ST.ANNE'S

COLLEGE OF ENGINEERING AND TECHNOLOGY

(An ISO 9001:2015 certified Institution)

Anguchettypalayam, Panruti – 607 106

EE8681- MICROPROCESSOR AND MICROCONTROLLER LABORATORY

OBSERVATION NOTE

(FOR III B.E ELECTRICAL AND ELECTRONICS ENGINEERING STUDENTS)

NAME :_____

REGISTER NO : _____

YEAR / SEMESTER: III Year / VI Semester

PERIOD : **Mar 2022 - June 2022**

AS PER ANNA UNIVERSITY (CHENNAI) SYLLABUS

2017 REGULATION

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING PREPARED

BY: Mr. D. UMA MAHESWARI, ASP/ECE

ABOUT OBSERVATION NOTE & PREPARATION OF RECORD

- This Observation contains the basic diagrams of the circuits enlisted in the syllabus of EE8681- MICROPROCESSOR AND MICROCONTROLLER LABORATORY course, along with the design the design of various components of the circuit and controller.
- Aim of the experiment is also given at the beginning of each experiment. Once the student is able to design the circuit as per the circuit diagram, he/she is supposed to go through the instructions carefully and do the experiments step by step.
- They should note down the readings (observations) and tabulate them as specified.
- It is also expected that the students prepare the theory relevant to the experiment referring to prescribed reference book/journals in advance, and carry out the experiment after understanding thoroughly the concept and procedure of the experiment.
- They should get their observations verified and signed by the staff within two days and prepare & submit the record of the experiment while they come for the laboratory in subsequent week.
- The record should contain experiment No., Date ,Aim, Apparatus required, Theory, Procedure and result on one side(i.e., Right hand side, where rulings are provided) and Circuit diagram, Design, Model Graphs, Tabulations and Calculations on the other side (i.e., Left hand side, where no rulings is provided)
- All the diagrams and table lines should be drawn in pencil
- The students are directed to discuss & clarify their doubts with the staff members as and when required. They are also directed to follow strictly the guidelines specified.

EE8681- MICROPROCESSOR AND MICROCONTROLLER LABORATORY

SYLLABUS

LIST OF EXPERIMENTS

1. Simple arithmetic operations:

   (i) addition (ii)subtraction (iii) multiplication (iv) 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 (8255)/ Serial communication(8251)

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.

OUTCOMES:

At the end of the course, the student should be able to:

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

CYCLE I

8-bit Microprocessor

1. Simple arithmetic operations:

    (a) Addition (b) subtraction (c) multiplication (d) division.

2. Programming with control instructions:

    (a) Ascending order (b) Descending order

    (c) Maximum of numbers (d) Minimum of numbers.

3. Rotate instructions:

4. Code conversion.

(a) ASCII to hexadecimal number (b) hexadecimal to ASCII(c) hexadecimal to decimal number (d) decimal to hexadecimal number

8085 Interfacing Program

5. Interface Experiments: (a) A/D Interfacing. (b) D/A Interfacing

6. Traffic light controller interfacing

7. I/O port Communication using 8255 & Serial Communication using 8251

8. Read a key using 8279 interface

9. Interface display using 8279 interface

10. 8085 Programming Practices with Simulators/ Jubin Open source Software

CYCLE II

8-bit Microcontroller

9. Demonstration of basic instructions with 8051 Micro controller execution, including:

    (a)Addition (b) subtraction (c) multiplication (d) division.

    (e) Conditional jumps, looping : Sum of elements in an array

    (f) 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

LIST OF EXPERIMENTS

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

LIST OF EXPERIMENTS

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

8085 MICROPROCESSOR EXPERIMENTS

EXP NO:	SIMPLE ARITHMETIC OPERATIONS
DATE	

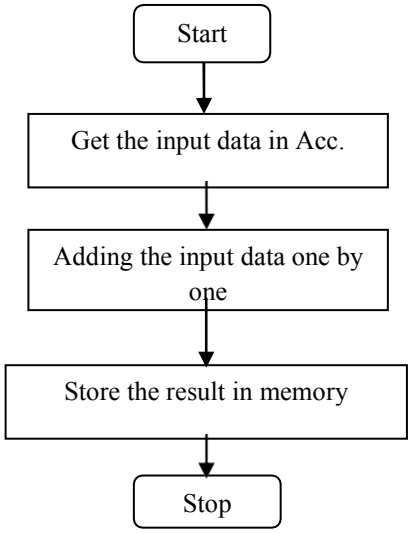
AIM:

To write an assembly language program to add, multiply, divide and subtract two 8 bit numbers at two consecutive locations using 8085 microprocessor kit.

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No

FLOWCHART FOR 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.

ADDITION:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVI A, 05H	3E, 05	Move immediate 05H data in Acc.
4102		MVI B, 06H	06, 06	Move immediate 06H data in B register
4104		MVI C, 0AH	0E, 0A	Move immediate 0AH data in C register
4106		MVI D, 12H	16, 12	Move immediate 12H data in D register
4108		MVI E, 20H	IE, 20	Move immediate 20H data in E register
410A		ADD B	80	Adding Acc and B
410B		ADD C	81	Adding Acc and C
410C		ADD D	82	Adding Acc and D
410D		ADD E	83	Adding Acc and E
410F		STA 4500	32, 00, 45	Store the result
4112		HLT	76	Stop the program

INPUT

4100-04

4103-05

4105-0A

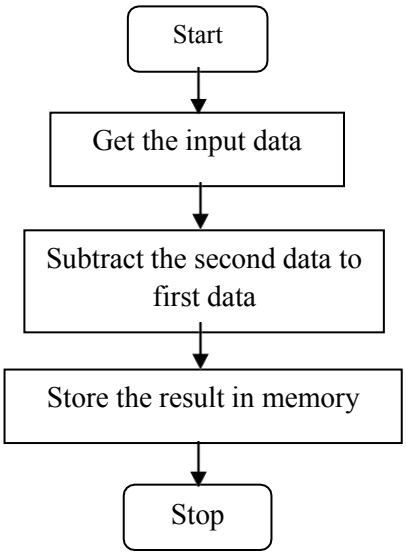
4107-12

4109- 20

OUTPUT

4500-47

FLOWCHART FOR SUBTRACTION



ALGORITHM

1. The subtrahend stored in A register.
2. The minuend stored in B register.
3. Store the result in 4600.

SUBTRACTION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVI A, 05H	3E, 05	Move immediate the 05H data in Acc.
4102		MVI B, 02H	06, 02	Move the data 02 data into B register
4104		SUB B	90	Subtract data of Acc & B register.
4105		STA 4600	32, 00, 46	Store the result in 4600
4108		HLT	76	STOP the program.

INPUT:

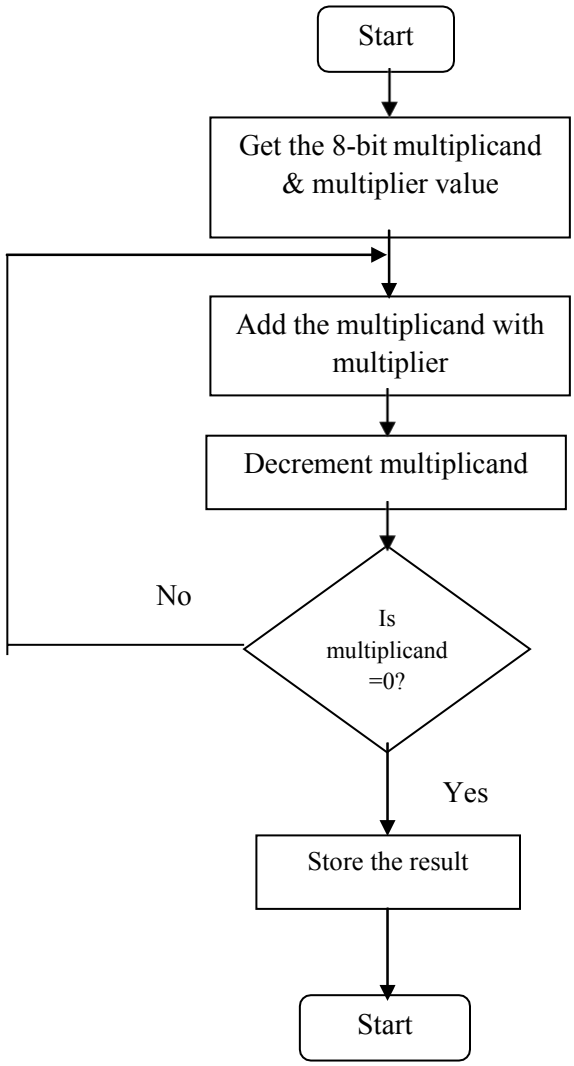
4100-05

4102-02

OUTPUT:

4A00-03

FLOWCHART FOR MULTIPLICATION



MULTIPLICATION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVI B, 07H	06, 07	Move immediate data 05H into B register
4102		MVI C, 06H	0E, 06	Move immediate data 04H into B register
4104		XRA A	AF	XOR operation of Acc.
4105	GO:	ADD B	80	Adding Acc. & B register
4106		DCR C	OD	Decrement C register
4107		JNZ GO	C2, 05, 41	Jump NO zero ON C register
410A		STA 4A00	32, 00, 4A	Store the result
410D		HLT	76	STOP the program

