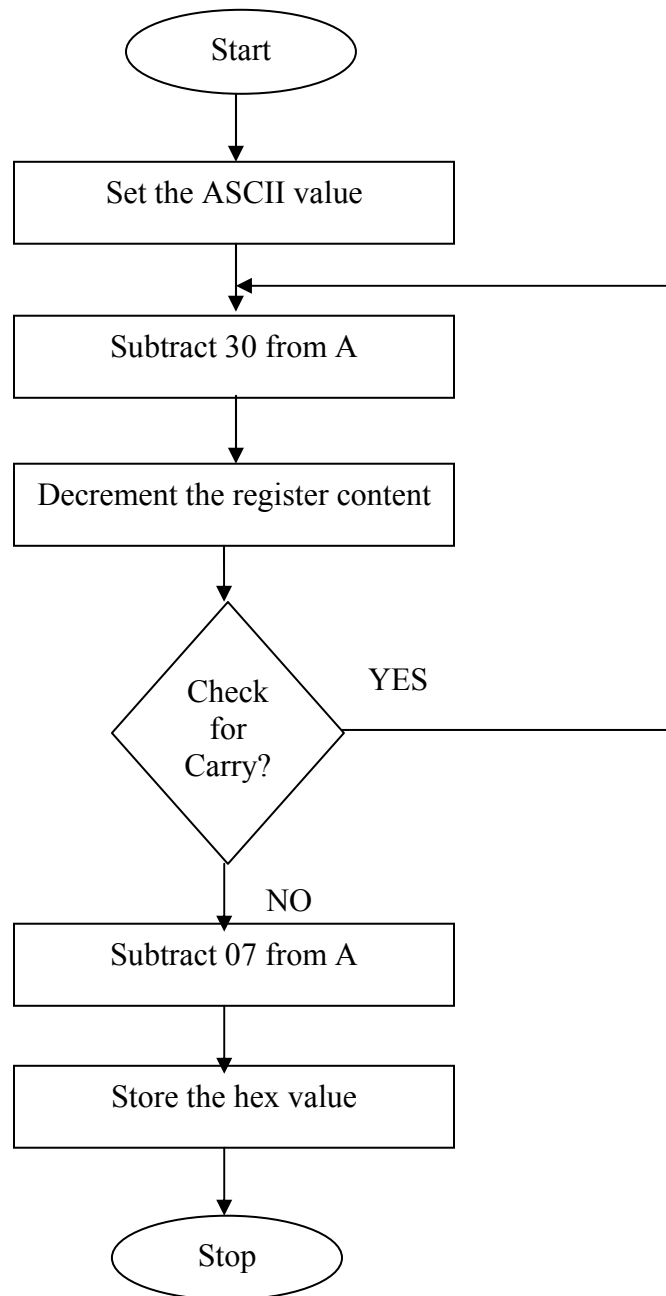
To write an assembly language program to perform the conversions of ASCII to hexadecimal number, hexadecimal to ASCII, hexadecimal to decimal number, binary to hexadecimal number and hexadecimal to binary number.

**a.ASCII TO HEXADECIMAL**

**ALGORITHM:**

1. Start the program
2. Load the data from address 4200 to A
3. Move data from accumulator to C
4. Move data from M to HL pair to accumulator
5. Subtract the data 30 from A
6. Decrement content of register
7. Stop the program if C is zero
8. Jump to Step 5
9. End the program

**FLOWCHART:**



**PROGRAM:**

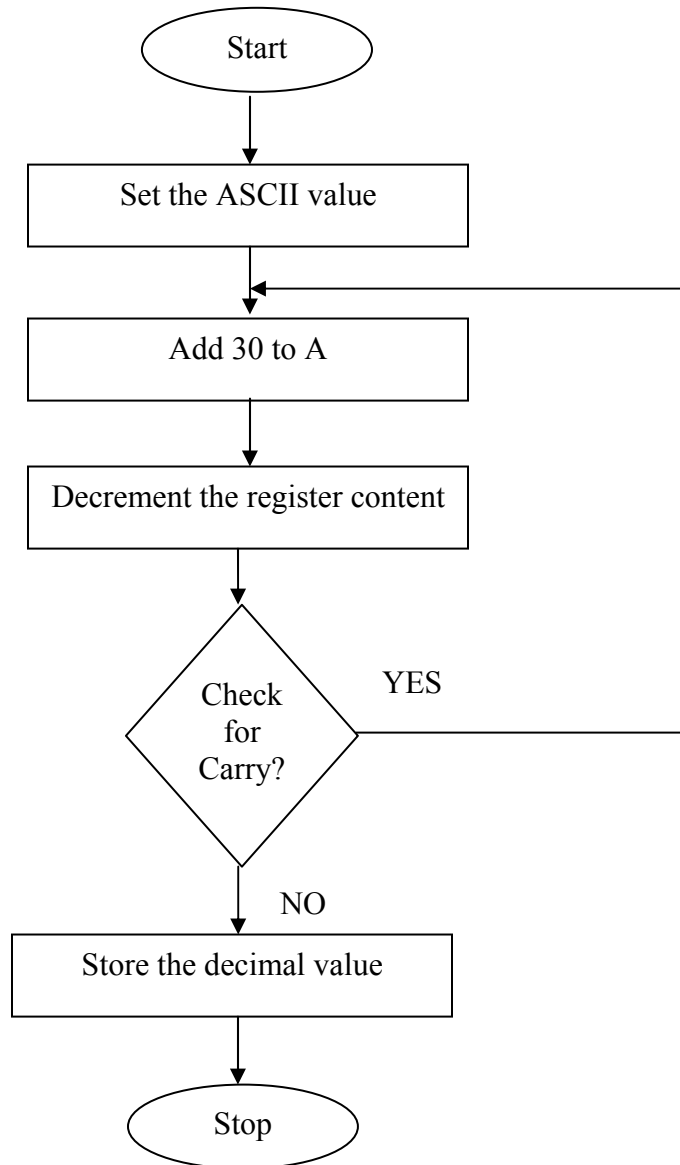
ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPER AND	COMMENTS
4100			LDA	H,4200	Load data 4200 to A
4101					
4102					
4103			MOV	C,A	Move data from A to C
4104			LXI	H,4201	Load address 4201 in HL
4105					
4106					
4107			LXI	D,4301	Load address 4301 in DF
4108					
4109					
410A		LOOP 1	MOV	A,M	Move data from M to A
410B			SUI	30	Subtract 30 from A
410C					
410D			STAX	D	Store data from accumulator to DE
410E			DCR	C	Decrement from C register
410F			JZ	LOOP	Stop program if C is 0
4110					
4111					
4112			INX	H	Increment HL register pair
4113			INX	D	Increment DE register pair
4114			JMP	LOOP 1	Jump to 410A
4115					
4116					
4117		LOOP	HLT		Stop

**b. HEXADECIMAL TO ASCII**

**ALGORITHM:**

1. Start the program
2. Load the data from address 4200 to A
3. Move data from accumulator to C
4. Move data from M to HL pair to accumulator
5. Add the data 30 to A
6. Decrement content of register
7. Stop the program if C is zero
8. Jump to Step 5
9. End the program

**FLOWCHART:**



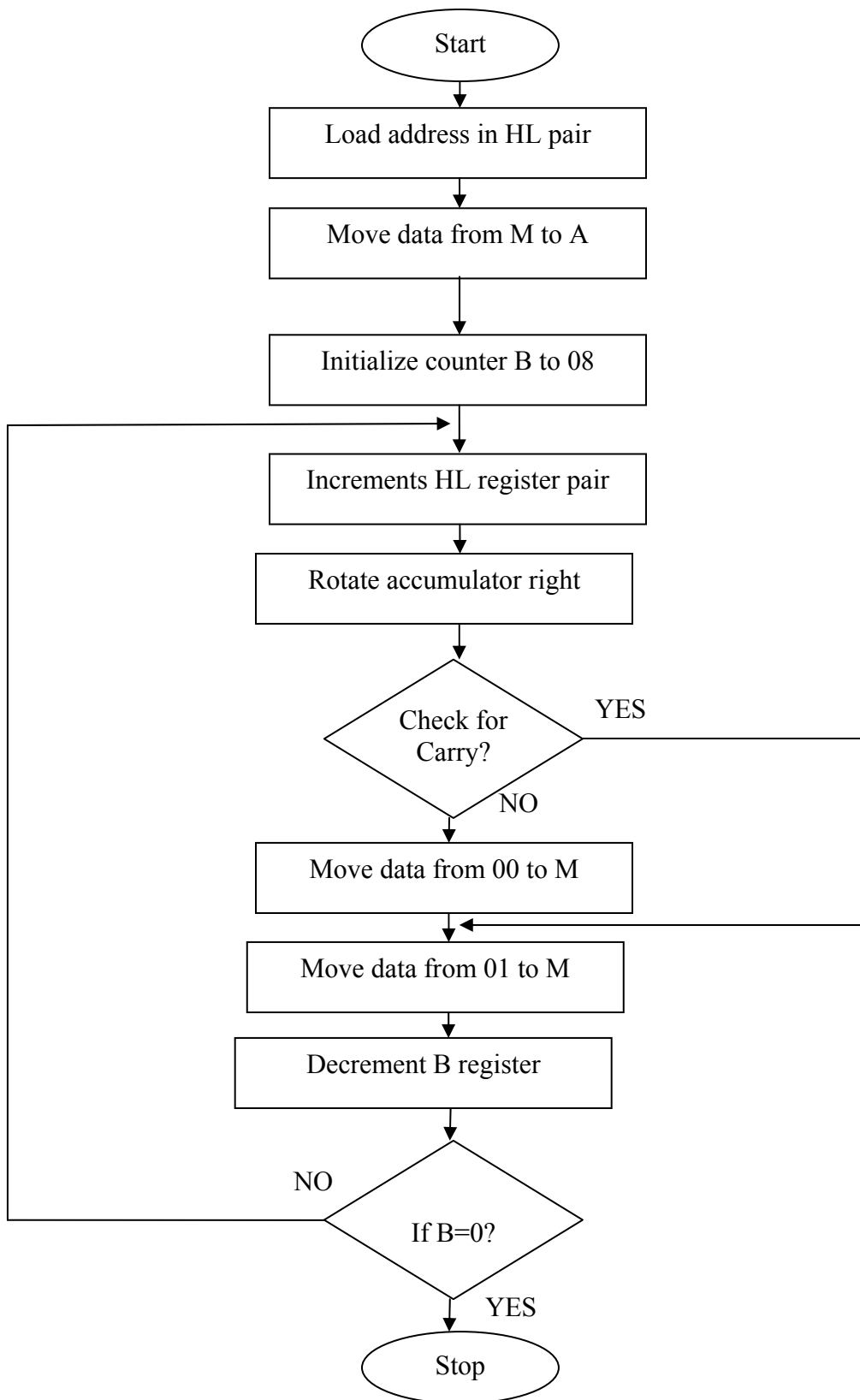
**PROGRAM:**

ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPER AND	COMMENTS
4100			LDA	H,4200	Load data 4200 to A
4101					
4102					
4103			MOV	C,A	Move data from A to C
4104			LXI	H,4201	Load address 4201 in HL
4105					
4106					
4107			LXI	D,4301	Load address 4301 in DF
4108					
4109					
410A		LOOP 1	MOV	A,M	Move data from M to A
410B			ADI	30	Subtract 30 from A
410C					
410D			STAX	D	Store data from accumulator to DE
410E			DCR	C	Decrement from C register
410F			JZ	LOOP	Stop program if C is 0
4110					
4111					
4112			INX	H	Increment HL register pair
4113			INX	D	Increment DE register pair
4114			JMP	LOOP 1	Jump to 410A
4115					
4116					
4117		LOOP	HLT		Stop

**c. HEXADECIMAL TO BINARY**

**ALGORITHM:**

1. Start the program
2. Move the content of memory to accumulator
3. Move data 0B to register B
4. Increment the content of HL register pair
5. Rotate the accumulator right
6. Jump to the specified address if carry generated
7. Move 00 to memory
8. Jump to specified address if there is no zero
9. Move 01 to memory
10. Jump to specified address if there is no zero
11. End the program

**FLOWCHART:**

**PROGRAM:**

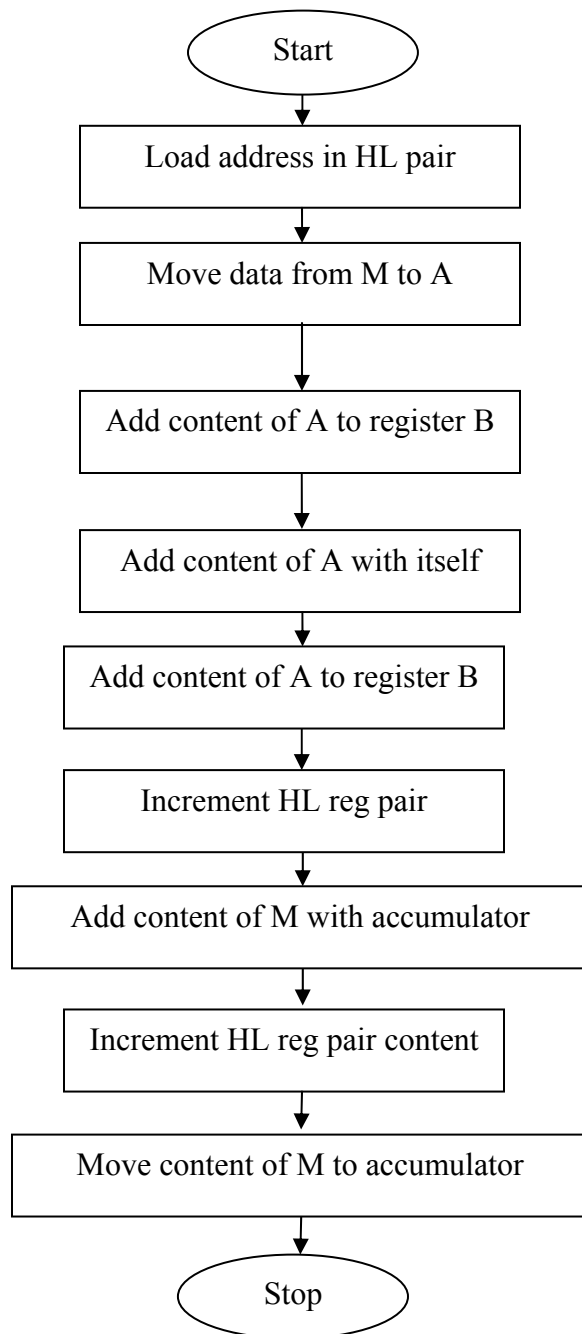
ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100			LXI	H, 4200	Load address in HL pair
4101					
4102					
4103			MOV	A, M	Move content of M to A
4104			MVI	B, 08	Move 08 to register pair
4105					
4106		L3	INX	H	Increment the content of HL pair
4107			RRC		Rotate accumulator right
4108			JC	L1	Jump to specified address if carry
4109					
410A					
410B			MVI	M, 00	Move 00 to M
410C			JMP	L2	Decrement B register
410D					
410E					
410F		L1	MVI	M, 01	Move 01 to M
4110					
4111		L2	DCR	B	Decrement B by 1
4112			JNZ	L3	Jump to the specified address if no zero
4113					
4114					
4115			HLT		Stop the program

#### **d. BINARY TO HEXADECIMAL**

##### **ALGORITHM:**

1. Start the program
2. Load the address in HL pair
3. Move the content of memory to accumulator
4. Add the content of accumulator with previous content of accumulator
5. Move the content of B to accumulator
6. Add the content of accumulator with previous content of accumulator
7. Repeat step 6
8. Add B with accumulator content
9. Increment H by 1
10. Move content of M to A
11. End the program

**FLOWCHART:**



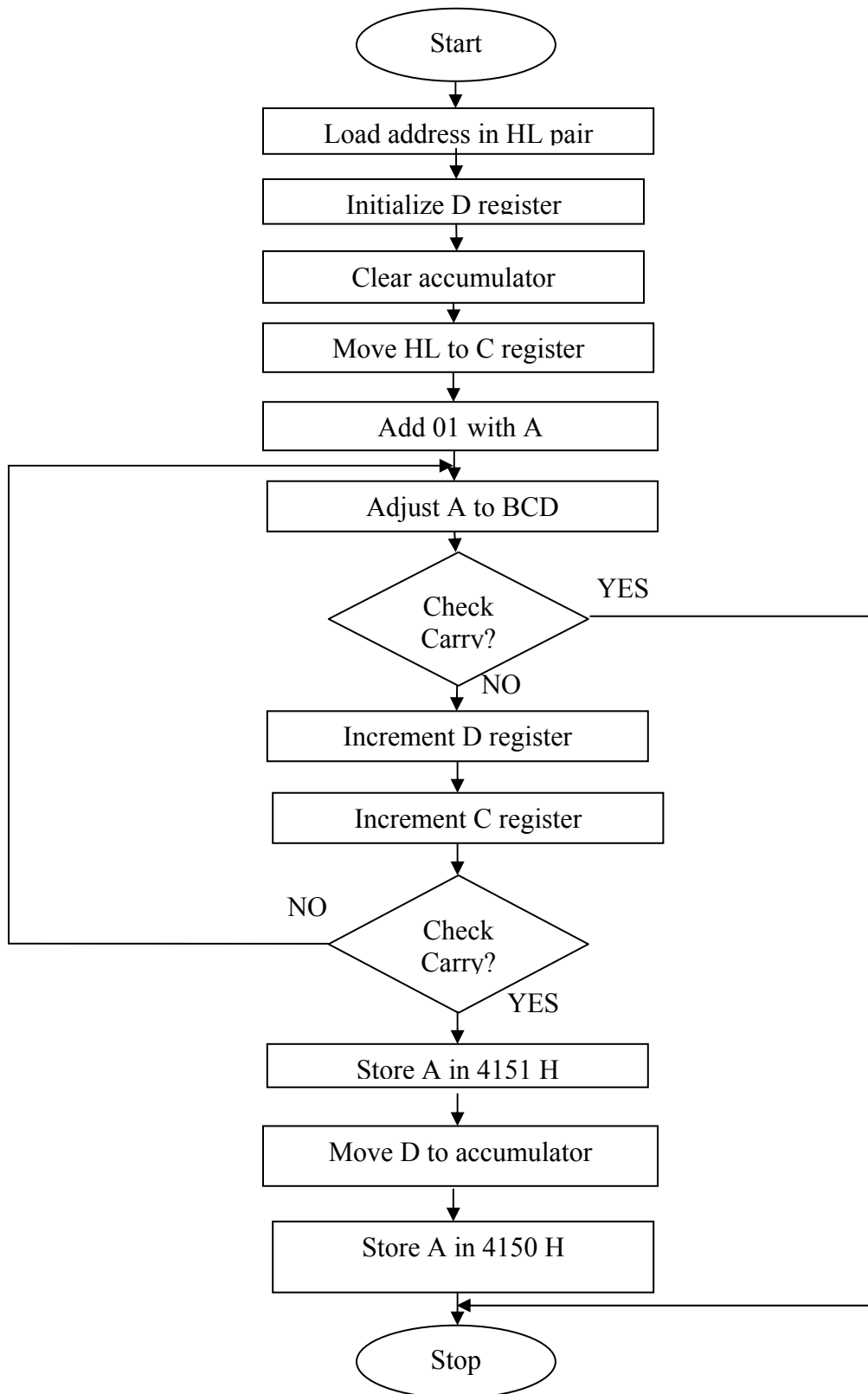
**PROGRAM:**

ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPERAND	COMMENTS
4100			LXI	H,4150	Load address in HL pair
4101					
4102					
4103			MOV	M,A	Move content of A to M
4104			ADD	A	Add A content with previous content of A
4105			MOV	B, A	Move the content from A to B
4106			ADD	A	Add A content with previous content of A
4107			ADD	B	Add B content with A
4108			INX	H	Increment H by 1
4109			ADD	M	Add M content with A
410A			INX	H	Increment H by 1
410B			MOV	M, A	Move content of A to M
410C			HLT		Stop the program

**e. HEXADECIMAL TO DECIMAL**

**ALGORITHM:**

1. Start the program
2. Load the address in HL pair
3. Move the content from HL to A
4. Subtract 64 from A
5. Increment BC pair
6. Jump to address 4207
7. Subtract 0A from A
8. Increment HL pair
9. Rotate accumulator left
10. Increment HL pair
11. End the program

**FLOWCHART:**

**PROGRAM:**

ADDRE SS	OPCO DE	LABEL	MNEM ONICS	OPER AND	COMMENTS
4100			LXI	H, 4150	Load data from 4150 to HL pair
4101					
4102					
4103			LXI	B, 0000	Load data from address to BC
4104					
4105					
4106			MOV	A, M	Move the content from HL to A
4107	L4		SUI	64	Subtract 64 from A
4108					
4109			JC	L1	Stop if A has carry
410A					
410B					
410C			INR	B	Increment BC
410D			JMP	L4	Jump to specified address
410E					
410F					
4110	L1		ADI	64	Add 64 to A
4111					
4112	L3		SUI	0A	Subtract 0A from A
4113					
4114			JC	L2	Stop if A has carry
4115					
4116					
4117			INR	C	Increment HL
4118	L2		JNC	L3	Stop if A has no carry
4119					
411A					
411B			ADI	0A	Add 0A to A
411D			INX	H	Increment HL
411E			MOV	M, B	Move B to M
411F			MOV	B, A	Move A to B
4120			MOV	A, B	Move B to A
4121			RLC		Rotate accumulator
4122			RLC		Rotate accumulator
4123			RLC		Rotate accumulator
4124			RLC		Rotate accumulator
4125			ADD	B	Add B to A

4126			INX	H	Increment H by 1
4127			MOV	M, A	Move content of A to M
4128			HLT		Stop the program

**OBSERVATION:**
**a. ASCII TO HEXADECIMAL**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4201		4301	

**b. HEXADECIMAL TO ASCII**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4201		4301	

**c. HEXADECIMAL TO BINARY**

INPUT		OUTPUT			
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4200		4200		4204	
		4201		4205	
		4202		4206	
		4203		4207	

**d. BINARY TO HEXADECIMAL**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4150		4152	
4151			

**e. HEXADECIMAL TO DECIMAL**

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4150		4152	
4151			

**RESULT:**

Thus the assembly language programs for various code conversions are executed using 8085 microprocessor.

**EX.No:5     INTERFACING A/D AND D/A CONVERTER WITH 8085****AIM:**

To write an assembly language program to convert an analog signal into a digital signal and a digital signal into an analog signal using an ADC interfacing and DAC interfacing respectively.

**a. ADC INTERFACING WITH 8085****APPARATUS REQUIRED:**

SL.NO	ITEM	SPECIFICATION	QUANTITY
1	Microprocessor kit	8085, Vi Microsystems	1
2	Power supply	+5 V dc	1
3	ADC Interface board	Vi Microsystems	1

**PROBLEM STATEMENT:**

To program starts from memory location 4100H. The program is executed for various values of analog voltage which are set with the help of a potentiometer. The LED display is verified with the digital value that is stored in the memory location 4150H.

**THEORY:**

An ADC usually has two additional control lines: the SOC input to tell the ADC when to start the conversion and the EOC output to announce when the conversion is complete. The following program initiates the conversion process, checks the EOC pin of ADC 0419 as to whether the conversion is over and then inputs the data to the processor. It also instructs the processor to store the converted digital data at RAM 4200H.

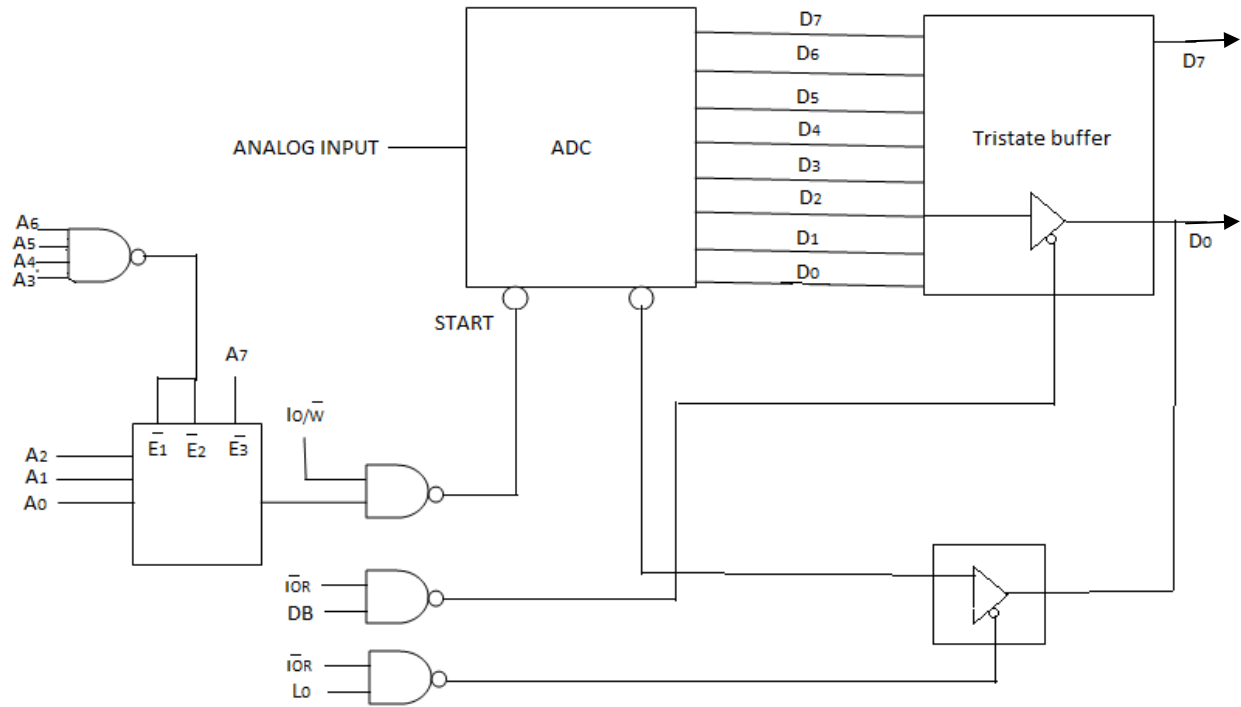
**ALGORITHM:**

1. Select the channel and latch the address.
2. Send the start conversion pulse.
3. Read EOC signal.
4. If EOC =1 continue else go to step (3)
5. Read the digital output.
6. Store it in a memory location.

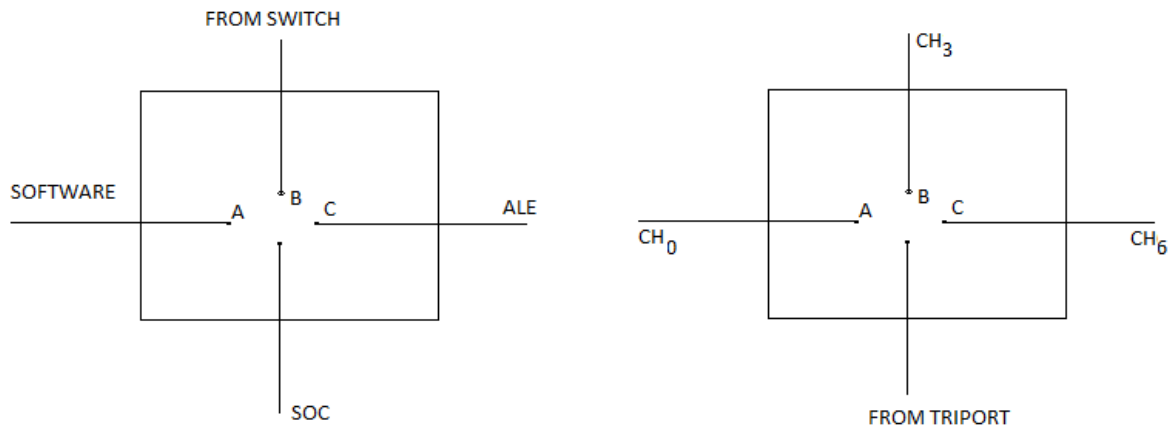
**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100			MVI	A, 10	Select channel 0 and to make accumulator low
4101					
4102			OUT	C8	Output the data
4103					
4104			MVI	A, 18	Make accumulator high
4105					
4106			OUT	C8	Display the data
4107					
4108			MVI	A, 01	Make 01 to accumulator
4109					
410A			OUT	D0	Display the data
410B					
410C			XRA		XOR with accumulator
410D			XRA		XOR with accumulator
410E			XRA		XOR with accumulator
410F			MVI	A, 00	Make 00 to accumulator
4110					
4111			OUT	D0	Load D0 in output port
4112					
4113		LOOP	IN	D8	
4114					
4115			ANI	01	Do and operation directly
4116					
4117			CPI	01	Compare with accumulator
4118					
4119			JNZ	LOOP	Jump to specified address
411A					
411B					
411C			IN	C0	
411D					
411E			STA	4150	Store the data
411F					
4120					
4121			HLT		End the program

### ADC- CIRCUIT:



### SOC JUMPER SELECTION:



J<sub>2</sub>: SOC Jumper selection

J<sub>5</sub>: Channel selection

**OBSERVATION**

<b>ANALOG VOLTAGE</b>	<b>DIGITAL DATA ON LED DISPLAY</b>	<b>HEX CODE IN LOCATION 4150</b>

**b. DAC INTERFACING WITH 8085****APPARATUS REQUIRED:**

SL.NO	ITEM	SPECIFICATION	QUANTITY
1	Microprocessor kit	8085, Vi Microsystems	1
2	Power supply	+5 V dc	1
3	DAC Interface board	Vi Microsystems	1

**SOFTWARE EXAMPLES**

The following examples illustrate how to control the DAC using 8085 and generate sine wave, saw tooth wave by means of software.