INPUT

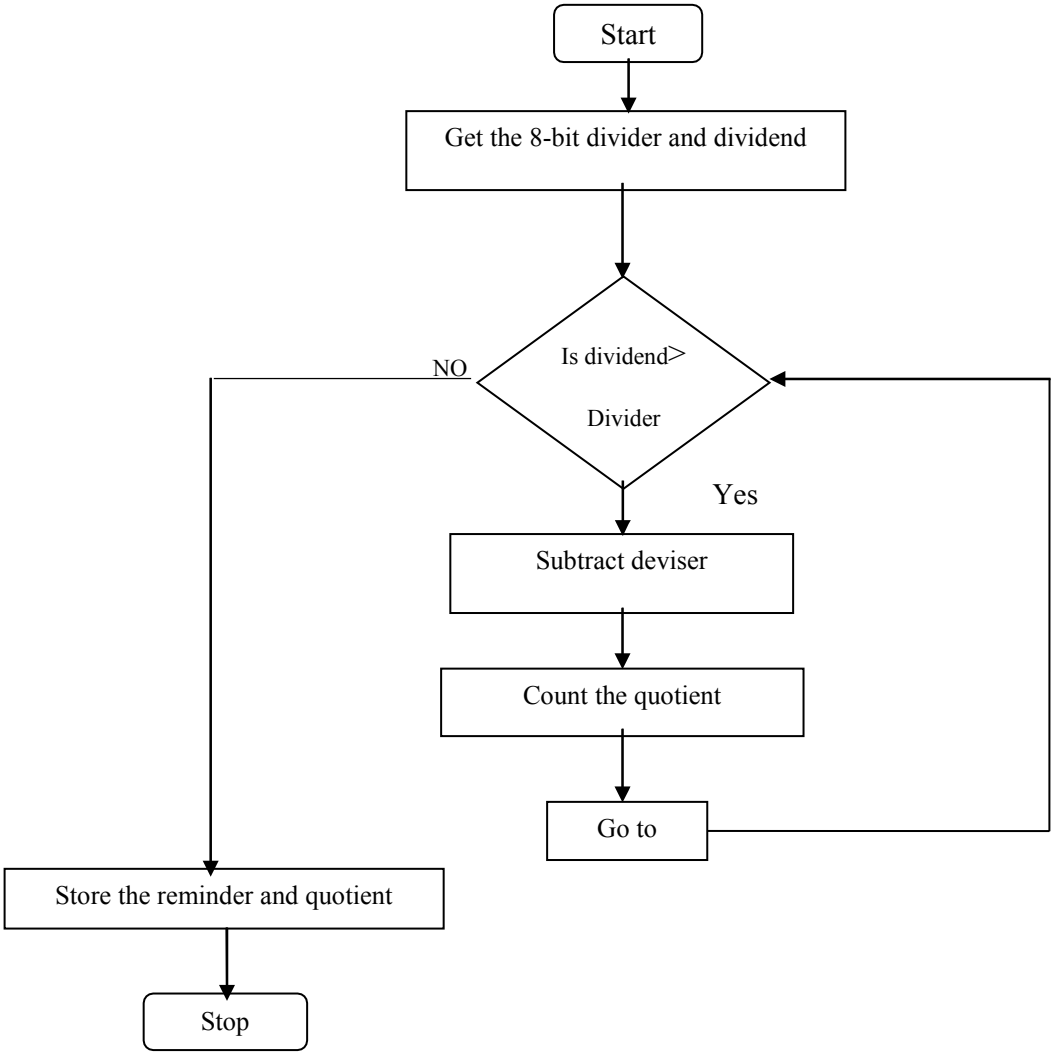
4100-07

4102-06

OUTPUT

4A00-2A

FLOWCHART FOR DIVISION



DIVISION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVI A, 25H	3E, 25	Move immediate data 0AH in Acc..
4102		MVI B, 05H	06, 05	Move immediate data into 'B' register
4104		MVI C, 00H	0E, 00	Move immediate data 00H in 'C' register
4106	LOOP2:	CMP, B	B8	Compare Acc & 'B' register
4107		JC LOOP1	DA, OF, 41	Jump with carry
410A		SUB, B	90	Subtract Acc & 'B' register data
410B		INR, C	0C	Increment 'C' register
410C		JMP LOOP2	C3, 06, 41	Jump in 4106
410F	LOOP1:	STA 4B00	32,00, 4B	Store the result 4A00
4112		MOV A, C	79	Move 'C' into Acc
4113		STA 4B01	32, 01, 4B	Store the result
4116		HLT	76	STOP the program

INPUT

4100-25

4102-05

OUTPUT

4B00-02 (Reminder)

4B01-07(Quotient)

RESULT

Thus the above assembly language program for 8 bit Addition, subtraction, division and multiplication has been executed and the output results are verified.

EXP NO:	PROGRAMMING WITH CONTROL INSTRUCTIONS
DATE	

A) SMALLEST AND BIGGEST NUMBER

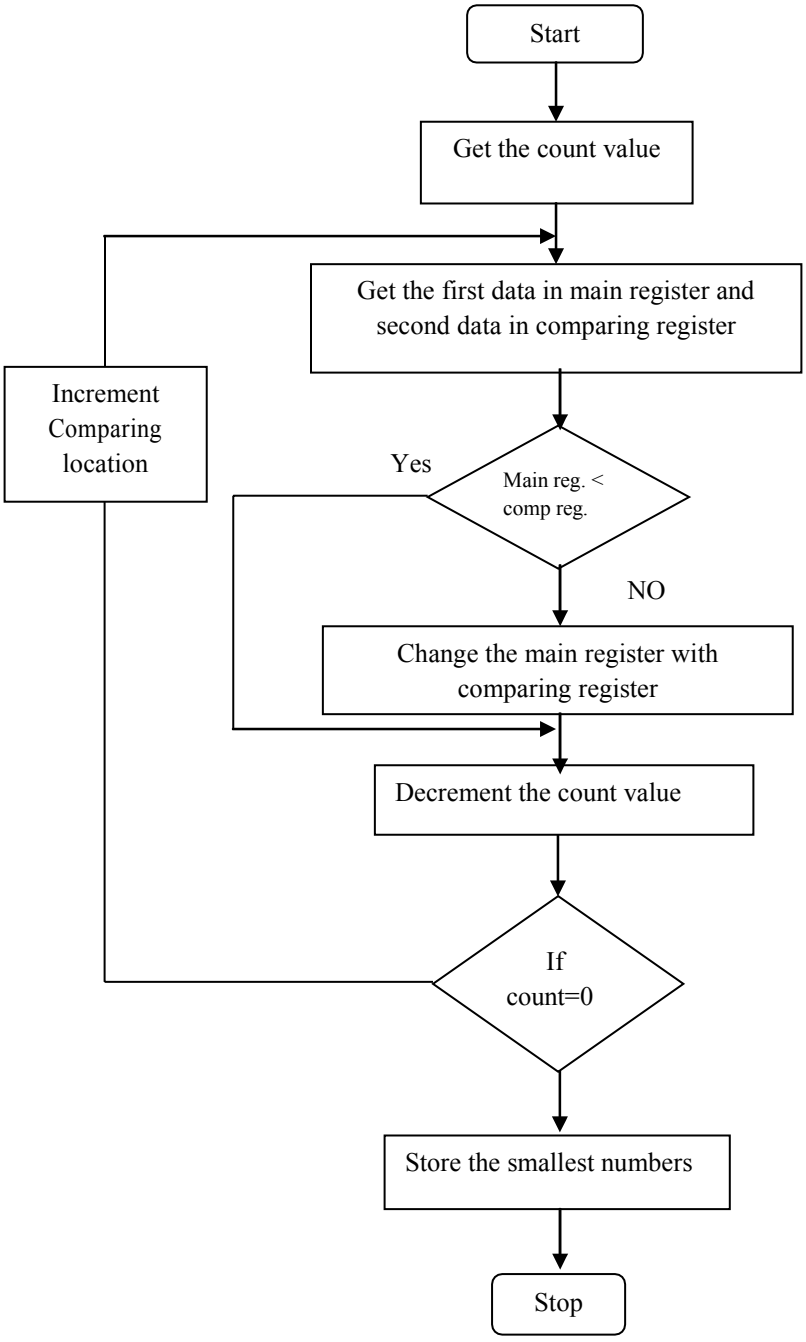
AIM

To write an assemble language program to find smallest and biggest number from the given set of values using 8085 Microprocessor kit.

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No

FLOWCHART FOR SMALLEST NUMBERS



SMALLEST VALUE

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		LXI H, 4200	21, 00, 42	Load the data from given
4103		MOV C, M	4E	Move content M to 'C' register
4104		MVI A, FF	3E, FF	Move immediate data in Acc..
4106	Loop2:	IN X H	23	Increment register pair (HL)
4107		CMP M	BE	Compare memory to register
4108		JC LOOP1	DA,OC, 41	Jump to carry LOOP1 If carry fly is set
410B		MOV A, M	7E	Move count memory to Accumulator
410C	Loop1:	DCR C	OD	Decrement 'C' register by one
410D		JNZ LOOP2	C2, 06, 41	Jump to LOOP2
4110		STA 4207	32, 07, 42	Store the result
4113		HLT	76	STOP the program

OBSERVATION

COUNT VALUE - 4200 - 04

INPUT

1) 4201 - 03

2) 4202 - 02

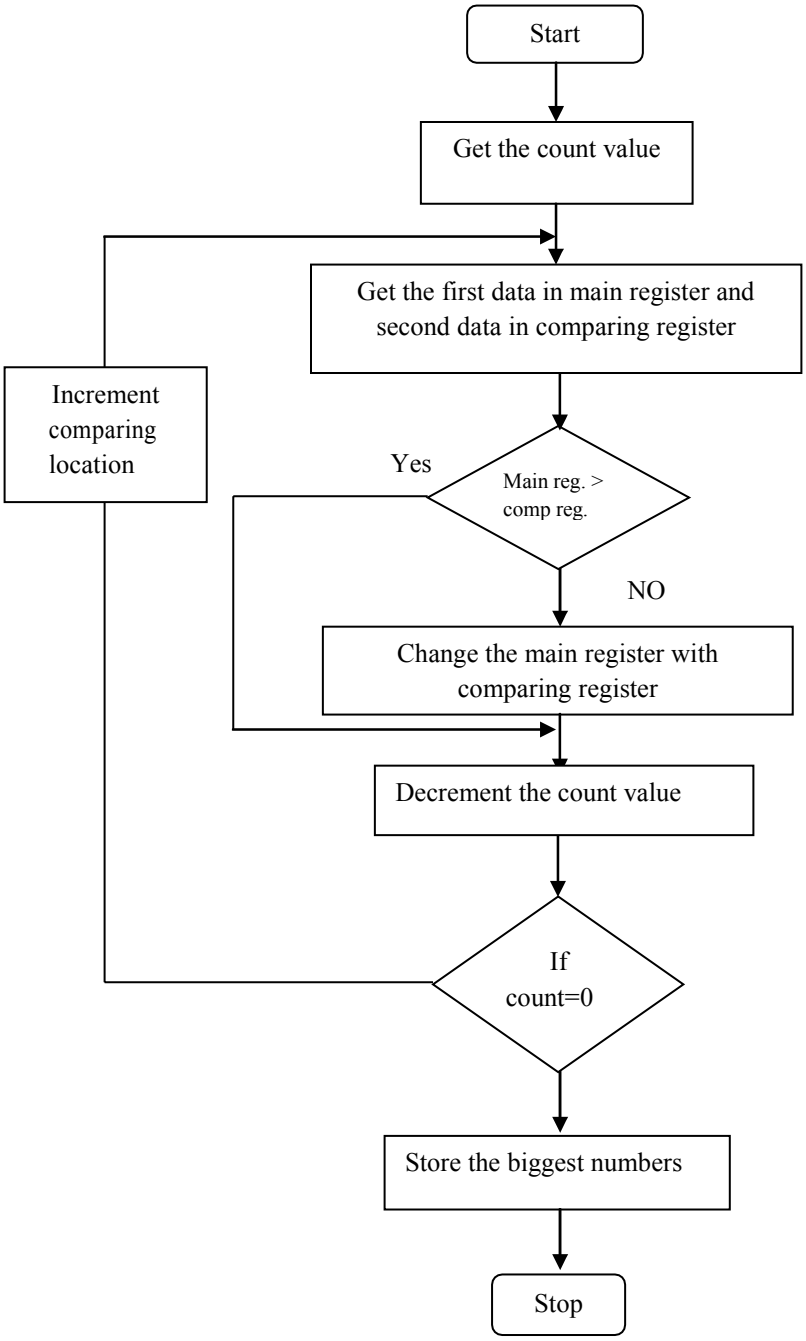
3) 4203 - 01

4) 4204 - 2F

OUTPUT

4207 - 01 Smallest value

FLOWCHART FOR BIGGEST NUMBERS



BIGGEST NUMBERS

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		LXI H, 4400	21,00,44	Load the data from given
4103		MOV C, M	4E	Move content M to Acc
4104		MVI A, 00	3E,00	Clear the Accumulator
4106	LOOP2:	IN X H	23	Increment 'H' data address location
4107		CMP M	BE	Compare the Acc with M
4108		JNC LOOP1	D2,OC,41	Jump ON no carry to LOOP1
410B		MOVA, M	7E	Move memory to Acc
410C	LOOP:	DCR C	OD	Decrement the count
410D		JNZ LOOP2	C2,06,41	Jump NO zero on 'C' register
4112		STA 4406	32,06,44	Store the result
4113		HLT	76	STOP the program

OBSERVATION:

COUNT VALUE - 4400 - 05

INPUT

1) 4401 - 10

2) 4402 - 08

3)4403 - 04

4)4404 - 05

5)4405 - 02

OUTPUT

4406 - 10 Biggest value

RESULT

Thus the program for finding smallest and biggest number form given set of value has been executed and the output results are verified.

B) ASCENDING AND DESCENDING ORDER

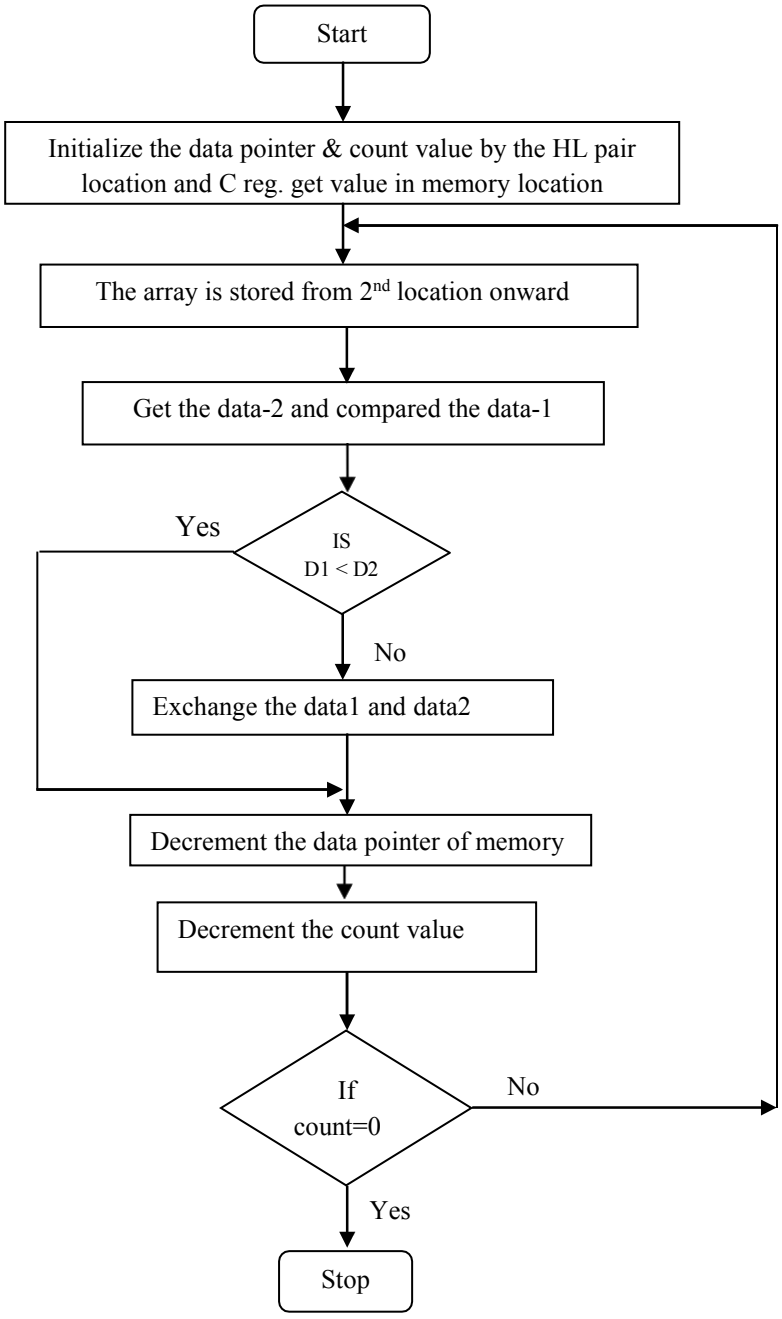
AIM

To write an assemble language program of ascending and descending order of N number.

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No

FLOWCHART FOR ASCENDING ORDER



ASCENDING ORDER

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100	UPON:	MVIB, 00	06, 00	Initialize the data pointer and count value
4102		LXIH, 4200	21,00,42	Store the HL pair and Move the count value
4105		MOV C, M	4E	Move the 'C' register to momery
4106		INX H	23	Increment the momery
4107		DCR C	OD	Decrement C
4108	REPT:	MOV A,M	7E	Move first data to Accummulater
4109		INX H	23	increment HL pair
410A		CMP M	BE	Compare first data and 2nd data
410B		JC DOWN	DA, 14, 41	If D1<D2 Jump to down otherwise exchange the first and 2nd data
410E		MOV D, M	56	
410F		MOV M, A	77	
4110		DCX H	2B	Decrement count value
4111		MOV M,D	72	
4112		MVI B, 01	06,01	Store '01' data to 'B' register
4114	DOWN:	DCR C	OD	Decrement count value
4115		JNZ REPT	C2,08,41	Jump Non zero in counter to REPT
4118		DCR B	05	
4119		JZ UPON	CA,00,41	If 'B' is zero Jump to UPON
411C		HLT	76	stop the program

OBSERVATION

COUNT VALUE - 4200 -09

INPUT

1)4201- 06 (2)4202 – 01 (3)4203 - 0A (4)4204- 0F (5)4205 – 02 (6)4206 -03

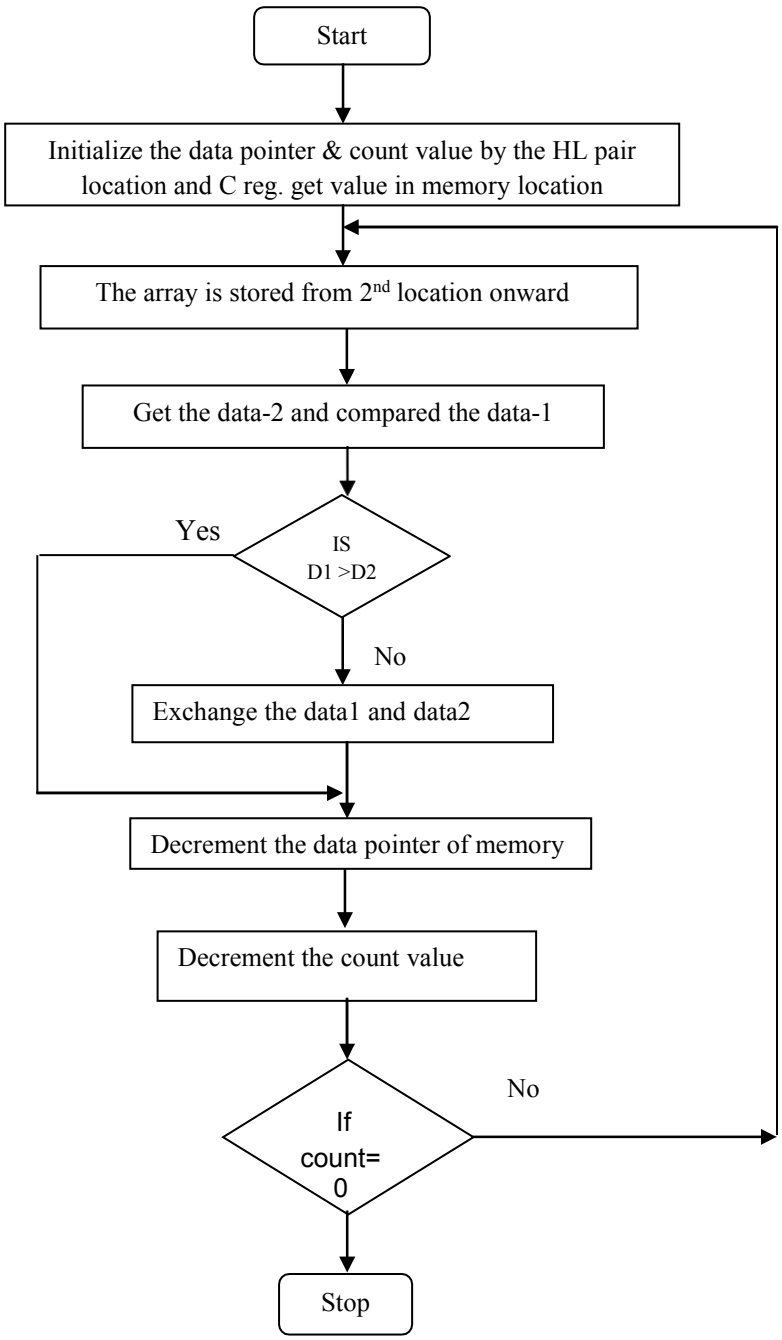
(7)4207 -0B (8)4208-20 (9)4209-04

OUTPUT

1)4201- 01 (2)4202 – 02 (3)4203 - 03 (4)4204- 04 (5)4205 – 06 (6)4206 -0A

(7)4207 -0B (8)4208-0F (9)4209-20

FLOWCHART FOR DESCENDING ORDER



DESCENDING ORDER

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100	UPON:	MVIB, 00	06, 00	Initialize the data pointer and count value
4102		LXIH, 4200	21,00,42	Store the HL pair and Move the count value
4105		MOV C, M	4E	Move the 'C' register to momery
4106		INX H	23	Increment the momery
4107		DCR C	OD	Decrement C
4108	REPT:	MOV A,M	7E	Move first data to Accummulater
4109		INX H	23	increment HL pair
410A		CMP M	BE	Compare first data and 2nd data
410B		JNC DOWN	D2, 14, 41	If D1<D2 Jump to down otherwise exchange the first and 2nd data
410E		MOV D, M	56	
410F		MOV M, A	77	
4110		DCX H	2B	Decrement count value
4111		MOV M,D	72	
4112		MVI B, 01	06,01	Store '01' data to 'B' register
4114	DOWN:	DCR C	OD	Decrement count value
4115		JNZ REPT	C2,08,41	Jump Non zero in counter to REPT
4118		DCR B	05	
4119		JZ UPON	CA,00,41	If 'B' is zero Jump to UPON
411C		HLT	76	stop the program

OBSERVATION

COUNT VALUE -4200 -09

INPUT

1)4201- 06 (2)4202 – 01 (3)4203 - 0A (4)4204- 0F (5)4205 – 02 (6)4206 -03 (7)4207 -0B (8)4208-20 (9)4209-04

OUTPUT

1)4201- 20 (2)4202 – 0F (3)4203 – 0B (4)4204- 0A (5)4205 – 06 (6)4206 -04 (7)4207 -03 (8)4208-02 (9)4209-01

RESULT

Thus the above assemble language program of ascending and descending order has been executed and the output results are verified.

EXP NO:	PROGRAMMING WITH ROTATE INSTRUCTIONS
DATE	

AIM

To write an assemble language program using rotate instruction in 8085 Microprocessor kit.

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No

THEORY:

Rotate accumulator left (RLC):

In this instruction, each bit is shifted to the adjacent left position. Bit D7 becomes D0. Carry flag CY is modified according to the bit D7.

A = D7 D6 D5 D4 D3 D2 D2 D0

A = 10101010; CY=0 //before the instruction

A = 01010101; CY=1 //after 1st RLC

A = 10101010; CY=0 //after 2nd RLC

Rotate accumulator left through carry (RAL):

In this instruction, each bit is shifted to the adjacent left position. Bit D7 becomes the carry bit and the carry bit is shifted into D0. Carry flag CY is modified according to the bit D7.

A = D7 D6 D5 D4 D3 D2 D2 D0

A = 10101010; CY=0 //before the instruction

A = 01010100; CY=1 //after 1st RAL

A = 10101001; CY=0 //after 2nd RAL

Rotate accumulator right (RRC):

In this instruction, each bit is shifted to the adjacent right position. Bit D7 becomes D0. Carry flag CY is modified according to the bit D0.