**(a) SQUARE WAVE GENERATION:**

The basic idea behind the generation of waveforms is the continuous generation of Analog output of DAC. With 00(HEX) as input to DAC2, the analog output is -5V. Similarly, with FF (Hex) as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC2, results in a square wave of amplitude 15 Volts

**ALGORITHM:**

1. Load the initial value (00) to Accumulator and move it to DAC.
2. Call the delay program
3. Load the final value (FF) to accumulator and move it to DAC.
4. Call the delay program.
5. Repeat steps 2 to 5.

**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	A, 00	Move 00 to A register
4101					
4102			OUT	C8	Load C8 to output port
4103					
4104			CALL	DELAY	Call delay program
4107			MVI	A, FF	Load FF to B register
4109			OUT	C8	
410B			CALL	DELAY	

410E			JMP START	START	Jump to start of address
4112		DELAY	MVI	B, 05	Move 05 to B register
4114		L1	MVI	C, FF	Move FF to C register
4116		L2	DCR	C	Decrement C
4117			JNZ	L2	Jump to L2 if no zero
411A			DCR	B	Decrement B register
411B			JNZ	L1	Jump to L1 if no zero
411E			RET		

Execute the program and using a CRO, verify that the waveform at the DAC2 output is a square-wave. Modify the frequency of the square-wave, by varying the time delay.

**(b) SAW TOOTH GENERATION:**

**ALGORITHM:**

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC.
3. Increment the accumulator content by 1.
4. Repeat steps 3 and 4.

Output digital data from 00 to FF constant steps of 01 to DAC1 repeat this sequence again and again. As a result a saw – tooth wave will be generated at DAC1 output.

**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	A, 00	Load 00 to accumulator
4102		L1	OUT	C0	Load CO in output port
4104			INR	A	Increment A register
4105			JNZ	L1	Jump to L1 if no zero
4108			JMP	START	Go to START unconditionally

**(c) TRIANGULAR WAVE GENERATION:**

**ALGORITHM:**

1. Load the initial value (00) to Accumulator.
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. If accumulator content is zero proceed to next step. Else go to step 3.
5. Load value (FF) to accumulator.
6. Move the accumulator content to DAC.

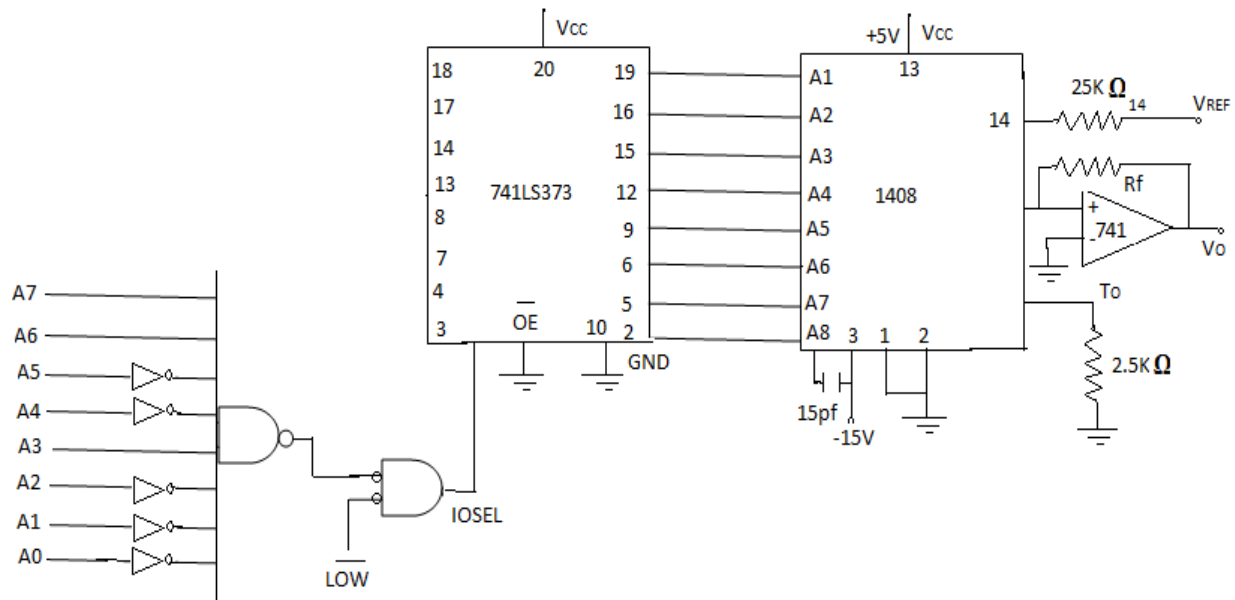
7. Decrement the accumulator content by 1.
8. If accumulator content is zero go to step 2. Else go to step 2.

The following program will generate a triangular wave at DAC2 output.

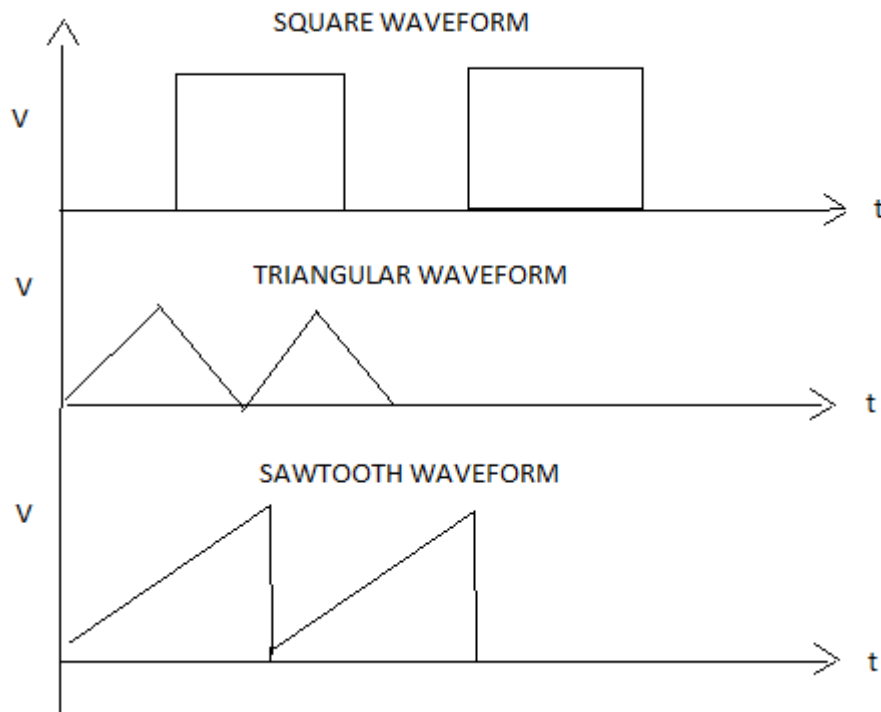
**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	L, 00	Move 00 to L register
4102		L1	MOV	A, L	Load L to a register
4103			OUT	C8	Load c8 to output port
4105			INR	L	Increment L register
4106			JNZ	L1	Jump to L1 if no zero
4109			MVI	L, FF	Load FF to L register
410B		L2	MOV	A, L	Move L to a register
410C			OUT	C8	Load C8 to output port
410E			DCR	L	Decrement L register
410F			JNZ	L2	Jump to L2 if no zero
4112			JMP	START	Go to START unconditionally

### DAC - CIRCUIT:



### WAEFORMS:



**OBSERVATION:**

WAVE FORMS	AMPLITUDE	TIME PERIOD
Square waveform		
Saw tooth waveform		
Triangular waveform		

**Result:**

Thus the conversion of an analog signal into a digital signal and a digital signal into an analog signal was done using interfacing of ADC and DAC respectively with 8085.

**EX.No:6      TRAFFIC LIGHT CONTROLLERS WITH 8085****AIM**

To write an assembly language program to simulate the traffic light at an intersection using a traffic light interface.

**APPARATUS REQUIRED:**

SL.NO	ITEM	SPECIFICATION	QUANTITY
1	Microprocessor kit	4185, Vi Microsystems	1
2	Power supply	+5 V dc	1
3	Traffic light interface kit	Vi Microsystems	1

**ALGORITHM:**

1. Initialize the ports.
2. Initialize the memory content, with some address to the data.
3. Read data for each sequence from the memory and display it through the ports.
4. After completing all the sequences, repeat from step2.

**A SAMPLE SEQUENCE:**

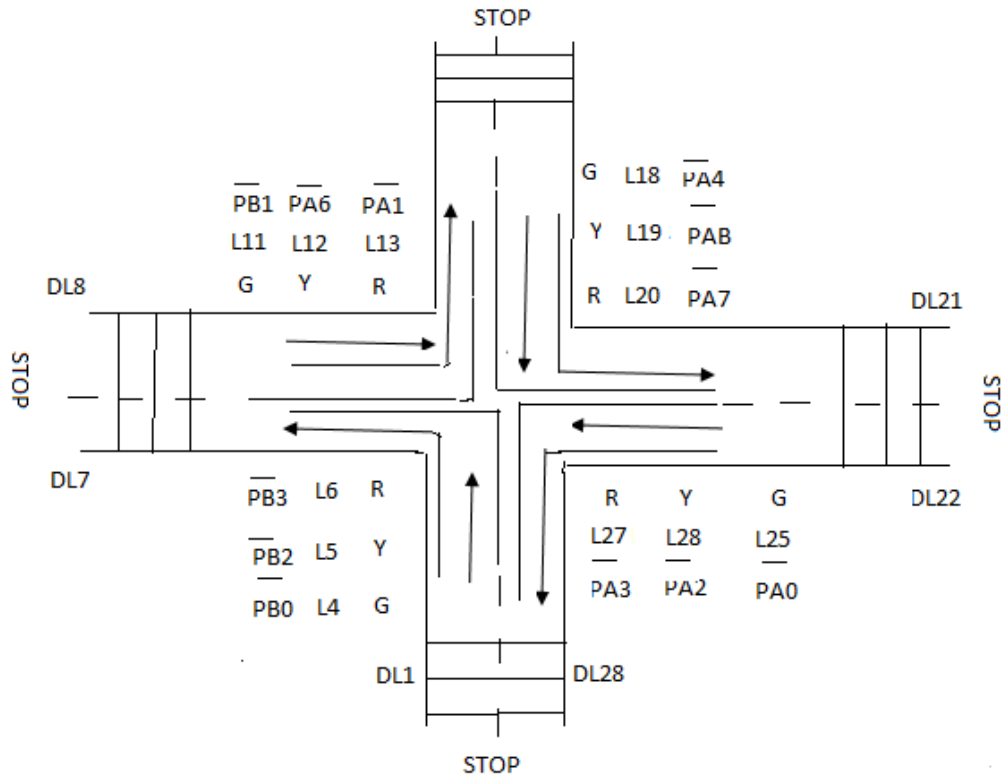
1. (a) Vehicles from south can go to straight or left.  
(b) Vehicles from west can cross the road.  
(c) Each pedestrian can cross the road.  
(d) Vehicles from east no movement.  
(e) Vehicles from north can go only straight.
2. All ambers are ON, indicating the change of sequence.
3. (a) Vehicles from east can go straight and left.  
(b) Vehicles from south can go only left.  
(c) North pedestrian can cross the road.  
(d) Vehicles from north, no movement.  
(e) Vehicles from west can go only straight.
4. All ambers are ON, indicating the change of sequence.
5. (a) Vehicles from north can go straight and left.  
(b) Vehicles from east can go only left.  
(c) West pedestrian can cross the road.  
(d) Vehicles from west, no movement.  
(e) Vehicles from south can go only straight.
6. All ambers are ON, indicating the change of sequence.

7. (a) Vehicles from west can go straight and left.  
(b) Vehicles from north can go only left.  
(c) South pedestrian can cross the road.  
(d) Vehicles from south, no movement.  
(e) Vehicles from east can go only straight.
8. All ambers are ON, indicating the change of sequence.
9. (a) All vehicles from all directions no movement.  
(b) All pedestrian can cross the road.