A = D7 D6 D5 D4 D3 D2 D2 D0

A = 10000001; CY=0 //before the instruction

A = 11000000; CY=1 //after 1st RRC

A = 01100000; CY=0 //after 2nd RRC

Rotate accumulator right through carry (RAR):

In this instruction, each bit is shifted to the adjacent right position. Bit D0 becomes the carry bit and the carry bit is shifted into D7. Carry flag CY is modified according to the bit D0.

A= D7 D6 D5 D4 D3 D2 D2 D0

A = 10000001; CY=0 //before the instruction

A = 01000000; CY=1 //after 1st RAR

A = 10100000; CY=0 //after 2nd RAR

ROTATE INSTRUCTION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVI A, 94H	3E, 94	
4102		RLC	07	Rotate accumulator left
4103		STA 5000	32, 00, 50	
4106		RRC	0F	Rotate accumulator right
4107		STA 5001	32, 01, 50	
410A		RAR	1F	Rotate accumulator right through carry
410B		STA 5002	32, 02, 50	
410E		RAL	17	Rotate accumulator left through carry
410F		STA 5003	32,02,50	
4112		HLT	76	

INPUT

4100- 94

OUTPUT

1)5000-29

2)5001–94

3)5002-CA

4)5003-94

RESULT

Thus the above assemble language program rotate instruction has been executed and the output results are verified.

EXP NO:	CODE CONVERSION
DATE	

AIM:

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

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No

A) BCD TO HEX CONVERSION

ALGORITHM:

1) Initialize memory pointer to 4150 H
2) Get the Most Significant Digit (MSD)
3) Multiply the MSD by ten using repeated addition
4) Add the Least Significant Digit (LSD) to the result obtained in previous step
5) Store the HEX data in Memory

PROGRAM:

```
LXI H,4150
MOV A,M // Initialize memory pointer
ADD A     // MSD X 2
MOV B,A   // Store MSD X 2
ADD A     // MSD X 4
ADD A     // MSD X 8
ADD B     // MSD X 10
INX H     // Point to LSD
ADD M     // Add to form HEX
INX H
MOV M,A // Store the result
HLT
```

OBSERVATION:

INPUT:

4150 : 02 (MSD)

4151 : 09 (LSD)

OUTPUT:

4152 : 1D

B) HEX TO BCD CONVERSION

ALGORITHM:

1) Initialize memory pointer to 4150 H
2) Get the Hexa decimal number in C - register
3) Perform repeated addition for C number of times
4) Adjust for BCD in each step
5) Store the BCD data in Memory

PROGRAM:

```
        LXI H,4150 // Initialize memory pointer
        MVI D,00   // Clear D- reg for Most significant Byte
        XRA A      // Clear Accumulator
        MOV C,M    // Get HEX data

LOOP2:  ADI 01     // Count the number one by one
        DAA// Adjust for BCD count
        JNC LOOP1
        INR D

LOOP1:  DCR C
        JNZ LOOP2
        STA 4151   // Store the Least Significant Byte
        MOV A,D
        STA 4152   // Store the Most Significant Byte
        HLT
```

OBSERVATION:

INPUT:

4150 : FF

OUTPUT:

4151 : 55 (LSB)

4152 : 02 (MSB)

C) HEX TO ASCII CONVERSION

ALGORITHM:

1. Load the given data in A- register and move to B – register
2. Mask the upper nibble of the Hexa decimal number in A – register
3. Call subroutine to get ASCII of lower nibble
4. Store it in memory
5. Move B –register to A – register and mask the lower nibble
6. Rotate the upper nibble to lower nibble position
7. Call subroutine to get ASCII of upper nibble
8. Store it in memory
9. Terminate the program.

PROGRAM:

```
        LDA 4200    // Get Hexa Data
        MOV B,A
        ANI 0F      // Mask Upper Nibble
        CALL SUB1       // Get ASCII code for upper nibble
        STA 4201
        MOV A,B
        ANI F0      // Mask Lower Nibble
        RLC
        RLC
        RLC
        RLC
        CALL SUB1       // Get ASCII code for lower nibble
        STA 4202
        HLT
SUB1:   CPI 0A
        JC SKIP
        ADI 07
SKIP:   ADI 30
        RET
```

OBSERVATION:
INPUT:

4200 - E4(Hexa data)

OUTPUT:

4201 - 34(ASCII Code for 4)

4202 - 45(ASCII Code for E)

D) ASCII TO HEX CONVERSION

ALGORITHM:

1. Load the given data in A- register
2. Subtract 30 H from A – register
3. Compare the content of A – register with 0A H
4. If A < 0A H, jump to step6. Else proceed to next step.
5. Subtract 07 H from A – register
6. Store the result
7. Terminate the program

PROGRAM

```
        LDA 4500
        SUI 30
        CPI 0A
        JC SKIP
        SUI 07

SKIP:   STA 4501
        HLT
```

OBSERVATION:

INPUT:

4500 - 31

OUTPUT:

4501 - 01

RESULT:

Thus the assembly language program to perform the conversions of BCD to hexadecimal number, hexadecimal to BCD, hexadecimal to ASCII, ASCII to hexadecimal number has been executed and the output results are verified.

EXP NO:	INTERFACING OF ADC AND DAC
DATE	

A) ANALOG TO DIGITAL CONVERTOR

AIM

To write an assembly language program to convert an analog signal into a digital and to store the digital data in memory

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	ADC interface Board	1 No

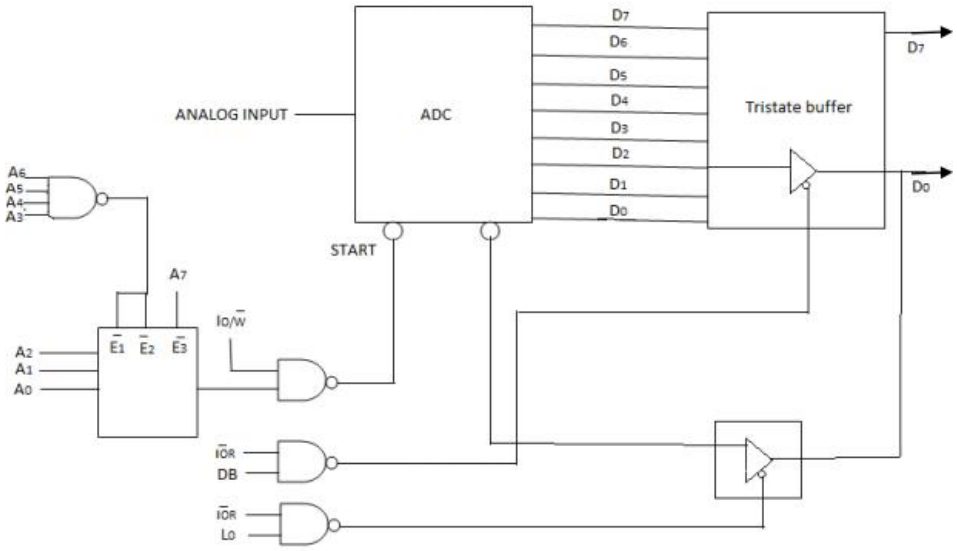
PROBLEM STATEMENT:

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

CIRCUIT DIAGRAM:



NOTE

1) Converter the analog signal in to digital and store the result in some memory location.

2) 4100 starting of use address.

3) 4500 result stored location.

SOC JUMPER SELECTION:

J2: SOC Jumper selection

J5: Channel selection

OBSERVATION:

Compare the data displayed at the LEDs with that stored at location 4150

PROGRAM:

```
        MVI A,10
        OUT C8
        MVI A,18
        OUT C8
        MVI A,01
        OUT D0
        MVI A,00
        OUT D0
LOOP:   IN D8
        ANI 01
        CPI 01
        JNZ LOOP
        IN C0
        STA 4150
        HLT
```

OUTPUT:

ANALOG VOLTAGE	DIGITAL DATA ON LED DISPLAY									HEX CODE IN LOCATION 4150
	D7	D6	D5	D4	D3	D2	D1	D0	OUTPUT	

RESULT:

Thus the assembly language program to convert an analog signal into a digital has been executed successfully and the digital data was stored at desired location.

B) DIGITAL TO ANALOG CONVERTOR

AIM:

To interface DAC with 8085 to demonstrate the generation of square, saw tooth and triangular wave.

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	DAC interface Board	1 No

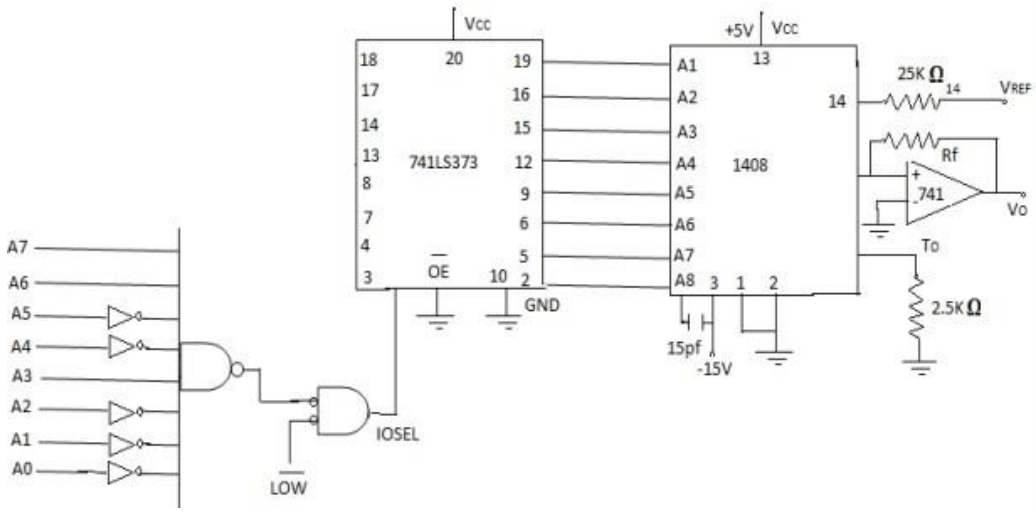
THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V.The output voltage varies in steps of 10/256 = 0.04 (appx.). The digital data input and the corresponding output voltages are presented in the below Table.

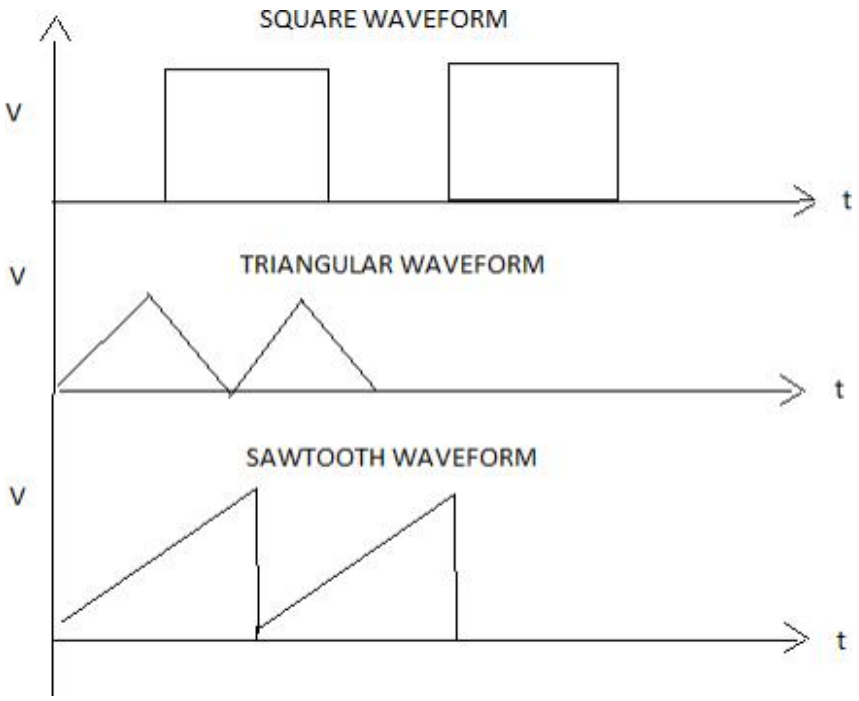
Input Data in HEX	Output Voltage
00	- 5.00
01	- 4.96
02	- 4.92
7F	0.00
FD	4.92
FE	4.96
FF	5.00

Referring to above table, with 00 as input to DAC, the analog output is – 5V. Similarly, with FF as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc,. The port address of DAC is 08 H.