**BIT ALLOCATION:**

BIT	LED	BIT	LED	BIT	LED
PA0	SOUTH LEFT	PB0	NORTH LEFT	PC0	WEST STRAIGHT
PA1	SOUTH RIGHT	PB1	NORTH RIGHT	PC1	NORTH STRAIGHT
PA2	SOUTH AMBER	PB2	NORTH AMBER	PC2	EAST STRAIGHT
PA3	SOUTH RED	PB3	NORTH RED	PC3	SOUTH STRAIGHT
PA4	EAST LEFT	PB4	WEST LEFT	PC4	NORTH PD
PA5	EAST RIGHT	PB5	WEST RIGHT	PC5	WEST PD
PA6	EAST AMBER	PB6	WEST AMBER	PC6	SOUTH PD
PA7	EAST RED	PB7	WEST RED	PC7	EAST PD

**PATH REPRESENTATION:**



CONTROL ----- 0F (FOR 8255 PPI)

PORT A ----- 0C

PORT B ----- 0D

PORT C ----- 0E

**PROGRAM:**

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100	3E		MVI	A, 41	Move 41H immediately to accumulator
4102	D3		OUT	0F	Output contents of accumulator to 0F port
4104		REPEAT	LXI	H, DATA_	Load address 417B to HL register
				SQ	
4107	11		LXI	D, DATA_	Load address 4187 to DE register
410A	CD		CALL	OUT	Call out address 4142
410D	EB		XCHG		Exchange contents of HL with DE pair
410E	7E		MOV	A, M	Move M content to accumulator
410F	D3		OUT	0D	Load port A into output port
4111	CD		CALL	DELAY1	Call delay address
4114	EB		XCHG		Exchange content of HL with DE pair
4115	13		INX	D	Increment the content of D
4116	23		INX	H	Increment the content of H
4117	CD		CALL	OUT	Call out the address
411A	EB		XCHG		Exchange content of HL with DE pair
411B	7E		MOV	A, M	Move M content to accumulator
411C	D3		OUT	0D	Load port B into output port
411E	CD		CALL	DELAY1	Call DELAY address
4121	EB		XCHG		Exchange content of HL with DE pair
4122	13		INX	D	Increment D register
4123	23		INX	H	Increment H register
4124	CD		CALL	OUT	Call specified address
4127	EB		XCHG		Exchange content of HL with DE pair
4128	7E		MOV	A, M	Move M content to accumulator
4129	D3		OUT	0E	Load port C into output port

412B	CD		CALL	DELAY1	Call DELAY address
412E	EB		XCHG		Exchange content of HL with DE pair
412F	13		INX	D	Increment D register
4130	23		INX	H	Increment H register
4131	CD		CALL	OUT	Call specified address
4134	EB		XCHG		Exchange content of HL with DE pair
4135	7E		MOV	A, M	Move M content to accumulator
4136	D3		OUT	0E	Load port C into output port
4138	23		INX	H	Increment H register
4139	7E		MOV	A, M	Move M content to accumulator
413A	D3		OUT	0C	Load port A into output port
413C	CD		CALL	DELAY1	Call DELAY address
413F	C3		JMP	REPEAT	Jump to specified address
4142	7E	OUT	MOV	A, M	Move M content to accumulator
4143	D3		OUT	0E	Load port C into output port
4145	23		INX	H	Increment H register
4146	7E		MOV	A, M	Move M content to accumulator
4147	D3		OUT	0D	Load port B into output port
4149	23		INX	H	Increment H register
414A	7E		MOV	A, M	Move M content to accumulator
414B	D3		OUT	0C	Load port A into output port
414D	CD		CALL	DELAY	Call DELAY address
4150	C9		RET		Return to accumulator
4151	E5	DELAY	PUSH	H	Push the register H
4152	21		LXI	H,001F	Load 00 1F in HL register pair
4155	01		LXI	B,FFFF	Load FF FF in DE register pair
4158	0B		DCX	B	Decrement B register
4159	78		MOV	A, B	Move B content to accumulator
415A	B1		ORA	C	OR content of C with accumulator
415B	C2		JNZ	LOOP	Jump to LOOP if no zero
415E	2B		DCX	H	Decrement H register
415F	7D		MOV	A, L	Move L content to accumulator

4160	B4		ORA	H	OR content of H with accumulator
4161	C2		JNZ	L1	Jump to L1 if no zero
4164	E1		POP	H	Pop the register H
4165	C9		RET		Return from subroutine
4166	E5	DELAY1	PUSH	H	Push the register H
4167	21		LXI	H,001F	Load 00 1F in HL register pair
416A	01		LXI	B,FFFF	Load FF FF in DE register pair
416D	0B		DCX	B	Decrement B register
416E	78		MOV	A, B	Move B content to accumulator
416F	B1		ORA	C	OR content of C with accumulator
4170	C2		JNZ	LOOP2	Jump to LOOP2 if no zero
4173	2B		DCX	H	Decrement H register
4174	7D		MOV	A, L	Move L content to accumulator
4175	B4		ORA	H	OR content of H with accumulator
4176	C2		JNZ	L2	Jump to L2 if no zero
4179	E1		POP	H	Pop the register H
417A	C9		RET		Return to subroutine
417B	12 27 44 10 2B 92 10 9D 84 48 2E 84	DATA SEQ DB			
4187	48 4B 20 49 04	DATA_E			

### **RESULT:**

Thus an assembly language program to simulate the traffic light at an intersection using a traffic light interfaces was written and implemented.

**EX.No:7                      INTERFACING 8251 WITH 8085****AIM:**

To write a program to initiate 8251 and to check the transmission and reception of character.

**APPARATUS REQUIRED:**

1. 8085 Microprocessor kit
2. 8251 Interface board
3. DC regulated power supply

**THEORY:**

The 8251 is used as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and the converts them in a continuous serial data stream of transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the status of USART at any time. These include data transmissions errors and control signals.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a RESET operation. Control words should be written in to the control register of 8251. Words should be written in to the control register of 8251. These control words are split into two formats.

1. MODE INSTRUCTION WORD
2. COMMAND INSTRUCTION WORD.

**1. MODE INSTRUCTION WORD**

This format defines the BAUD rate, character length, parity and stop bits required to work with asynchronous data communication. By selecting the appropriate BAUD factor synchronous mode, the 8251 can be operated in synchronous mode.

Initializing 8251 using the Mode instructions to the following conditions.

8 bit data  
No parity  
Baud rate factor (16X)  
1 stop bit

Gives a mode command word of 01001110=4E(X)

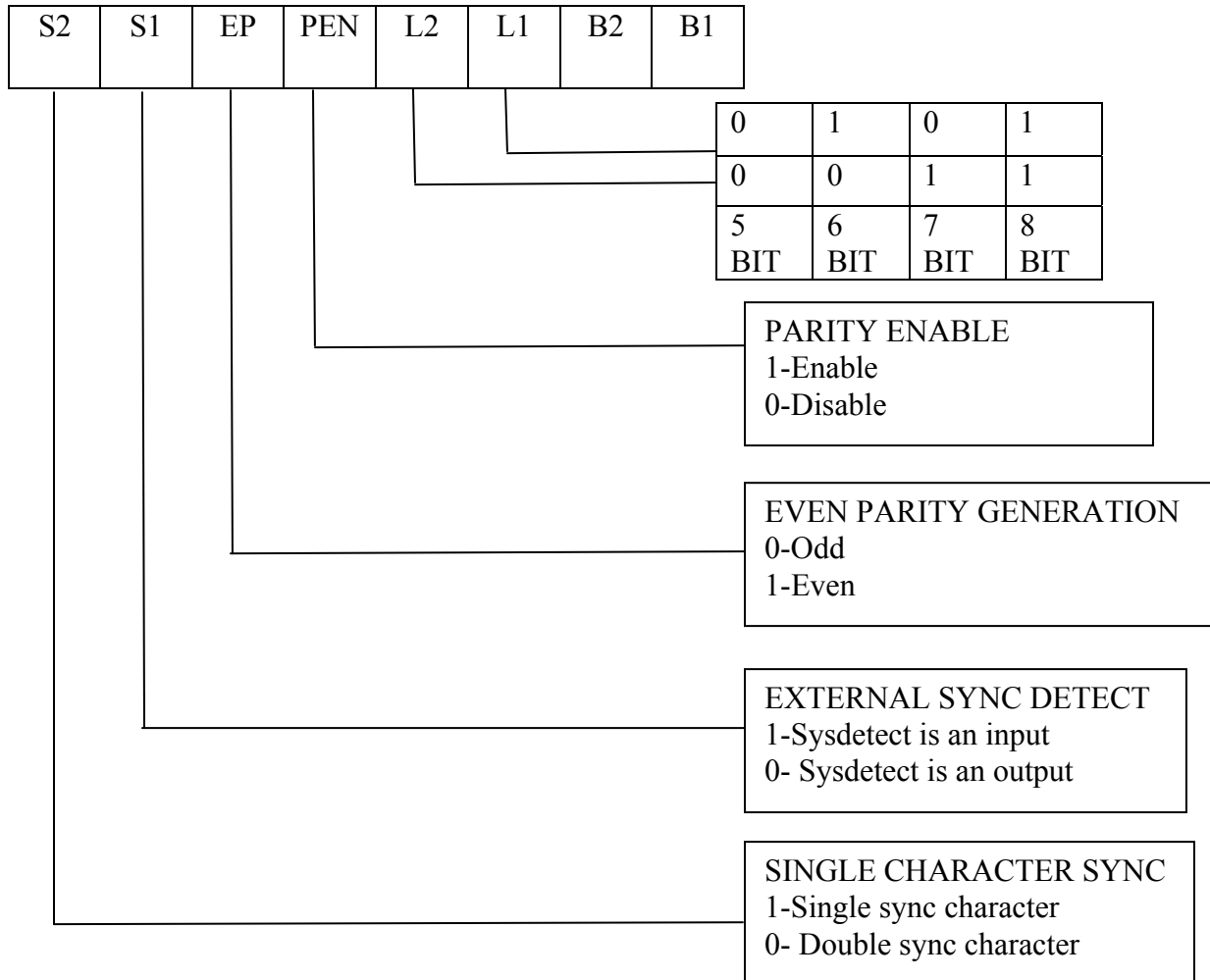
**ALGORITHM**

1. Initialize timer (8253) IC
2. Move the Mode command word (4EH) to A reg.
3. Output it port address C2
4. Move the command instruction word (37H) to A reg.
5. Output it to port address C2
6. Move the data to be transfer to A reg.
7. Output it to port address C0.
8. Reset the system
9. Get the data through input port address C0.
10. Store the value in memory
11. Reset the system

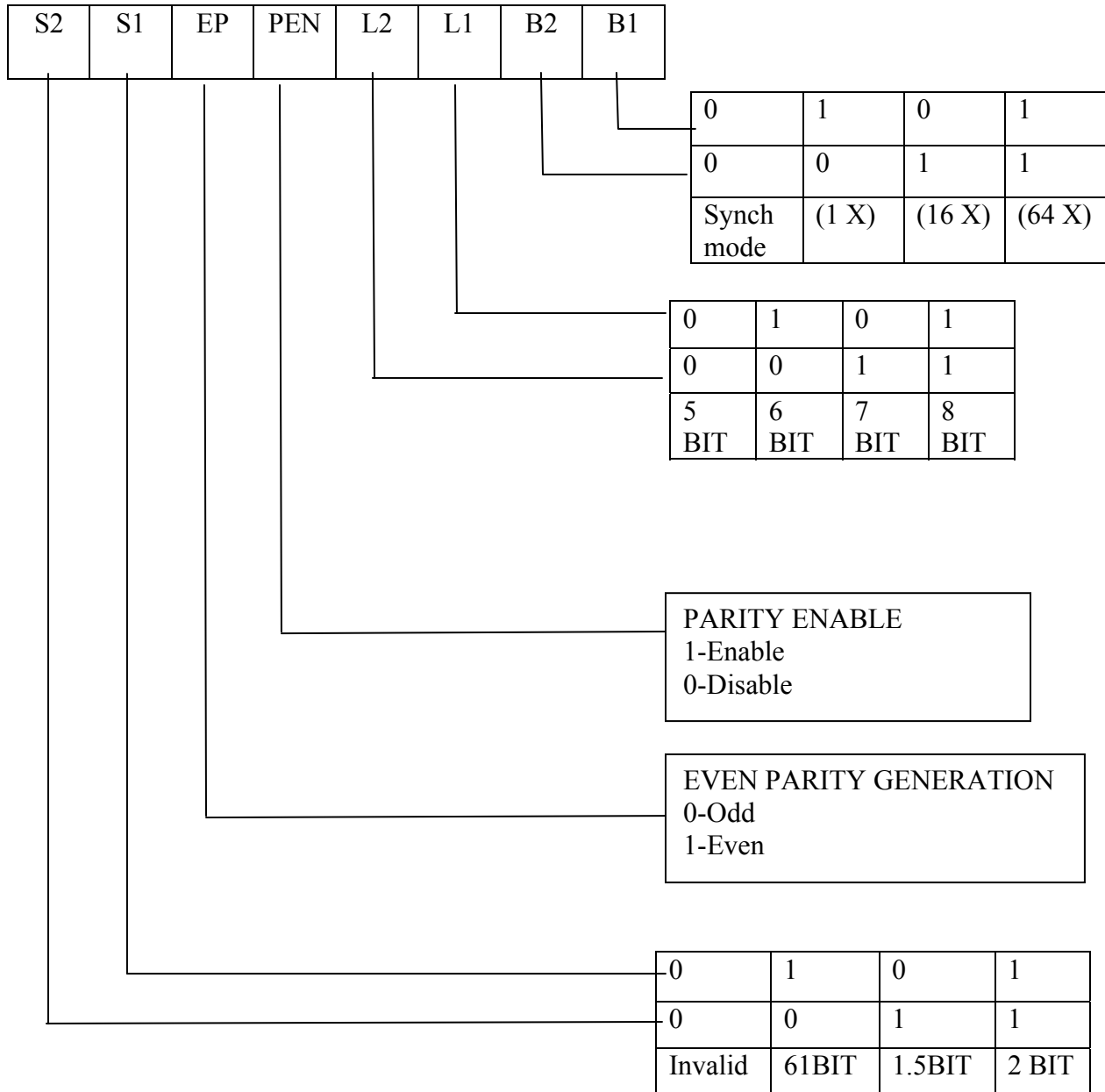
**PROGRAM:**

ADDRESS	OPC ODE	LAB EL	MNEM ONICS	OPERA ND	COMMENT
4100			MVI	A, 36	Move 36 to A
4102			OUT	CE	Output contents of accumulator to CE port
4104			MVI	A, 0A	Move 0A to accumulator
4106			OUT	C8	Output contents of accumulator to C8 port
4108			MVI	A, 00	Move 00 to accumulator
410A			OUT	C8	Output contents of accumulator to C8 port
410C			LXI	H, 4200	Store 4200 address in HL register pair
410F			MVI	A, 4E	Move 4E to accumulator
4111			OUT	C2	Output contents of accumulator to C2 port
4113			MVI	A, 37	Move 37 to accumulator
4115			OUT	C2	Output contents of accumulator to C2 port
4117			MVI	A, 41	Move 41 to accumulator
4119			OUT	C0	Output contents of accumulator to C0 port
411B			RST1		
4200			IN	C0	Input the contents from port C0 to accumulator
4202			STA	4150	Store the output from accumulator to 4150
4205			RST1		

**SYNCHRONOUS MODE:**



**ASYNCHRONOUS MODE:**



**OBSERVATION:**

MEMORY LOCATION	INPUT DATA	OUTPUT DATA

**RESULT:**

Thus the program to initiate 8251 was written and the transmission and reception of character was checked by interfacing 8251 with 8085.

**Ex No. 8****INTERFACING 8279 WITH 8085****AIM:**

To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor.

**APPARATUS REQUIRED:**

1. 8085 Microprocessor toolkit.
2. 8279 Interface board
3. Regulated D.C. power supply.

**PROGRAM:**

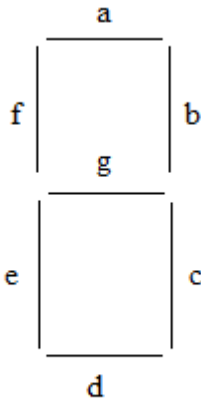
ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	LXI	H, 4130	Store the 16 bit address in HL pair
4103			MVI	D, 0F	Move 0F to D register
4105			MVI	A, 10	Move 10 to A
4107			OUT	C2	Output the contents of A to C2 output port
4109			MVI	A, CC	Move CC to A
410B			OUT	C2	Output the contents of A to C2 output port
410D			MVI	A, 90	Move 90 to A
410F			OUT	C2	Output the contents of A to C2 output port
4111		LOOP	MOV	A, M	Move content of M to A
4112			OUT	C0	Output the contents of M to A
4114			CALL	DELAY	Call the delay address
4117			INX	H	Increment H register
4118			DCR	D	Decrement D register
4119			JNZ	LOOP	Jump to specified address

411C			JMP START	START	Jump to START address
411F		DELAY	MVI	B, A0	Move a to B register
4121		LOOP1	MVI	C, FF	Move FF to C register
4123		LOOP2	DCR	C	Decrement C register
4124			JNZ	LOOP 1	Jump to LOOP 1 if no zero
4127			DCR	B	Decrement B register
4128			JNZ	LOOP 2	Jump to LOOP 2 if no zero
412B			RET		

Pointer equal to 4130 .FF repeated eight times

4130	FF
4131	FF
4132	FF
4133	FF
4134	FF
4135	FF
4136	FF
4137	FF
4138	98
4139	68
413 <sup>a</sup>	7C
413B	C8
413C	1C
413D	29
413E	FF
413F	FF

### **SEGMENT DEFINITION:**



DATA BUS	D7	D6	D5	D4	D3	D2	D1	D0
SEGMENTS	d	c	b	a	dp	g	f	e

**OBSERVATION:**

LETTER	7 SEGMENT	DATA BUS								HEXADECIMAL
		D7	D6	D5	D4	D3	D2	D1	D0	

**RESULT:**

Thus 8279 controller was interfaced with 8085 and program for rolling display was executed successfully.

# **MICROCONTROLLER**

**Ex.No: 9**

**8051 – BASIC SIMPLE PROGRAM**

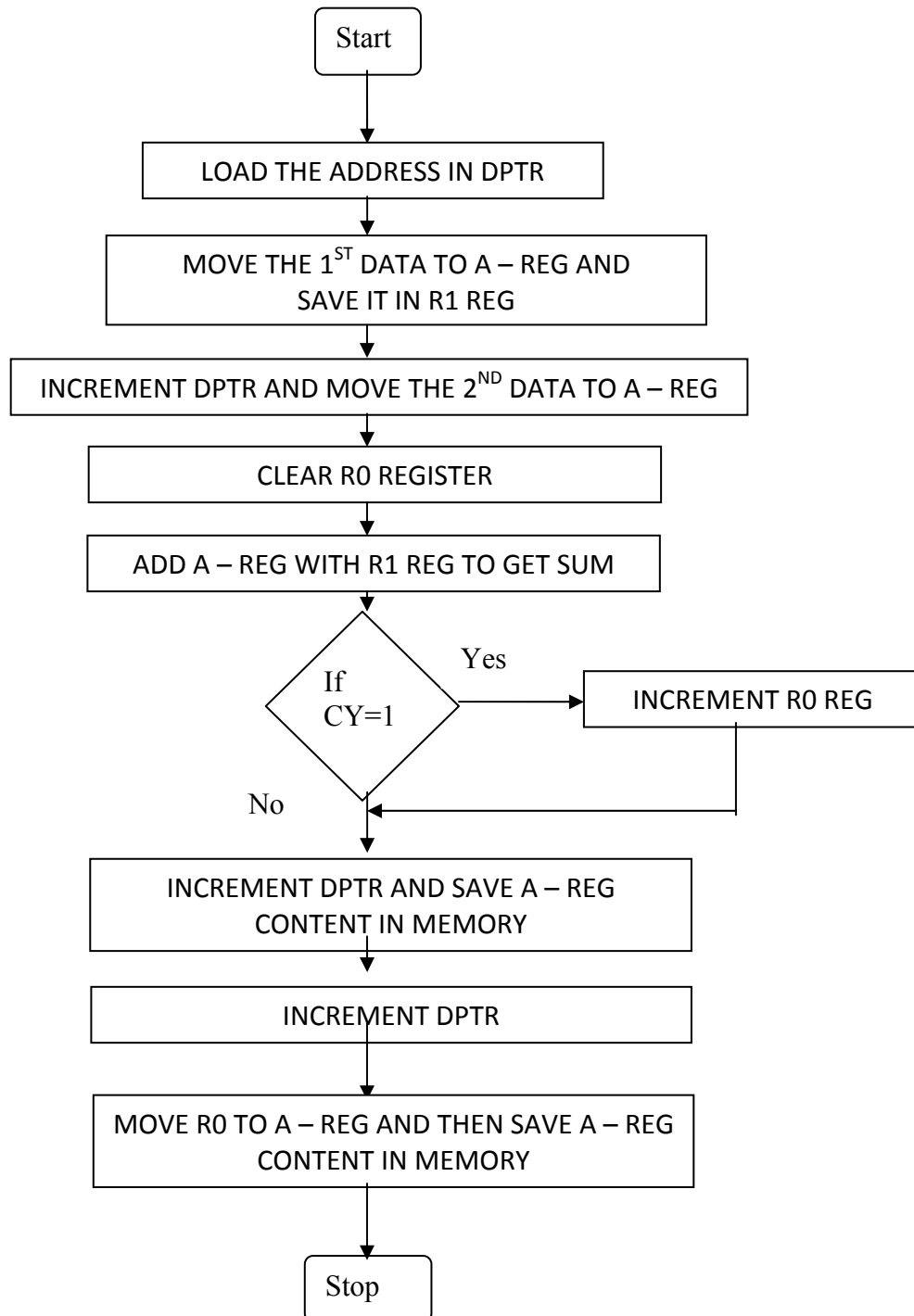
**AIM:**

To write an assembly language program to add, subtract, multiply and divide the given data stored at two consecutive locations using 8051 microcontroller.

**a. ADDITION OF TWO 8 BIT NUMBERS**

**Algorithm:**

1. Set DPTR as pointer for data.
2. Move first data from external memory to accumulator and save it in R1 register.
3. Increment DPTR.
4. Move second data from external memory to accumulator
5. Clear R0 register to account for carry.
6. Add the content of R1 register to accumulator.
7. Check for carry. If carry is not set go to step 8. Otherwise go to next step.
8. Increment R0 register.
9. Increment DPTR and save the sum in external memory.
10. Increment DPTR, move carry to accumulator and save it in external memory.
11. Stop



Label	Program	Comments
	MOV DPTR,#4500	Load address of 1 <sup>st</sup> data in DPTR
	MOVX A,@DPTR	Move the 1 <sup>st</sup> data to A
	MOV R1,A	Save the first data in R1
	INC DPTR	Increment DPTR to point 2 <sup>nd</sup> data
	MOVX A,@DPTR	Load the 2 <sup>nd</sup> data in A
	MOV R0,#00	Clear R0 for the account of carry
	ADD A,R1	Get the sum in A reg
	JNC AHEAD	Check carry flag
	INC R0	If carry is set increment R0
AHEAD:	INC DPTR	Increment DPTR
	MOVX A,@DPTR	Save the sum in external memory
	INC DPTR	Increment DPTR
	MOV A,R0	Move carry to A reg
	MOVX A,@DPTR	Save the carry in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

**Observation:**

Input:

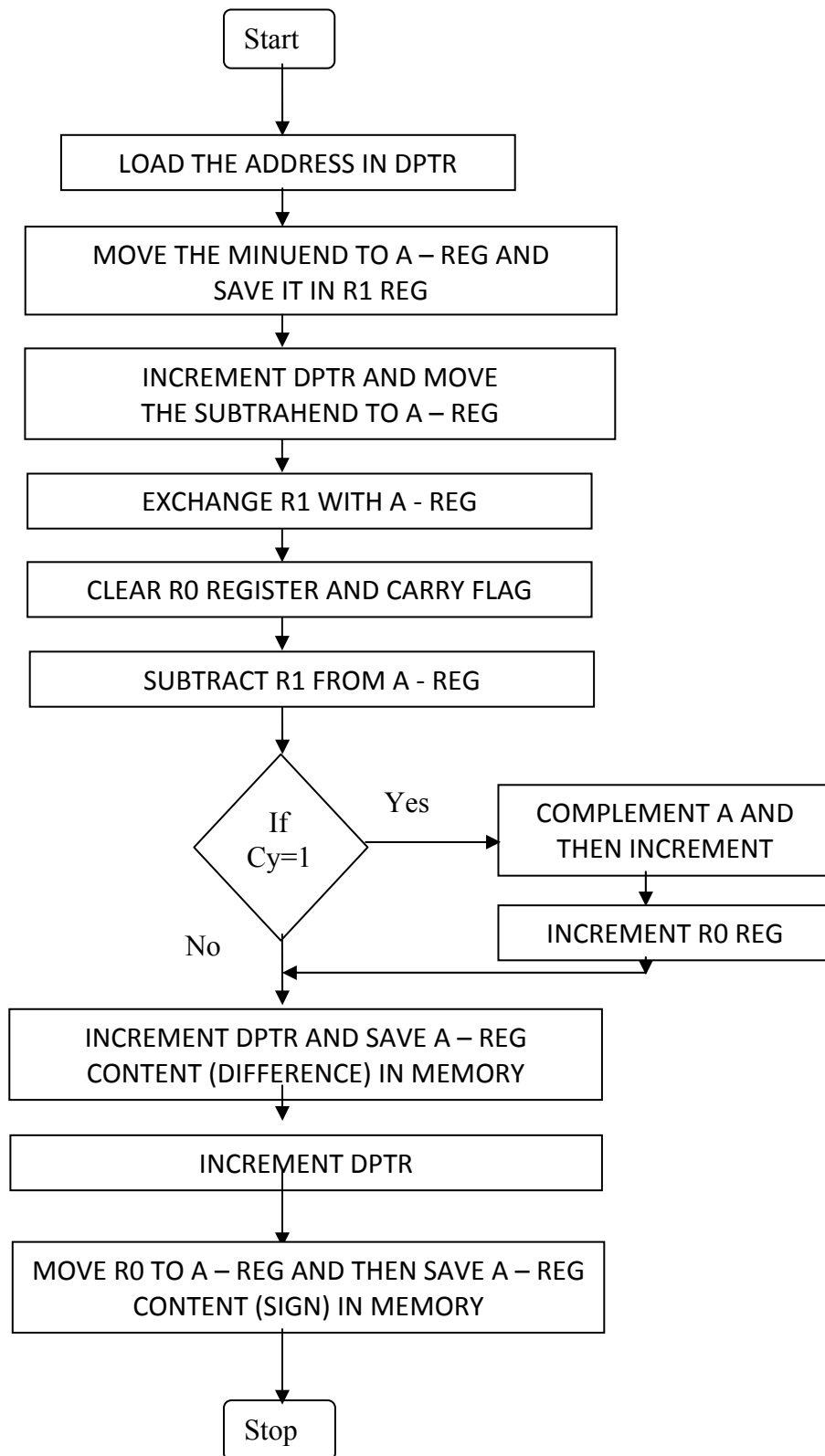
4500: [Addend]

4501: [Augend]

Output:

4502: [Sum]

4503: [Carry]



## **b. SUBTRACTION OF TWO 8 BIT NUMBERS**

### **Algorithm:**

1. Set DPTR as pointer for data.
2. Move the minuend from external memory to accumulator and save it in R1 register.
3. Increment DPTR.
4. Move subtrahend from external memory to accumulator
5. Exchange the contents of R1 and A such that minuend is in A and subtrahend is in R1
6. Clear R0 register to account for sign.
7. Clear carry flag.
8. Subtract the content of R1 register from accumulator.
9. Check for carry. If carry is not set go to step 12. Otherwise go to next step.
10. Complement the content of A – reg and increment by 1 to get 2's complement of result in A – reg
11. Increment R0 register.
12. Increment DPTR and save the result in external memory.
13. Increment DPTR, move R0 (sign bit) to accumulator and then save it in external memory.
14. Stop