CIRCUIT DIAGRAM:



WAVEFORMS:



SQUARE WAVE GENERATION

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

```
START: MVI A,00
       OUT C8
       CALL DELAY
       MVI A,FF
       OUT C8
       CALL DELAY
       JMP START
DELAY: MVI B,05
L1:    MVI C,FF
L2:    DCR  C
       JNZ L2
       DCR B
       JNZ L1
       RET
```

OBSERVATION:

AMPLITUDE	TIME PERIOD

TRIANGULAR WAVE GENERATION

ALGORITHM:

1. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM

```
START: MVI L,00
L1:    MOV A,L
       OUT C8
       INR L
       JNZ L1
       MVI L,FF
L2:    MOV A,L
       OUT C8
       DCR L
       JNZ L2
       JMP START
```

OBSERVATION:

AMPLITUDE	TIME PERIOD

SAW TOOTH 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. Repeat Steps 3 and 4.

PROGRAM:

```
START: MVI A,00
L1:    OUT C8
       INR A
       JNZ L1
       JMP START
```

OBSERVATION:

AMPLITUDE	TIME PERIOD

RESULT

Thus the above interface assembler language digital to analog converter is verified.

EXP NO:	TRAFFIC LIGHT CONTROLLER
DATE	

AIM:

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

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	Traffic Light Controller Interface Board	1No

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.

ALGORITHM:

1. Initialize 8255, port A and port B in output mode
2. Send data on PA to glow R1 and R2.
3. Send data on PB to glow G3 and G4.
4. Load multiplier count (40) for delay.
5. Call delay subroutine.
6. Send data on PA to glow Y1 and Y2.
7. Send data on PB to glow Y3 and Y4.
8. Load multiplier count (10) for delay.

9. Call delay subroutine.

10.Send data on PA to glow G1 and G2.

11.Send data on PB to glow R3 and R4.

12.Load multiplier count (40) for delay.

13.Call delay subroutine.

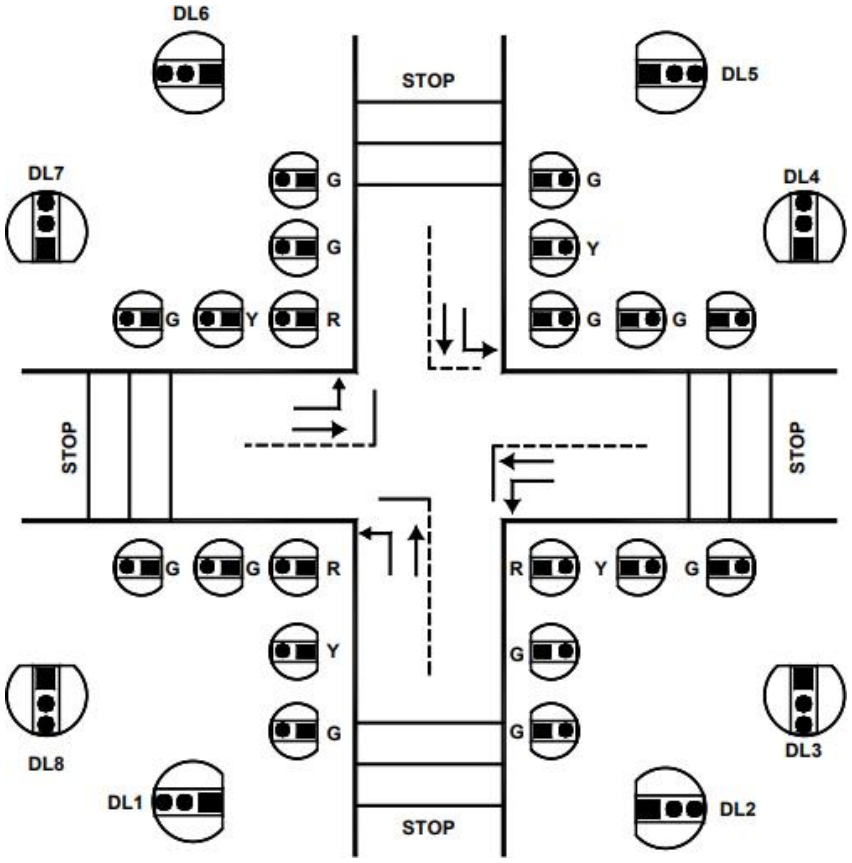
14.Send data on PA to glow Y1 and Y2.

15.Send data on PA to glow Y3 and Y4.

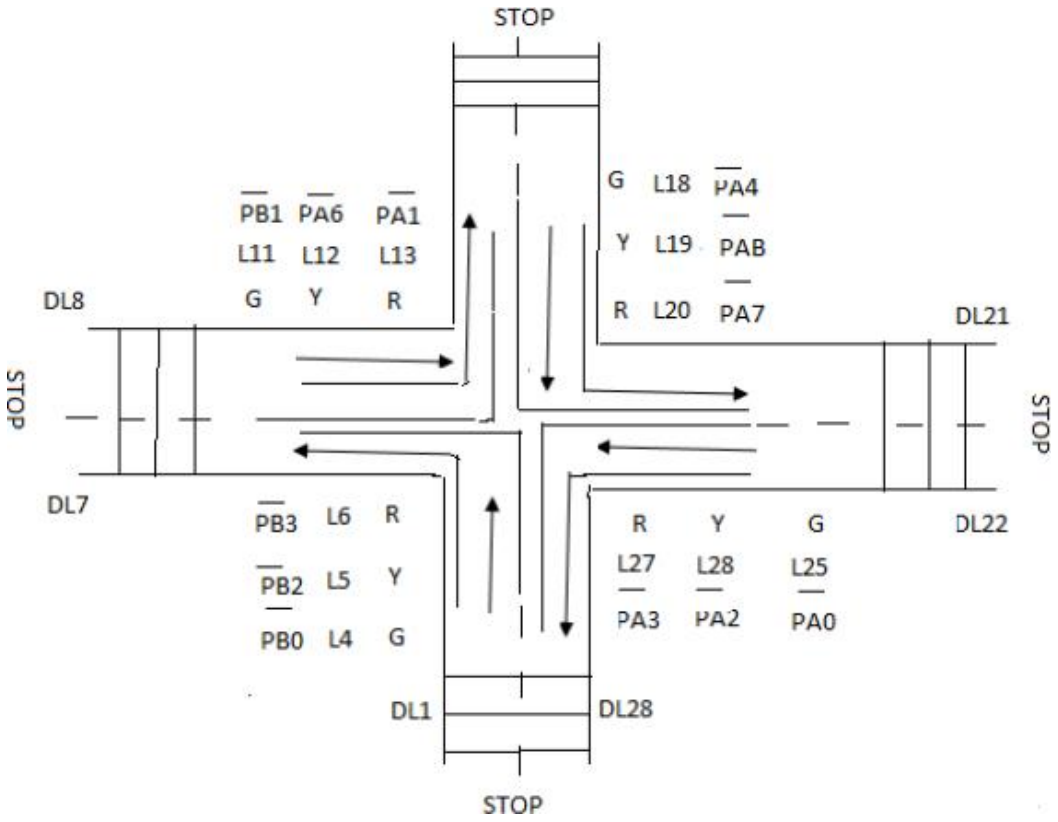
16.Load multiplier count (10) for delay.

17.Call delay subroutine

HARDWARE LAYOUT



PIN LAYOUT



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

CONTROL:

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

PORT A----- 0C

PORT B------ 0D

PORT C------ 0E

PROGRAM:

```
START: LXI H,4500
       MVI C,04
       MOV A,M
       OUT 0F
       INX H

LOOP1: MOV A,M
       OUT 0C
       INX H
       MOV A,M
       OUT 0D
       CALL DELAY
       INX H
       DCR C
       JNZ LOOP1
       JMP START

DELAY: PUSH B
       MVI C,05

LOOP3: LXI D,FFFF

LOOP2: DCX D
       MOV A,D
       ORA E
       JNZ LOOP2
       DCR C
       JNZ LOOP3
       POP B
       RET
```

OBSERVATION

4500-80
4501-1A
4502-A1
4503-64
4504-A4
4505-81
4506-5A
4507-64
4508-54
4509-8A
450A-B1
450B -A8
450C- B4
450D -88
450E -DA
450F -68
4510 -D8
4511 -1A
4512 -E8
4513 -46
4514 -E8
4515 -83
4516 -78
4517- 86
4518 -74

RESULT:

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

EXP NO:	I/O PORT INTERFACING
DATE	

AIM:

To interface 8255 PPI with microprocessor and to perform read and write operation using 8085 kit

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	8255 Interface board	1No

PERFORMING OPERATION

i) Read the data from Accumulator (use LED)

ii) Read the data to Accumulator (DIP switch)

NOTE

i. Use port A as input

ii. Use port B and C as output

iii. Address of the post are given below

a. Port A(PA) – C0

b. Port B(PB) – C2

c. Port C(PC) – C4

d. Control Register(CR) – C6

e. Control Word(CW) – 90

PROGRAM

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MVIA, CW	3E, 90	Initializations of control words
4102		OUT CR	D3,C6	Initializations of output control register
4104		MVI A, 05	3E,05	Move 05 data to accumulator
4106		OUT PB	D3, C2	Initializations of port B as output
4108		IN PA	DB, C0	Initializations of port A as input
410A		STA 4500	32,00,45	Store the accumulator result in 4500
410D		HLT	76	Stop the program

RESULT

Thus the 8255 peripheral IC is interfaced with microprocessor and performs the READ/WRITE operation is verified

EXP NO:	SERIAL COMMUNICATION
DATE	

AIM:

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

REQUIREMENTS:

S. No.	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	8251 Interface board	1 No
5	VXT parallel bus/ RS 232C cable	Few

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.words should be written in to the control register of 8251.Thesecontrol 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
16x Baud rate factor
1 stop bit
B2 , B1 = 1 , 0
L2 , L1 = 1 ,1
PEN = 0
EP = 0
S2 , S1 = 0 , 1

gives a Mode command word of 4E.