Label	Program	Comments
	MOV DPTR,#4500	Load address of minuend in DPTR
	MOVX A,@DPTR	Move the minuend to A
	MOV R1,A	Save the minuend in R1
	INC DPTR	Increment DPTR to point subtrahend
	MOVX A,@DPTR	Load the subtrahend in A
	XCH A,R1	Get minuend in A and Subtrahend in R1
	MOV R0,#00	Clear R0 for the account of Sign
	CLR C	Clear carry
	SUBB A,R1	Subtract R1 from A
	JNC AHEAD	Check Carry flag. If carry is set then
	CPL A	Get 2's complement of result in A
	INC A	
	INC R0	Set R0 to indicate negative sign
	INC DPTR	Increment DPTR
AHEAD:	MOVX A,@DPTR	Save the result in external memory
	INC DPTR	Increment DPTR
	MOV A,R0	Move sign bit to A reg
	MOVX A,@DPTR	Save the sign in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

**Observation:**

Input:

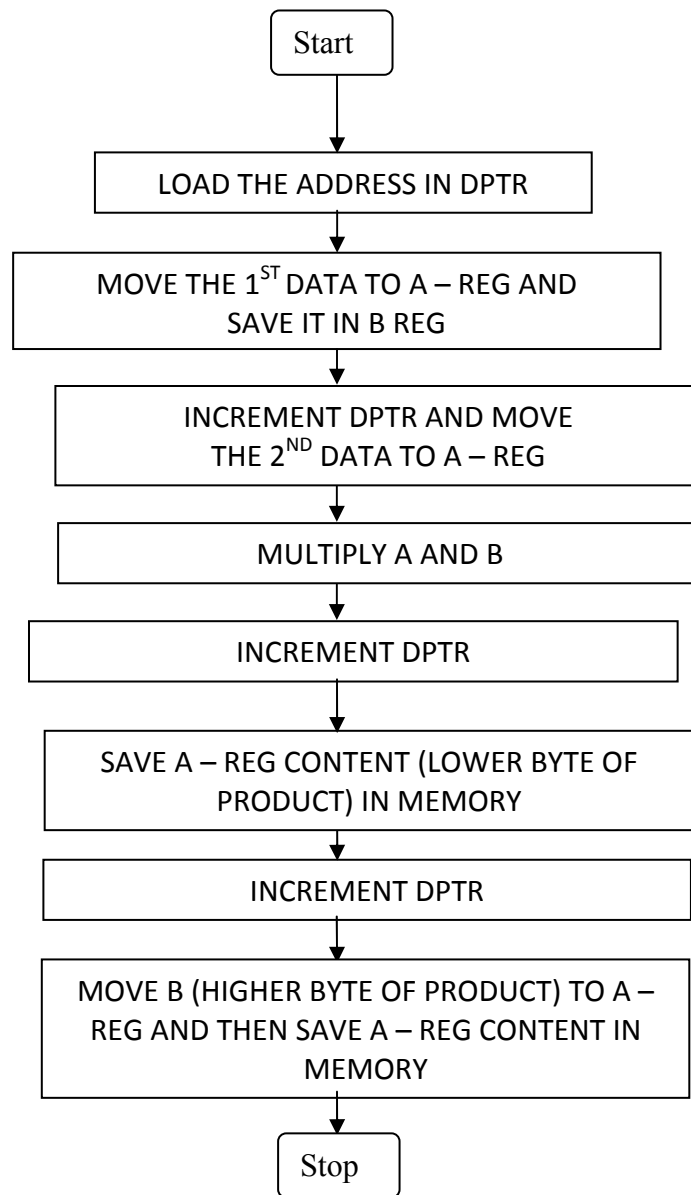
4500: [Minuend]

4501: [Subtrahend]

Output:

4502: [Difference]

4503: [Sign Bit]



### c. MULTIPLICATION OF TWO 8 BIT NUMBERS

#### Algorithm:

1. Load address of data in DPTR
2. Move the first data from external memory to A and save in B.
3. Increment DPTR and move second data from external memory to B.
4. Perform multiplication to get the product in A and B.
5. Increment DPTR and save A ( lower byte of product) in memory
6. Increment DPTR , move B ( lower byte of product) to A and save it in memory
7. Stop

Label	Program	Comments
	MOV DPTR,#4500	Load address of 1 <sup>st</sup> data in DPTR
	MOVX A,@DPTR	Move the 1 <sup>st</sup> data to A
	MOV B,A	Save the 1 <sup>st</sup> data in B
	INC DPTR	Increment DPTR to point 2 <sup>nd</sup> data
	MOVX A,@DPTR	Load the 2 <sup>nd</sup> data in A
	MUL AB	Get the product in A and B
	INC DPTR	Increment DPTR
	MOVX A,@DPTR	Save the lower byte of result in external memory
	INC DPTR	Increment DPTR
	MOV A,B	Move the higher byte of product to A reg
	MOVX A,@DPTR	Save it in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

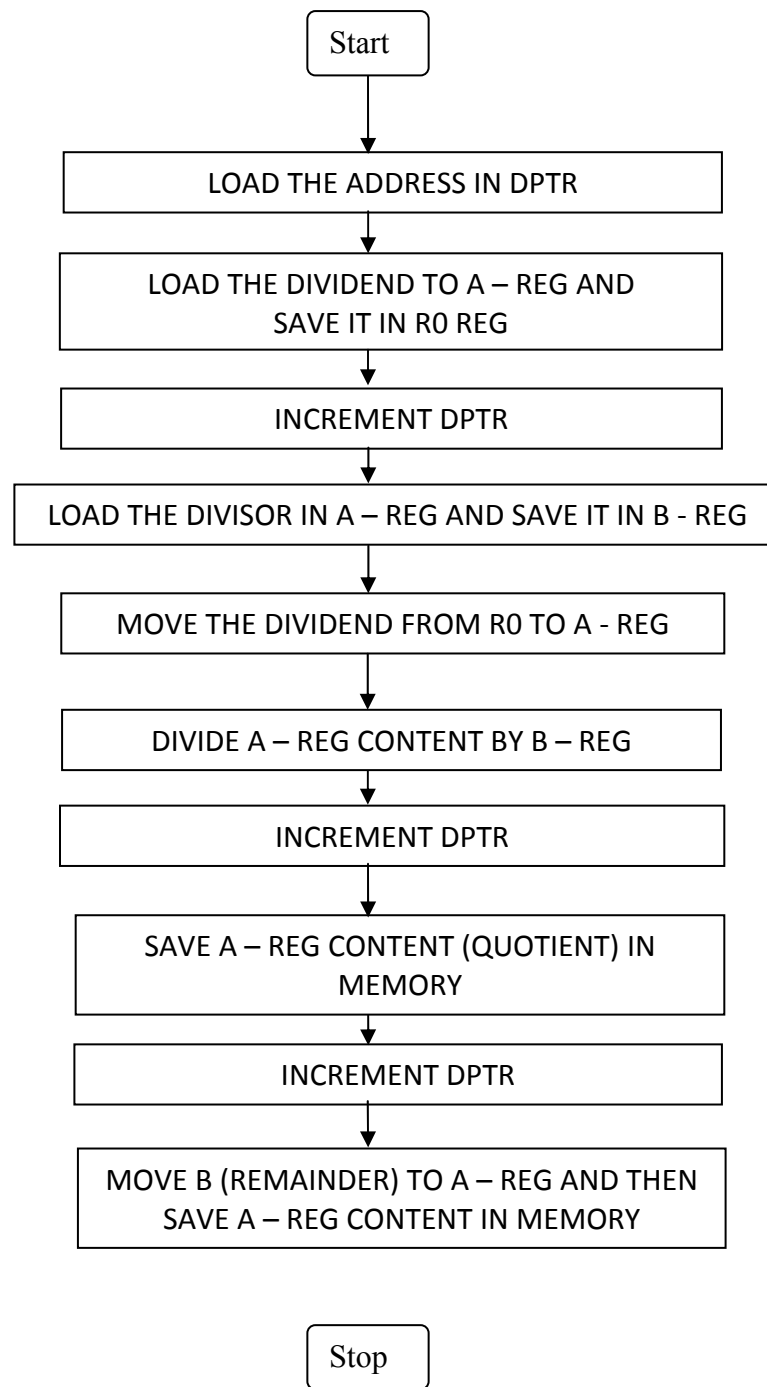
Observation

Input:

4500: [1<sup>st</sup> data]  
4501: [2<sup>nd</sup> data]

Output:

4502: [Lower byte of product]  
4503: [Higher byte of product]



#### d. DIVISION OF TWO 8 BIT NUMBERS

##### Algorithm:

1. Load address of data in DPTR
2. Move the dividend from external memory to A and save it in R0 register.
3. Increment DPTR and move the divisor from external memory to A and save it in B reg.
4. Move the dividend from R0 to A.
5. Perform division to get quotient in A and remainder in B.
6. Increment DPTR and save quotient (content of A - reg) in memory
7. Increment DPTR.
8. Move the remainder (Content of B – reg) to A and save in memory.
9. Stop

Label	Program	Comments
	MOV DPTR,#4500	Load address of dividend in DPTR
	MOVX A,@DPTR	Move the dividend to A
	MOV R0,A	Save the dividend in R0
	INC DPTR	Increment DPTR to point divisor
	MOVX A,@DPTR	Load the divisor in A
	MOV B,A	Move the divisor to B
	MOV A,R0	Move the dividend to A
	DIV AB	Divide the content of A by B
	INC DPTR	Increment DPTR
	MOVX A,@DPTR	Save the quotient in external memory
	INC DPTR	Increment DPTR
	MOV A,B	Move the remainder to A reg
	MOVX A,@DPTR	Save it in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

**Observation:**

Input:

4500: [Dividend]  
4501: [Divisor]

Output:

4502: [Quotient]  
4503: [Remainder]

**Result:**

Thus the addition, subtraction, multiplication and division of two numbers were performed using the 8051 microcontroller.

**Ex N0. 10a**

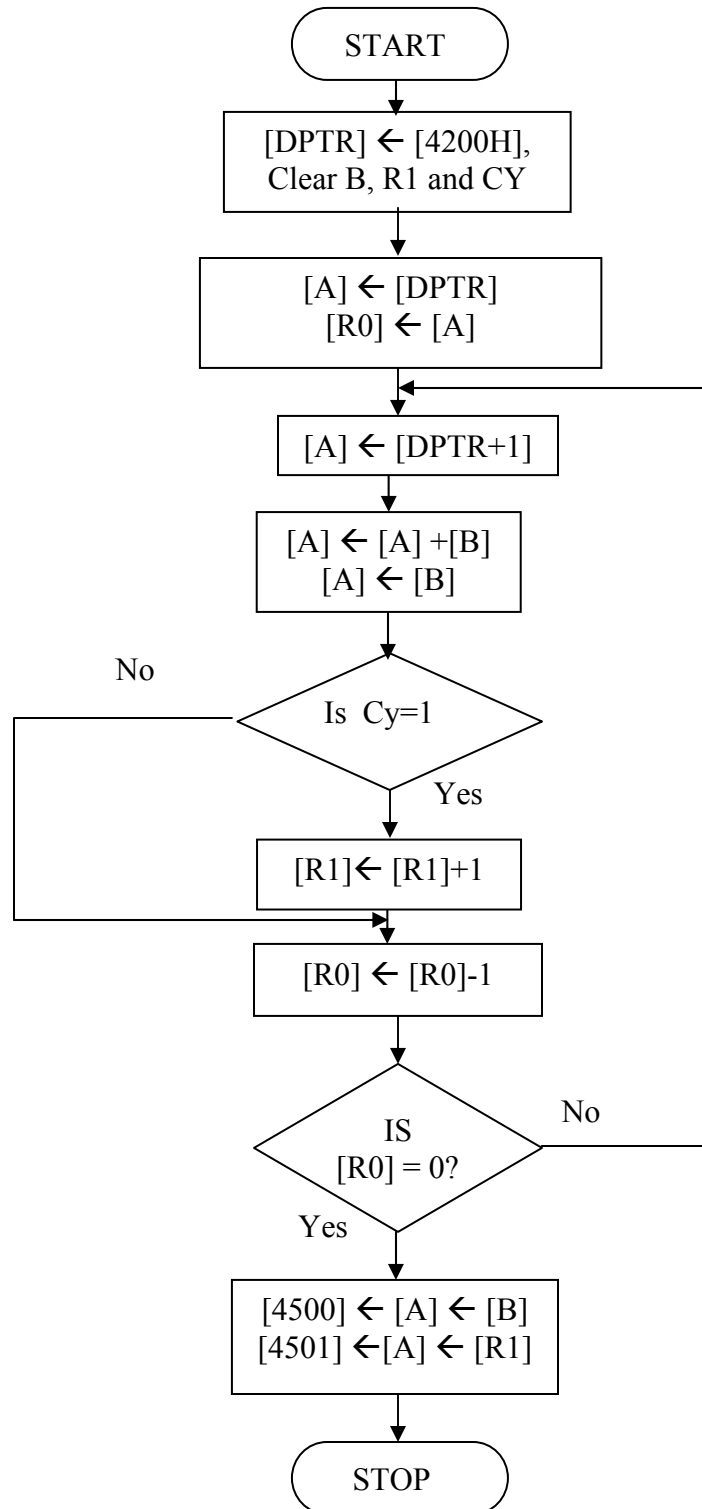
**SUM OF ELEMENTS IN AN ARRAY**

**AIM:**

To find the sum of elements in an array.

**ALGORITHM:**

1. Load the array in the consecutive memory location and initialize the memory pointer with the starting address.
2. Load the total number of elements in a separate register as a counter.
3. Clear the accumulator.
4. Load the other register with the value of the memory pointer.
5. Add the register with the accumulator.
6. Check for carry, if exist, increment the carry register by 1. otherwise, continue
7. Decrement the counter and if it reaches 0, stop. Otherwise increment the memory pointer by 1 and go to step 4.



**PROGRAM:**

Label	Program	Comments
AGAIN	MOV DPTR, #4200	Load 4200 to DPTR, get array size
	MOVX A, @DPTR	Copy array size value to A
	MOV R0, A	Move contents of A to R0
	MOV B, #00	
	MOV R1, B	
	CLR C	Clear B, R1 & Carry Flag
	INC DPTR	Get value from data pointer
	MOVX A, @DPTR	
	ADD A, B	Sum of two numbers
	MOV B, A	
NC	JNC NC	If no carry jump to NC
	INC R1	Else increment R1
	DJNZ R0, AGAIN	Decrement size of array, if not zero fetch data from DPTR and add to the resultant value
	MOV DPTR, #4500	Store the result in 4500 memory locations
HERE	MOV A, B	
	MOVX @DPTR, A	
	INC DPTR	
	MOV A, R1	
	MOVX @DPTR, A	
	SJMP HERE	

**OBSERVATION:**

INPUT		OUTPUT	
4200		4500	
4201			
4202		4501	
4203			

**RESULT:**

The sum of elements in an array is calculated.

**Exp. No 10(b)**

**8051 – Check Odd or Even using Call Option**

**AIM:**

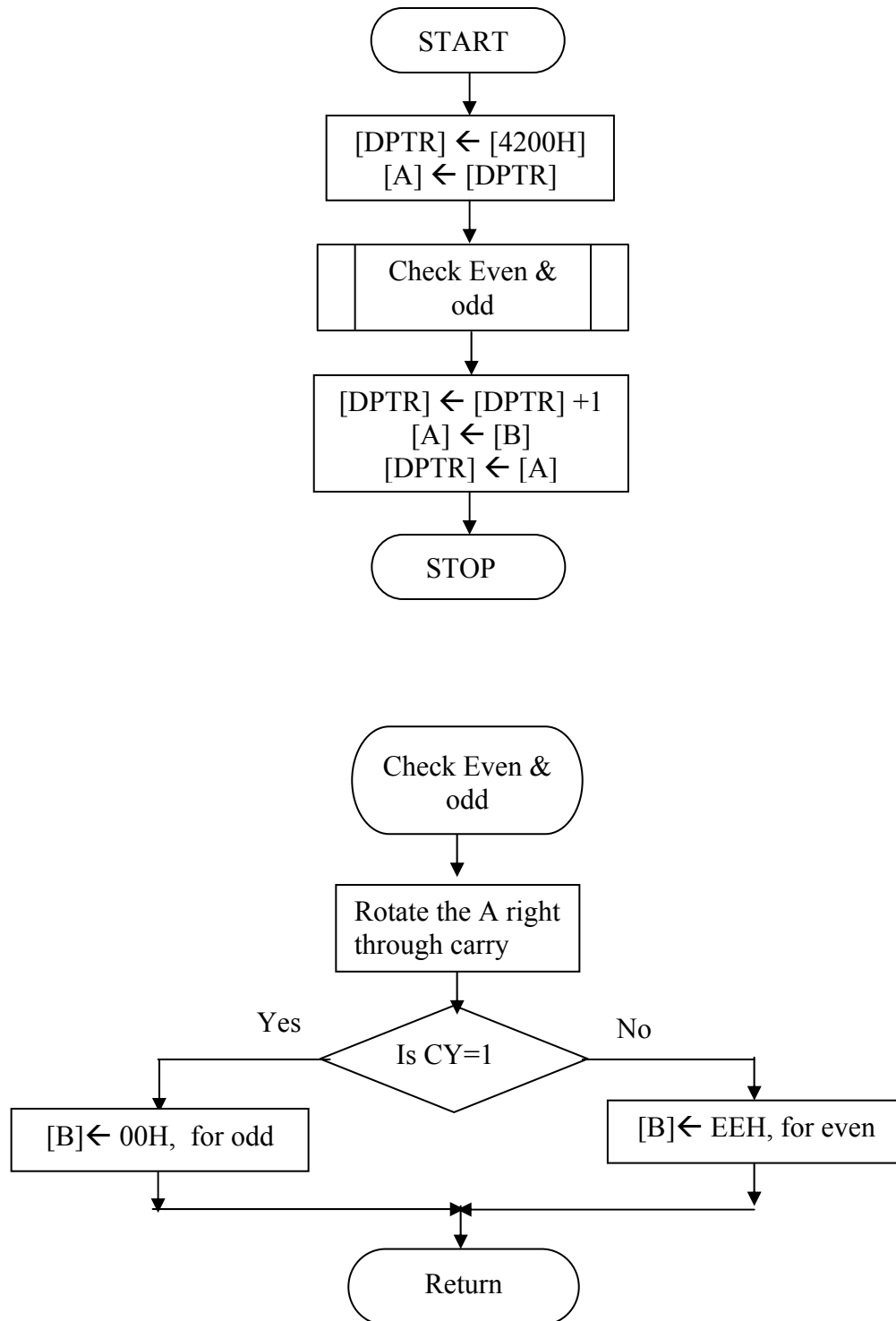
To Check whether given number is Odd or Even using Call Option.

**ALGORITHM:**

1. Start
2. Move the data to DPTR
3. Move the data to accumulator
4. Call check function whether given number is odd or even
5. Increment DPTR
6. Resultant value stored in DPTR-4201
7. Halt

Check Function

8. Rotate right through carry
9. If carry copy 00 to B reg (odd number)
10. Else copy EE to B reg (even number)
11. return



**PROGRAM:**

Label	Program	Comments
HERE	MOV DPTR, #4200	Load 4200 to DPTR, get a number
	MOVX A, @DPTR	Copy value to A
	CALL CHECK_EV_OD	Call check function whether given number is odd or even
	INC DPTR	
	MOV A,B	
	MOVX @DPTR, A	Resultant value stored in DPTR-4201
CHECK_EV_OD	SJMP HERE	
	RRC A	Rotate right through carry
	JC L1	If carry copy 00 to B reg (odd number)
L1	MOV B,#0EEH	Else copy EE to B reg (even number)
	MOV B,#00H	
	RET	

**OBSERVATION:**

INPUT		OUTPUT	
4200		4201	

**RESULT:**

Thus the given number is either even or odd has been found using 8051 microcontroller.

## **11 INTERFACING A/D & D/A CONVERTER WITH 8051**

### **a. ADC INTERFACING WITH 8051**

#### **APPARATUS REQUIRED:**

SL.NO	ITEM	SPECIFICATION	QUANTITY
1	Microcontroller kit	8051, Vi Microsystems	1
2	Power supply	+5 V dc	1
3	ADC Interface board	Vi Microsystems	1

#### **PROBLEM STATEMENT:**

To program starts from memory location 4100H. The program is executed for various values of analog voltage which are set with the help of a potentiometer. The LED display is verified with the digital value that is stored in the memory location 4150H.

#### **THEORY:**

An ADC usually has two additional control lines: the SOC input to tell the ADC when to start the conversion and the EOC output to announce when the conversion is complete. The following program initiates the conversion process, checks the EOC pin of ADC 0419 as to whether the conversion is over and then inputs the data to the processor. It also instructs the processor to store the converted digital data at RAM 4200H.

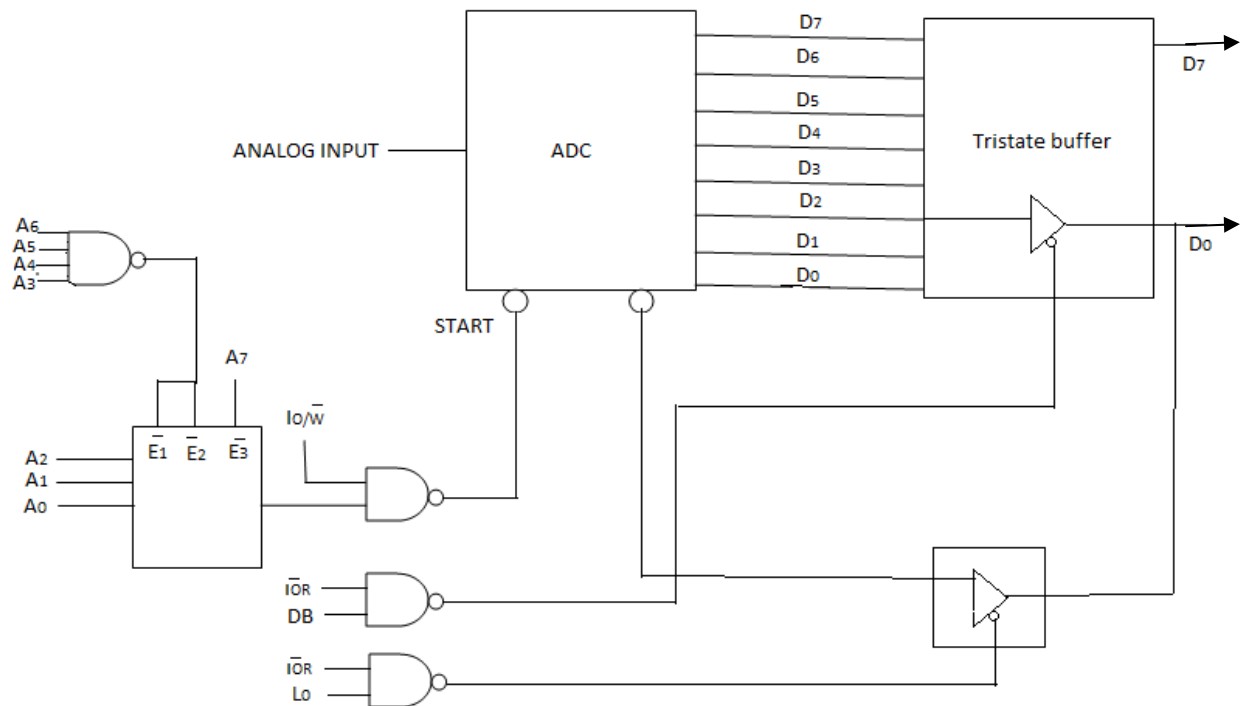
#### **ALGORITHM:**

1. Select the channel and latch the address.
2. Send the start conversion pulse.
3. Read EOC signal.
4. If EOC =1 continue else go to step (3)
5. Read the digital output.
6. Store it in a memory location.

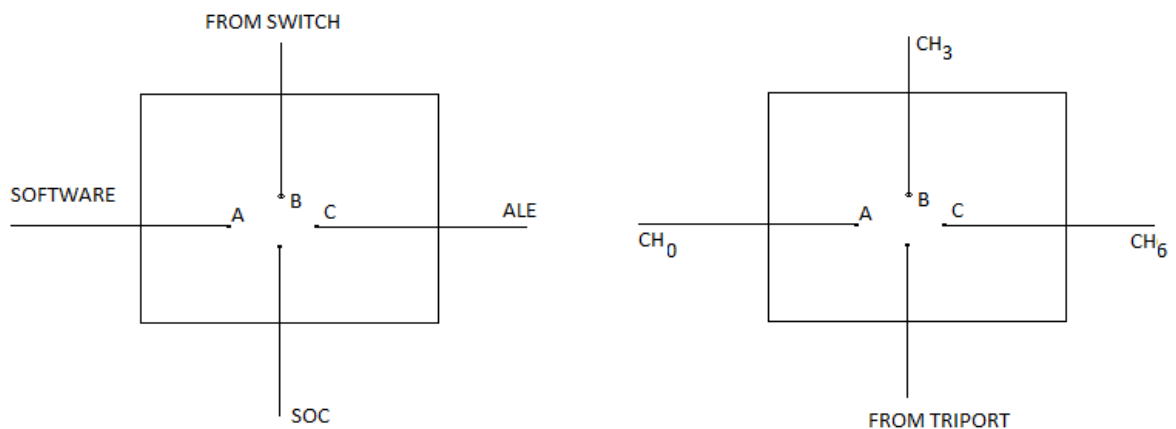
**PROGRAM:**

Label	Program	Comments
WAIT	MOV DPTR, #FFC8	Select Channel 0 and make ALE Low
	MOV A, #10	
	MOVX @DPTR, A	make ALE High
	MOV A, #18	
	MOVX @DPTR, A	SOC signal High
	MOV DPTR, #FFD0	
	MOV A, #01	SOC signal low
	MOVX @DPTR, A	
	MOV A, #00	
	MOVX @DPTR, A	
	MOV DPTR, #FFD8	
	MOVX A, @DPTR	
	JNB E0, WAIT	Check for EOC
	MOV DPTR, #FFC0	Read ADC data
	MOVX A, @DPTR	Store the data in memory location
	MOV DPTR, #4150	
HERE	MOVX @DPTR, A	
	SJMP HERE	

### ADC- CIRCUIT:



### SOC JUMPER SELECTION:



J<sub>2</sub>: SOC Jumper selection

J<sub>5</sub>: Channel selection

**OBSERVATION**

ANALOG VOLTAGE	DIGITAL DATA ON LED DISPLAY	HEX CODE IN LOCATION 4150

**(b) INTERFACING D/A CONVERTER WITH 8051****APPARATUS REQUIRED:**

SL.NO	ITEM	SPECIFICATION	QUANTITY
1	Microprocessor kit	4185, Vi Microsystems	1
2	Power supply	+5 V dc	1
3	DAC Interface board	Vi Microsystems	1

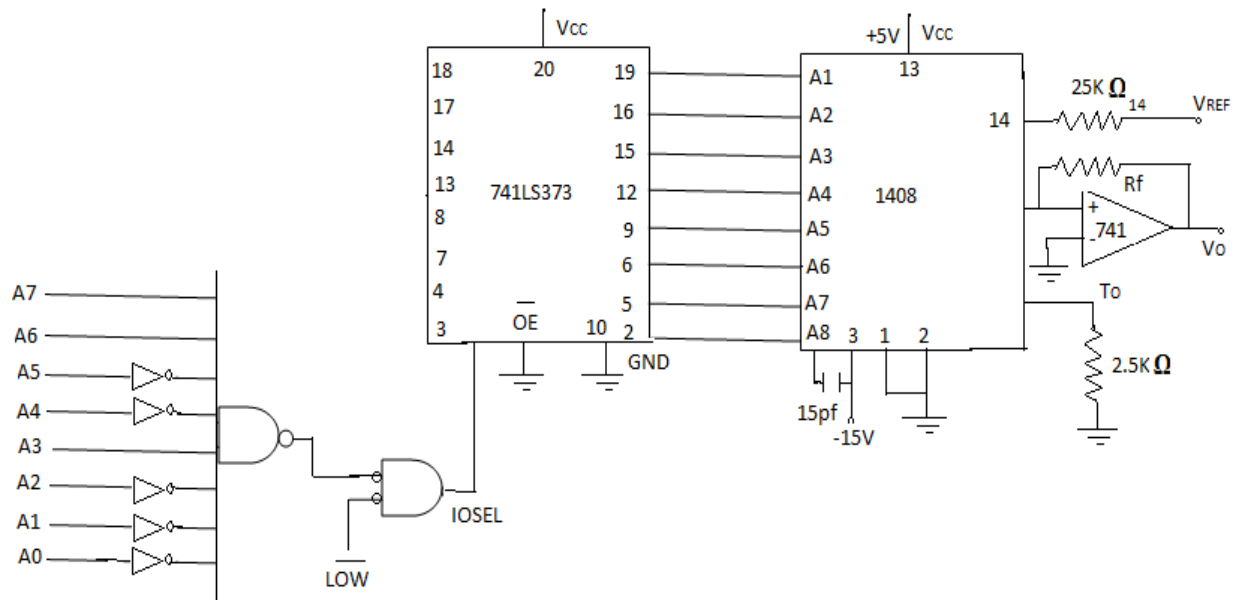
**THEORY:****SOFTWARE EXAMPLES**

After going through the software examples you can learn how to control the DAC using 8051 and generate sine wave, saw tooth wave etc by means of software.