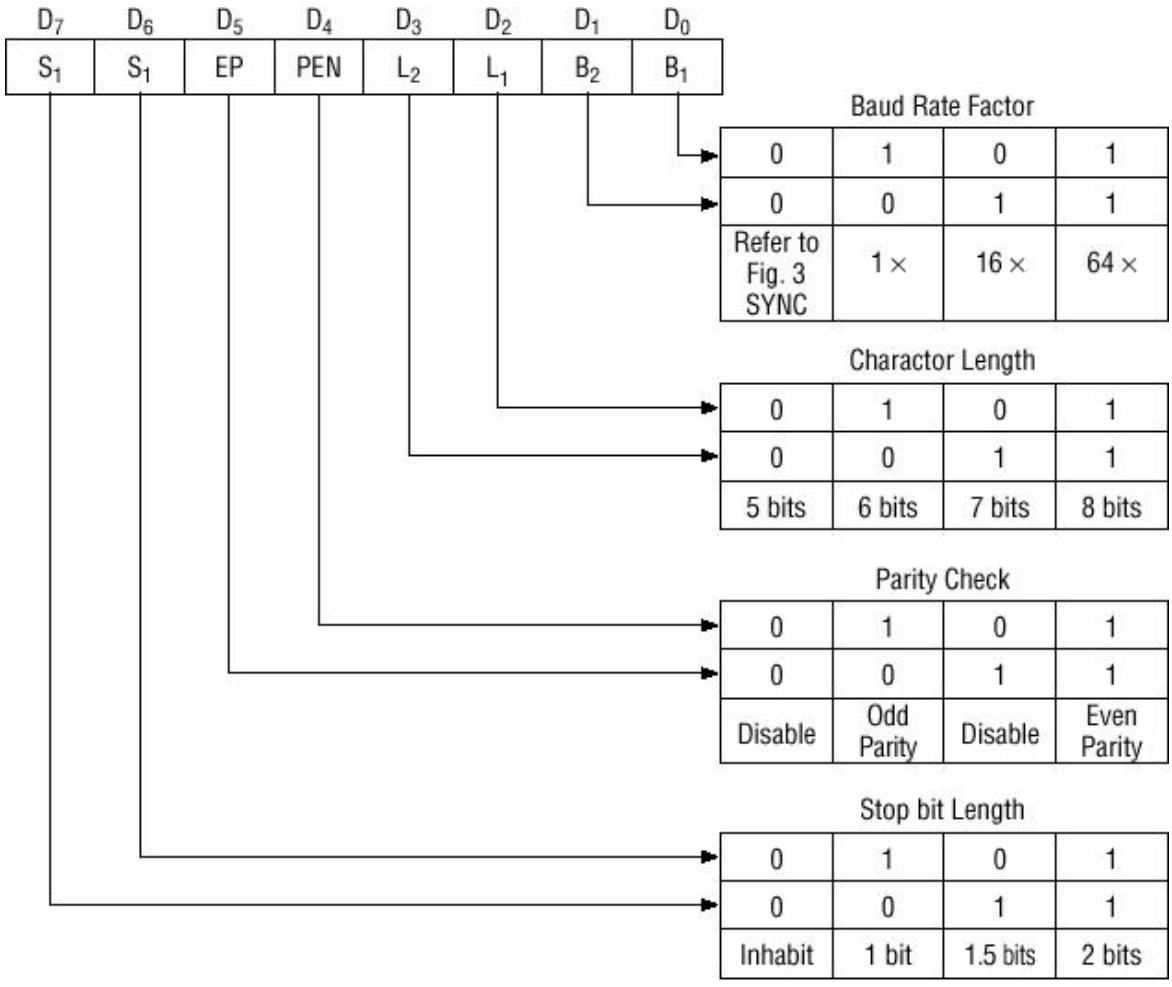


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

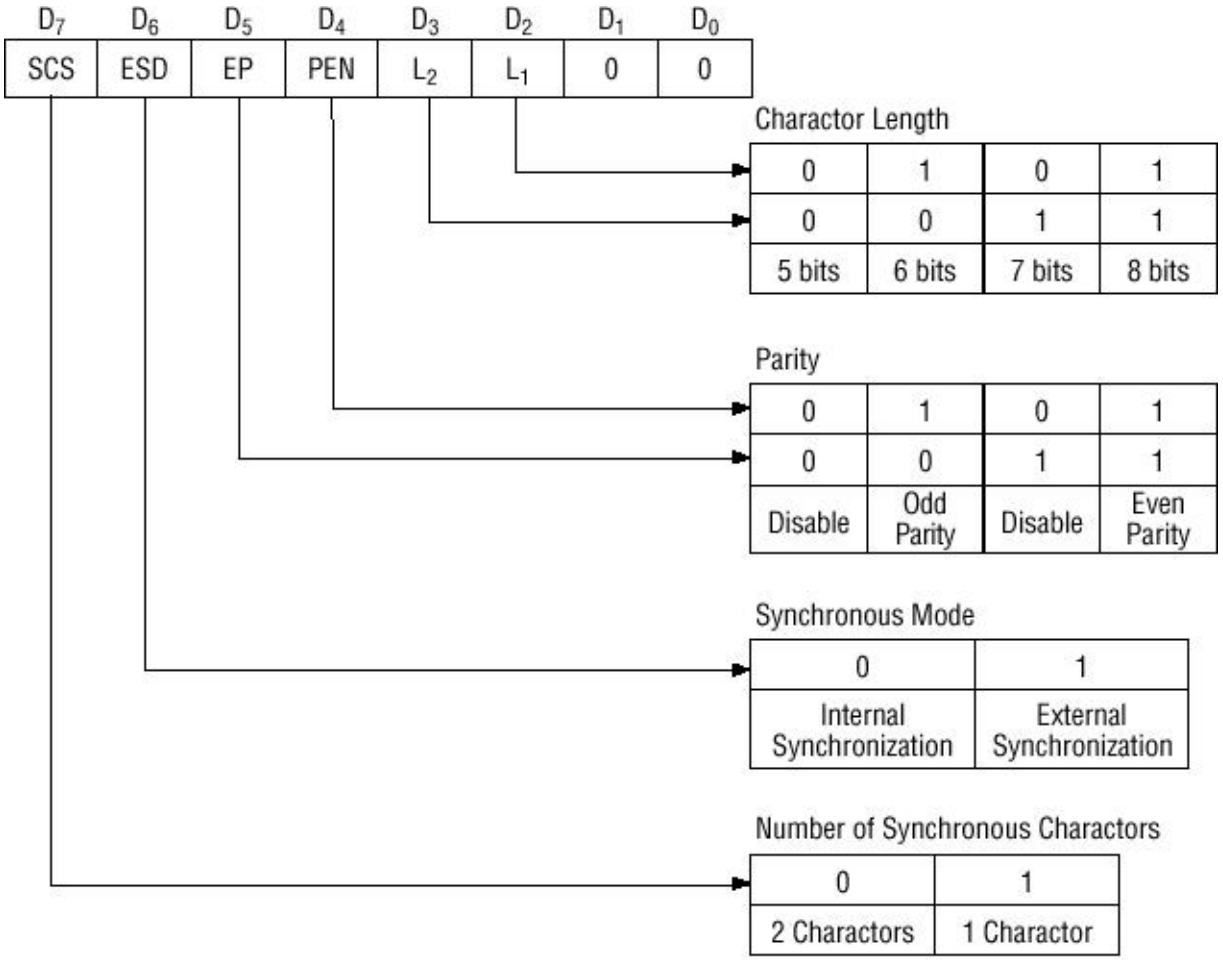


Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

2. COMMAND INSTRUCTION WORD

This format defines a status word that is used to control the actual operation of 8251. All control words written into 8251 after the mode instruction will load the command instruction. The command instructions can be written into 8251 at any time in the data block during the operation of the 8251. to return to the mode instruction format, the master reset bit in the command instruction word can be set to initiate an internal reset operation which automatically places the 8251 back into the mode instruction format. Command instructions must follow the mode instructions or sync characters. Thus the control word 37 (HEX) enables the transmit enable and receive enable bits, forces DTR output to zero, resets the error flags, and forces RTS output to zero.

When 8251A is initialized as follows using the command instruction,

Reset Error flags,
Enable transmission and reception,
Make RTS and DTR active low.

EH = 0 SBRK = 0
IR = 0 RxE = 1
RTS = 1 DTR = 1
ER = 1 TxEN = 1

We get a command word of 37

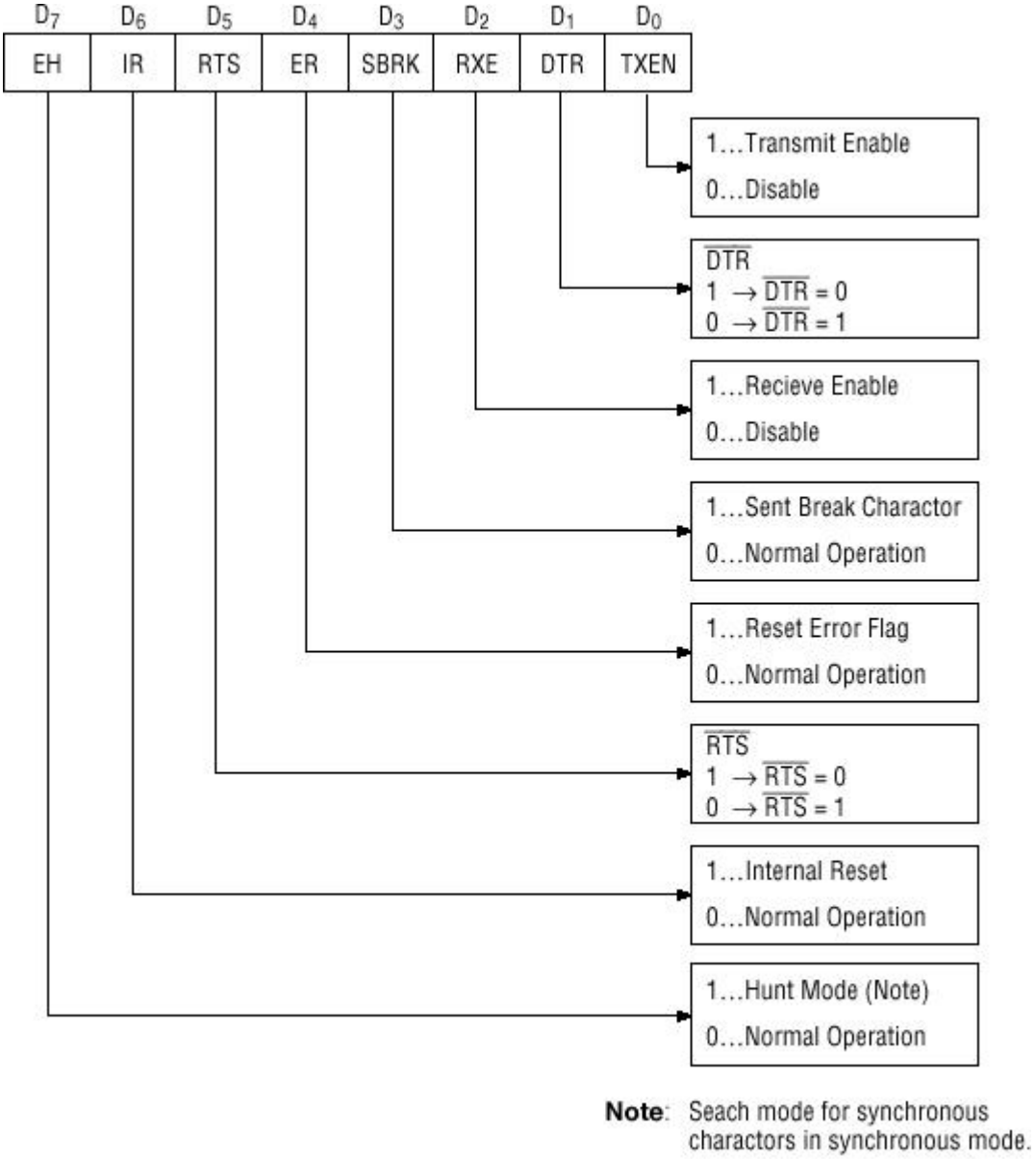


Fig. 4 Bit Configuration of Command

The program after initializing , will read the status register and check for TxEMPTY. If the transmitter buffer is empty then it will send 31 to the serial port and then check for a character in the receive buffer. If some character is present then, it is received and stored at location 4200H.