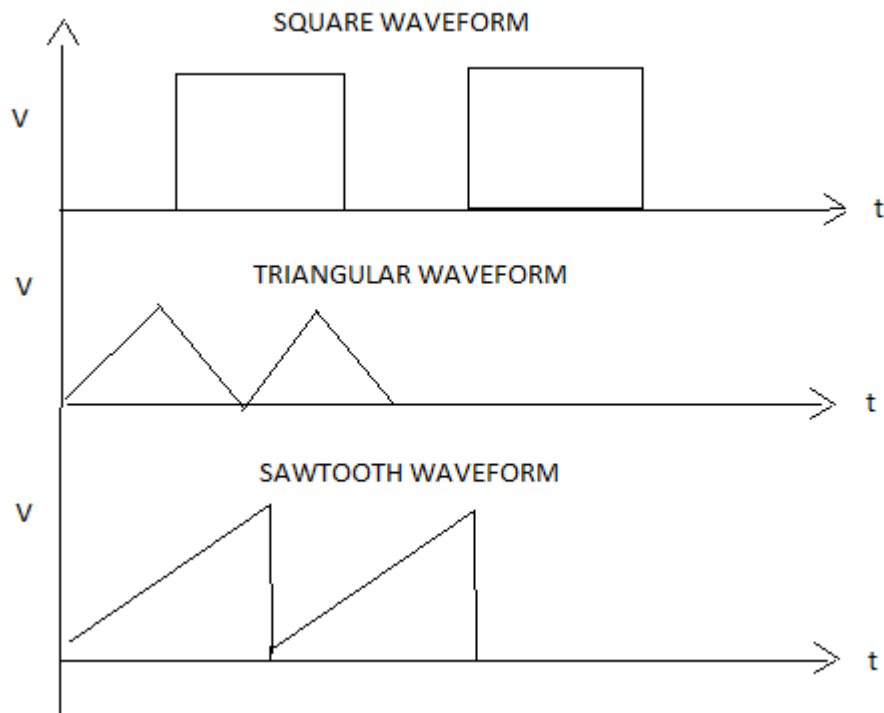
**ALGORITHM:****(a) SQUARE WAVE GENERATION:**

1. Load the initial value (00) to Accumulator and move it to DAC.
2. Call the delay program
3. Load the final value (FF) to accumulator and move it to DAC.
4. Call the delay program.
5. Repeat steps 2 to 5.

### DAC - CIRCUIT:



### WAVEFORMS:



### **OBSERVATION:**

WAVE FORMS	AMPLITUDE	TIME PERIOD
Square waveform		
Saw tooth waveform		
Triangular waveform		

### **PROGRAM:**

The basic idea behind the generation of waveforms is the continuous generation of Analog output of DAC.

With 00(HEX) as input to DAC2, the analog output is -5V. Similarly, with FF (Hex) as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC2, results in a square wave of amplitude 15 Volts.

ADDRESS	LABEL	MNEMONICS	OPCODE	OPERAND	COMMENT
		MOV DPTR,#FFC8			
	START	MOV A,#00			
		MOVX @DPTR,A			
		LCALL DELAY			
		MOV A,# FF			
		MOVX @DPTR,A			
		LCALL DELAY			
		LJMP START			
	DELAY	MOV R1,#05			
	LOOP	MOV R2,#FF			
		DJNZ R2,HERE			
		DJNZ R1,LOOP			
		RET			
		SJMP START			

Execute the program and using a CRO, verify that the waveform at the DAC2 output is a square-wave. Modify the frequency of the square-wave, by varying the time delay.

### **(b) SAW TOOTH GENERATION**

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC.
3. Increment the accumulator content by 1.
4. Repeat steps 3 and 4.

Output digital data from 00 to FF constant steps of 01 to DAC1 repeat this sequence again and again. As a result a saw – tooth wave will be generated at DAC1 output.

**PROGRAM:**

ADDRESS	LABEL	MNEMON ICS	OPCODE	OPERAND	COMMENT
		MOV DPTR,#FFC0			
		MOV A,#00			
	LOOP	MOVX @DPTR,A			
		INC A			
		SJMP LOOP			

**(c) TRIANGULAR WAVE GENERATION**

1. Load the initial value (00) to Accumulator.
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. If accumulator content is zero proceed to next step. Else go to step 3.
5. Load value (FF) to accumulator.
6. Move the accumulator content to DAC.
7. Decrement the accumulator content by 1.
8. If accumulator content is zero go to step 2. Else go to step 2.

The following program will generate a triangular wave at DAC2 output. The program is self explanatory.

ADDRESS	LABEL	MNEMON ICS	OPCODE	OPERAND	COMMENT
		MOV DPTR,#FFC8			
	START	MOV A,#00			
	LOOP1	MOVX @DPTR,A			
		INC A			
		JNZ LOOP1			
		MOV A,#FF			
	LOOP2	MOVX @DPTR,A			
		DEC A			
		JNZ LOOP2			
		LJMP START			

**OBSERVATION:**

WAVE FORMS	AMPLITUDE	TIME PERIOD
Square waveform		
Saw tooth waveform		
Triangular waveform		

**Result:**

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8051 trainer kit.

**Ex.No:15****INTERFACING 8253 TIMER WITH 8085****AIM:**

To interface 8253 Interface board to 8085 microprocessor to demonstrate the generation of square wave.

**APPARATUS REQUIRED:**

1. 8085 microprocessor kit
2. 8253 Interface board
3. DC regulated power supply
4. CRO.

**PROGRAM:**

Address	Opcodes	Label	Mnemonic	Operands	Comments
4100	3E 36	START:	MVI	A, 36	Channel 0 in mode 3
4102	D3 CE		OUT	CE	Send Mode Control word
4104	3E 0A		MVI	A, 0A	LSB of count
4106	D3 C8		OUT	C8	Write count to register
4108	3E 00		MVI	A, 00	MSB of count
410A	D3 C8		OUT	C8	Write count to register
410C	76		HLT		

Set the jumper, so that the clock 0 of 8253 is given a square wave of frequency 1.5 MHz. This program divides this PCLK by 10 and thus the output at channel 0 is 150 KHz.

Vary the frequency by varying the count. Here the maximum count is FFFF H. So, the square wave will remain high for 7FFF H counts and remain low for 7FFF H counts. Thus with the input clock frequency of 1.5 MHz, which corresponds to a period of 0.067 microseconds, the resulting square wave has an ON time of 0.02184 microseconds and an OFF time of 0.02184 microseconds.

To increase the time period of square wave, set the jumpers such that CLK2 of 8253 is connected to OUT 0. Using the above-mentioned program, output a square wave of frequency 150 KHz at channel 0. Now this is the clock to channel 2.

**CONTROL WORD:**

SC1	SC2	RW1	RW0	M2	M1	M0	BCD
-----	-----	-----	-----	----	----	----	-----

**SC-SELECT COUNTER:**

SC1	SC0	SELECT COUNTER
0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Read back command

**M-MODE:**

M2	M1	M0	MODE
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

**READ/WRITE:**

RW1	RW0	
0	0	Counter latch command
0	1	R/W least significant bit only
1	0	R/W most significant bit only
1	1	R/W least sig first and most sig byte

BCD:

0	Binary counter 16-bit
1	Binary coded decimal counter

**Result:**

Thus the 8253 has been interfaced to 8085  $\mu$ p and with different modes of 8253 have been studied.

**Ex.No:15****STEPPER MOTOR INTERFACING WITH 8051****AIM:**

To interface a stepper motor with 8051 microcontroller and operate it.

**THEORY:**

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotary motion occurs in a step-wise manner from one equilibrium position to the next. Stepper Motors are used very wisely in position control systems like printers, disk drives, process control machine tools, etc.

The basic two-phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding. The excitation of any one winding generates a North Pole. A South Pole gets induced at the diametrically opposite side. The rotor magnetic system has two end faces. It is a permanent magnet with one face as South Pole and the other as North Pole.

The Stepper Motor windings A1, A2, B1, B2 are cyclically excited with a DC current to run the motor in clockwise direction. By reversing the phase sequence as A1, B2, A2, B1, anticlockwise stepping can be obtained.

**2-PHASE SWITCHING SCHEME:**

In this scheme, any two adjacent stator windings are energized. The switching scheme is shown in the table given below. This scheme produces more torque.

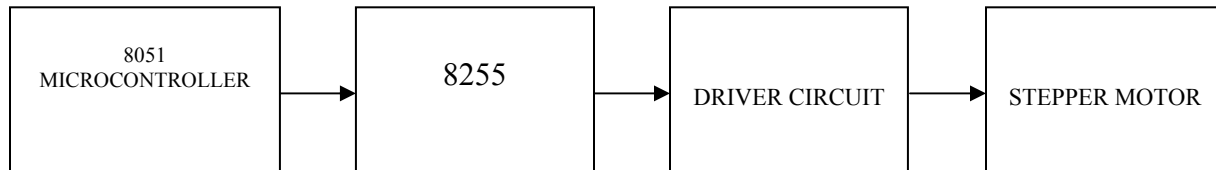
ANTICLOCKWISE						CLOCKWISE					
STEP	A1	A2	B1	B2	DATA	STEP	A1	A2	B1	B2	DATA
1	1	0	0	1	9h	1	1	0	1	0	Ah
2	0	1	0	1	5h	2	0	1	1	0	6h
3	0	1	1	0	6h	3	0	1	0	1	5h
4	1	0	1	0	Ah	4	1	0	0	1	9h

**ADDRESS DECODING LOGIC:**

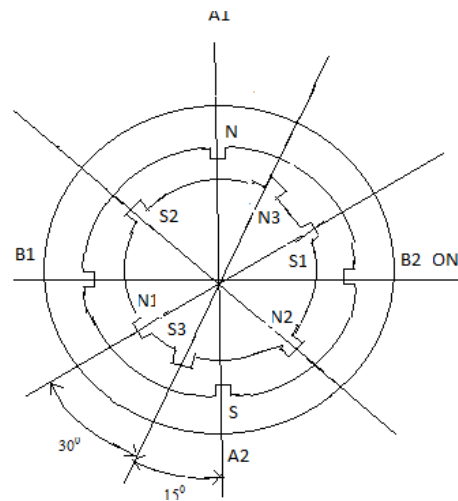
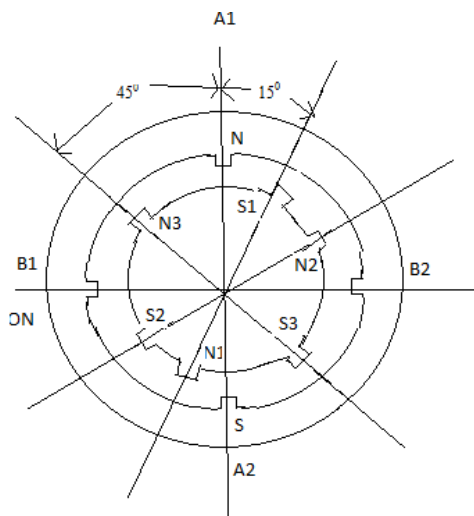
The 74138 chip is used for generating the address decoding logic to generate the device select pulses, CS1 & CS2 for selecting the IC 74175. The 74175 latches the data bus to the stepper motor driving circuitry.

Stepper Motor requires logic signals of relatively high power. Therefore, the interface circuitry that generates the driving pulses use silicon Darlington pair transistors. The inputs for the interface circuit are TTL pulses generated under software control using the Microcontroller Kit. The TTL levels of pulse sequence from the data bus are translated to high voltage output pulses using a buffer 7407 with open collector.

**BLOCK DIAGRAM:**



**REPRESENTATION:**



**PROGRAM :**

Address	OPCODES	Label	MNEMONICS	OPERAND	Comments
			ORG	4100h	
4100		START	MOV	DPTR, #TABLE	Load the start address of switching scheme data TABLE into Data Pointer (DPTR)
4103			MOV	R0, #04	Load the count in R0
4105		LOOP:	MOVX	A, @DPTR	Load the number in TABLE into A
4106			PUSH	DPH	Push DPTR value to Stack
4108			PUSH	DPL	
410A			MOV	DPTR, #0FFC0h	Load the Motor port address into DPTR
410D			MOVX	@DPTR, A	Send the value in A to stepper Motor port address
410E			MOV	R4, #0FFh	Delay loop to cause a specific amount of time delay before next data item is sent to the Motor
4110		DELAY:	MOV	R5, #0FFh	
4112		DELAY1:	DJNZ	R5, DELAY1	
4114			DJNZ	R4, DELAY	
4116			POP	DPL	POP back DPTR value from Stack
4118			POP	DPH	
411A			INC	DPTR	Increment DPTR to point to next item in the table
411B			DJNZ	R0, LOOP	Decrement R0, if not zero repeat the loop
411D			SJMP	START	Short jump to Start of the program to make the motor rotate continuously
411F		TABLE: :	DB	09 05 06 0Ah	Values as per two-phase switching scheme

**PROCEDURE:**

1. Enter the above program starting from location 4100.and execute the same.
2. The stepper motor rotates.
3. Varying the count at R4 and R5 can vary the speed.
4. Entering the data in the look-up TABLE in the reverse order can vary direction of rotation.

**RESULT:**

Thus a stepper motor was interfaced with 8051 and run in forward and reverse directions at various speeds.