TRANSMITTER: Starting address: 4100

```
MVI A, 36
OUT CE
MVI A,0A
OUT C8
MVI A,00
OUT C8
MVI A,4E
OUT C2
MVI A,37
OUT C2
MVI A,31
OUT C0
RST 1
```

RECEIVER: Starting address: 4200

```
IN C0
STA 4150
RST 1
```

OBSERVATION:

Feed the above program 4100 acts as Transmitter and 4200 acts as Receiver. Execute the two programs simultaneously. Check the Receiver at location 4150H. It's content will be 31.

INPUT DATA	OUTPUT DTA

RESULT:

Thus the assembly language program for transmission and reception of character was executed and verified successfully by interfacing 8251 with 8085.

EXP NO:	INTERFACE DISPLAY
DATE	

AIM:

To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor

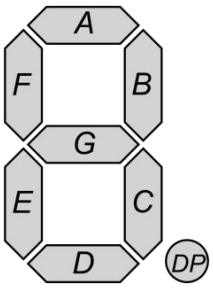
REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	8279 Interface board	1 No
5	VXT parallel bus, RS 232C cable	1 No

DISPLAY PROGRAM:

```
START: LXI H,4130
       MVI D,0F     // Initialize counter
       MVI A,10
       OUT C2       // Set Mode and Display
       MVI A,CC     // Clear display
       OUT C2
       MVI A,90     // Write Display
       OUT C2

LOOP:  MOV A,M
       OUT C0
       CALL DELAY
       INX H
       DCR D
       JNZ LOOP
       JMP START

DELAY: MVI B,A0

LOOP2: MVI C,FF

LOOP1: DCR C
       JNZ LOOP1
       DCR B
       JNZ LOOP2
       RET
```

SEVEN SEMENT DETAILS:



DATA BUS	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
SEGMENTS	**D**	**C**	**B**	**A**	**DP**	**G**	**F**	**E**

OBSERVATION:

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

INPUT:

4130 - FF
4131 –FF
4132 –FF
4133 –FF
4134 –FF
4135 –FF
4136 –FF
4137 –FF
4138 –FF
4139 –FF
413A -FF
413B -FF
413C -FF
413D -FF
413E -FF
413F -FF

RESULT:

Thus 8279 Controller was interfaced with 8085 and Rolling Display was executed Successfully and the output is verified.

EXP NO:	READ A KEY
DATE	

AIM:

To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8085 Trainer Kit	1No
2	Power Chord	1No
3	Opcode Sheet	1 No
4	8279 Interface board	1No
5	VXT parallel bus, RS 232C cable	Few

KEYBOARD PROGRAM:

```
        MVI B,08
        MVI A,02
        OUT C2
        MVI A,CC
        OUT C2
        MVI A,90
        OUT C2
        MVI A,FF

BACK: OUT C0
        DCR B
        JNZ BACK

LOP:    IN C2
        ANI 07
        JZ LOP
        MVI A,40
        OUT C2
        IN C0
        ANI 0F
        MOV L,A
        MVI H,42
        MOV A,M
        OUT C0
        JMP LOP
```

OBSERVATION

4200 - 0C

4201 - 9F

4202- 4A

4203 - 0B

4204 - 99

4205 - 29

4206 - 28

4207 - 8F

4208 - 08

4209 - 09

420A - 88

420B - 38

420C - 6C

420D - 1A

RESULT:

Thus 8279 Controller was interfaced with 8085 and Program for Read Key was Executed Successfully.

EXP NO:	PROGRAMMING PRACTICES WITH SIMULATORS
DATE	

AIM:

To write an assembly language program to add the given data stored at two consecutive locations using 8085 microprocessor simulator.

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	Jubin Open source Software	1No
2	PC	1No

THEORY

Understanding of Intel 8085 microprocessor is fundamental to getting insight into the Von-Neumann Architecture. It was first introduced in 1976, since then many generations of computer architecture have come up, some still persists while others are lost in history. This microprocessor still survives because it is still popular in university and training institutes to get students acquainted with basic computer architecture. For this purpose 8085 trainer kit are available on the market. With this academic learning purpose in mind 8085 simulator software is designed. It helps in get started easily with example codes, and to learn the architecture playfully. It also provides a trainer kit as an appealing functional alternative to real hardware. The users can write assembly code easily and get results quickly without even having the actual hardware. Jubin 8085 simualor is open source software which is available at

https://8085simulator.codeplex.com/downloads/get/86234

PROGRAM

```
MVI A,05
MVI B,06
MVI C,0A
MVI D,12
MVI E,20
ADD B
ADD C
ADD D
ADD E
STA 4500
HLT
```

OBSERVATION:

INPUT

4100-04

4103-05

4105-0A

4107-12

4109- 20

OUTPUT

4500-47

RESULT

Thus the addition of two numbers was performed using the 8085 microprocessor simulator.

8051 MICROCONTROLLER EXPERIMENTS

EXP NO:	SIMPLE ARITHMETIC OPERATION
DATE	

AIM:

To perform 8 bit addition, subtraction multiplication and division using immediate addressing in microcontroller (8051) and store result in memory.

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8051 Trainer Kit	1No
2	Power Chord	1No
3	Keyboard	1 No

NOTE:

A <enter>→ Address
GO <address><enter>
SP<address> <enter>
U<enter><address>

A) ADDITION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		CLR C	C3	Clear the carry
4101		MOV A, # 20	74,20	Get the first data
4103		ADD A, # 21	24,21	Get the second data
4105		MOV DPTR, # 4500	90,00,45	Initialize the DPTR
4108		MOVX @ DPTR, A	F0	Store the result in memory
4109	HLT	SJMP HLT	80, FE	Stop the program

OUTPUT: 4500 - 41

B) SUBTRACTION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4200		CLR C	C3	Clear carry flay
4201		MOV A, # 20	74,20	Move data 1 to Acc
4203		SUBB A, # 10	94,10	Subtract data 2from data 1
4205		MOV DPTR, # 4600	90,46,00	Initialize DPTR 4600
4208		MOV X @ DPTR, A	F0	Store the result
4209	HLT	SJMP HLT	80,FE	STOP the program

OUTPUT: 4600- 10

C) MULTIPLICATION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4300		MOV A, # 11	74,11	Move 1st data in 'A' register
4302		MOV B, # 88	75,88	Move 2nd data to 'B' register
4305		MUL AB	A4	Multiply two data
4306		MOV DPTR, # 4700	90,00,47	Initialize the DPTR
4309		MOVX @ DPTR,A	F6	Store the LSB
430A		INC DPTR	A3	Increment DPTR
430B		MOV A,B	E5	Move 'B' data into 'A'
430D		MOVX @ DPTR, A	F0	Store the MSB of result
430E	HLT	STMP HLT	80, FE	STOP the program

OUTPUT: 4700: C8 & 4701: 03

DIVISION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4400		MOV A, # 05	74,55	Move data 1 in 'A' register
4402		MOV B, # 03	75,16	Move data 2 in 'B' register
4405		DIV AB	84	Dived in First data in Second Data
4406		MOV DPTR, # 4800	90,00,48	Initialize DPTR
4409		MOVX @ DPTR, A	F0	Store the LSB
440A		INC DPTR	A3	Increment DPTR
440B		MOV A,B	E5	Store count value
440C		MOVX @ DPTR, A	F0	Store the remained value in DPTR
440D	HLT	SJMP HLT	80,FE	STOP the program

OUTPUT: 4800: 01 & 4801: 02

RESULT

Thus the 8 bit addition, subtraction multiplication and division are executed and output was verified by using microcontroller (8051).

EXP NO:	INTERFACING OF STEPPER MOTOR
DATE	

AIM:

To interface the stepper motor with microcontroller (8051) and rotate in different speed in clock wise and anticlockwise direction

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8051 Trainer Kit	1No
2	Power Chord	1No
3	Stepper motor interface board	1 No

THEORY:

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotor motion occurs in a stepwise manner from one equilibrium position to next. The motor under our consideration uses 2 – phase scheme of operation. In this scheme, any two adjacent stator windings are energized. The switching condition for the above said scheme is shown in Table.

NOTE

1. Load the given set of data from 4300 onwards.
2. Clockwise (09, 05, 06, 0A) H
3. Anticlockwise (0A, 06, 05, 04) H
4. Load time delay in DE pair.

ANTI CLOCK WISE

step	A1	A2	B1	B2	
1	1	0	1	0	0A
2	0	1	1	0	06
3	0	1	0	1	05
4	1	0	0	1	09

CLOCKWISE

step	A1	A2	B1	B2	
1	1	0	0	1	09
2	0	1	0	1	05
3	0	1	1	0	06
4	1	0	1	0	0A

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100	START	MOV DPTR, # 4500	90,45,00	Load the DPTR
4103		MOV R0, # 04	78,04	Count value in R0
4105	UP	MOVX A,@ DPTR	E0	Move the content
4106		PUSH DPH	C0,83	Of DPTR to Acc
4108		PUSH DPL	C0,82	Push the value of
410A		MOV DPTR, #FFC0	90,FF,C0	DPL & DPH
410D		MOV R2,#04	7A,04	Move the content
410E		MOV R1, # 0F	79,0F	of Accumulator to DPTR
4110	DELAY2	MOV R3 # 0F	7B,0F	
4112	DELAY1	DJNZ R1, DELAY1	DB,FE	Decrement Lump NON zero
4114		DJNZ R2, DELAY2	D9,FA	R5 to count
4116		POP DPL	DA,F8	
		MOVX @DPTR,A	F0	Pop the value of DPL
4118		POP DPL	D0,82	Pop the value of DPH
		POP DPH	D0,83	Increment the DPTR
411A		INC DPTR	A3	Content
411B		DJNZ R0, UP	D8,E4	
411E		SJMP START	80,DD	DATA

OBSERVATION:

CLOCK WISE	ANTI CLOCKWISE
4500-09H	4500-0AH
4501-05H	4501-06H
4502-06H	4502-05H
4503-0AH	4503-09H

RESULT

Thus by using the above assembly language program was successfully verified and the direction of motor was controlled using 8051 trainer kit.

EXP NO:	INTERFACING OF ADC
DATE	

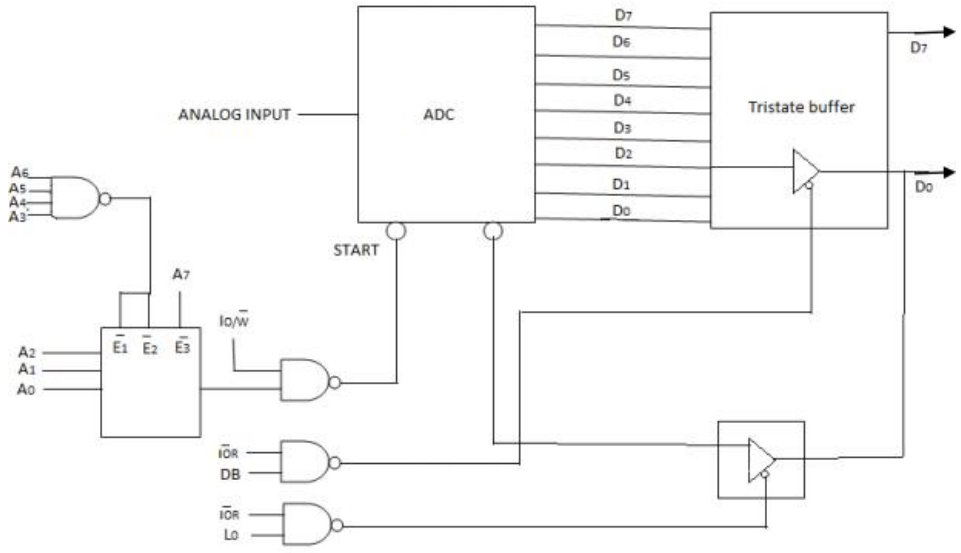
AIM:

To interface the ADC with 8051 microcontroller and generate the square wave, saw tooth wave and triangular wave.

REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8051 Trainer Kit	1No
2	Power Chord	1No
3	ADC interface Board	1 No

CIRCUIT DIAGRAM:



NOTE

1) Converter the analog signal in to digital and store the result in some memory location.

2) 4100 starting of use address.

3) 4500 result stored location.

SOC JUMPER SELECTION:

J2: SOC Jumper selection

J5: Channel selection

OBSERVATION:

Compare the data displayed at the LEDs with that stored at location 4150

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MOV DPTR, #FFC8	90,FF,C8	
4103		MOV A,#10	74,10	Select Channel 0 and make ALE Low
4105		MOVX @DPTR, A	F0	
4106		MOV A,#18	74,18	make ALE High
4108		MOVX @DPTR, A	F0	
4109		MOV DPTR, #FFD0	90,FF,D0	
410C		MOV A,#01	74,01	SOC signal High
410E		MOVX @DPTR, A	F0	
411F		MOV A,#00	74,00	SOC signal low
4111		MOVX @DPTR, A	F0	
4112		MOV DPTR, #FFD8	90,FF,D8	
4115	WAIT	MOVX A,@DPTR	E0	
4116		JNB E0,WAIT	30,E0,FC	Check for EOC
4119		MOV DPTR,#FFC0	90,F,C0	Read ADC data
411C		MOVX A,@DPTR	E0	
411D		MOV DPTR,#4150	90,41,50	Store the data in memory location
4120		MOVX @DPTR, A	F0	
	HERE	SJMP HERE	80,FE	

OUTPUT:

ANALOG VOLTAGE	DIGITAL DATA ON LED DISPLAY									HEX CODE IN LOCATION 4150
	D7	D6	D5	D4	D3	D2	D1	D0	OUTPUT	

RESULT:

Thus the assembly language program to convert an analog signal into a digital has been executed successfully and the digital data was stored at desired location.

EXP NO:	INTERFACING OF DAC
DATE	

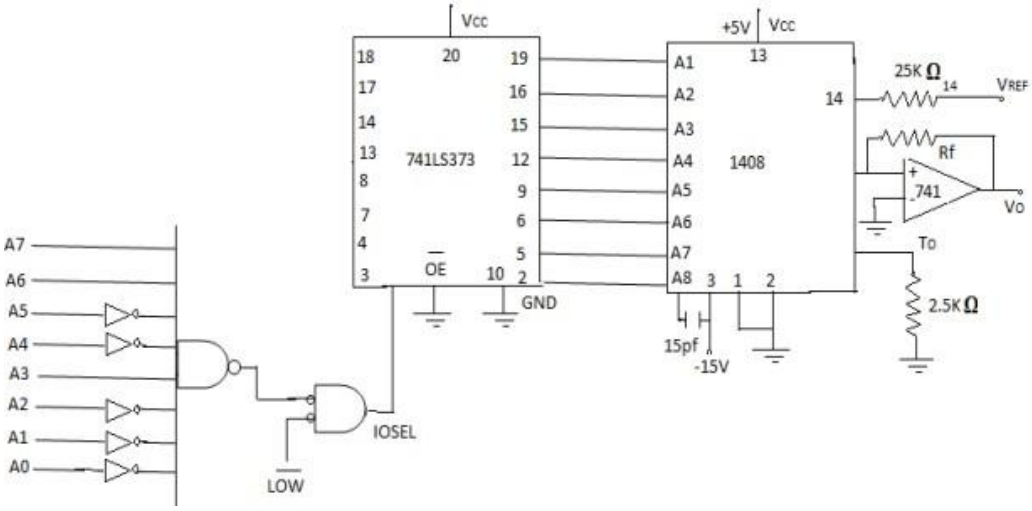
AIM:

To interface the DAC with 8051 microcontroller and generate the square wave, saw tooth wave and triangular wave.

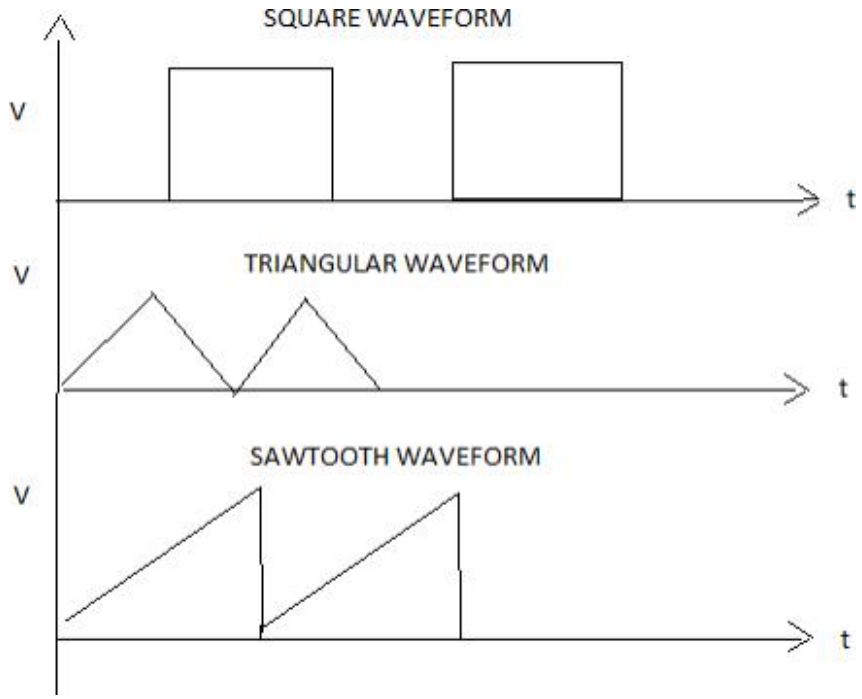
REQUIREMENTS:

S.No	Hardware & Software Requirements	Quantity
1	8051 Trainer Kit	1No
2	Power Chord	1No
3	DAC interfacing board	1 No
4	CRO	1 No

CIRCUIT DIAGRAM:



WAVEFORMS:



SQUARE WAVE

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MOV DPTR, # E0C0	90,FF,C8	Move the immediate Data EOCO
4103	START	MOV A, # 00	74,00	Initialize the Accumulator To zero
4105		MOVX @ DPTR, A	F0	Long call the delay
4106		LCALL DELAY	12,41,12	Move the content of Accumulator to FF
4109		MOV A, # FF	74,FF	Long call delay
410B		MOVX @ DPTR, A	F0	Long jump to start
410C		LCALL DELAY	12,41,12	Move the 05 data To R register
410F		LJMP START	02,41,03	Decrement Jump NON zero
4112	DELAY	MOV R1, # 05	79,05	Return to main program
4114	LOOP	MOV R2, # FF	74,FF	Short jump to start
4116	HERE	DJNZ R2, HERE	DA,FE	
4118		DJNZ R1, LOOP	D9,FA	
411A		RET	22	
411C		SZMP START	80,E3	

OBSERVATION:

AMPLITUDE	TIME PERIOD

TRIANGULAR WAVE

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MOV DPTR, # FFC8	90,FF,C8	Initialize the DPTR Content. Initialize Accumulator value is zero Increment accumulator Content Move the content Of Accumulator FF Decrement the Accumulator Content Long Jump to Start
4103		MOV A, # 00	74,00	
4105	LOOP	MOVX @ DPTR, A	F0	
4106		INC A	04	
4107		JNZ LOOP1	70,FC	
4109		MOV A, # FF	74,FF	
410B	LOOP2	MOVX @ DPTR, A	F0	
410C		DEC A	14	
410D		JNZ LOOP2	70,FC	
410F		LJMP START	02,41,03	

OBSERVATION:

AMPLITUDE	TIME PERIOD

SAWTOOTH

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		MOV DPTR, # FFC8	90,FF,C8	
4103		MOV A, # 00	74,00	
4105	LOOP	MOVX @ DPTR, A	F0	
4106		INC A	04	
4107		JNZ LOOP	80,FC	

OBSERVATION:

AMPLITUDE	TIME PERIOD

RESULT:

Thus the digital to analog converter is interfaced with microcontroller and generate the square wave, saw tooth & triangular wave.

APPENDIX-I

8085 INSTRUCTION SET

Following is the table showing the list of Control instructions with their meanings.

Opcode	Operand	Meaning	Explanation
NOP	None	No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT	None	Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI	None	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI	None	Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM	None	Read interrupt mask	This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
SIM	None	Set interrupt mask	This instruction is used to implement the interrupts 7.5, 6.5, 5.5, and serial data output.

The following table shows the list of Logical instructions with their meanings.

Opcode	Operand	Meaning	Explanation
CMP	R M	Compare the register or memory with the accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator.
CPI	8-bit data	Compare immediate with the accumulator	The second byte data is compared with the contents of the accumulator.
ANA	R M	Logical AND register or memory with the accumulator	The contents of the accumulator are logically AND with M the contents of the register or memory, and the result is placed in the accumulator.
ANI	8-bit data	Logical AND immediate with the accumulator	The contents of the accumulator are logically AND with the 8-bit data and the result is placed in the accumulator.

XRA	R M	Exclusive OR register or memory with the accumulator	The contents of the accumulator are Exclusive OR with M the contents of the register or memory, and the result is placed in the accumulator.
XRI	8-bit data	Exclusive OR immediate with the accumulator	The contents of the accumulator are Exclusive OR with the 8-bit data and the result is placed in the accumulator.
ORA	R M	Logical OR register or memory with the accumulator	The contents of the accumulator are logically OR with M the contents of the register or memory, and result is placed in the accumulator.
ORI	8-bit data	Logical OR immediate with the accumulator	The contents of the accumulator are logically OR with the 8-bit data and the result is placed in the accumulator.
RLC	None	Rotate the accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.
RRC	None	Rotate the accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.
RAL	None	Rotate the accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
RAR	None	Rotate the accumulator right through carry	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
CMA	None	Complement accumulator	The contents of the accumulator are complemented. No flags are affected.
CMC	None	Complement carry	The Carry flag is complemented. No other flags are affected.
STC	None	Set Carry	Set Carry

The following table shows the list of Branching instructions with their meanings.

<table>
<tr><th colspan="3">Opcode</th><th>Operand</th><th>Meaning</th><th>Explanation</th></tr>
<tr><td colspan="3">JMP</td><td>16-bit address</td><td>Jump unconditionally</td><td>The program sequence is transferred to the memory address given in the operand.</td></tr>
<tr><td colspan="3"><table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>JC</td><td>Jump on Carry</td><td>CY=1</td></tr><tr><td>JNC</td><td>Jump on no Carry</td><td>CY=0</td></tr><tr><td>JP</td><td>Jump on positive</td><td>S=0</td></tr><tr><td>JM</td><td>Jump on minus</td><td>S=1</td></tr><tr><td>JZ</td><td>Jump on zero</td><td>Z=1</td></tr><tr><td>JNZ</td><td>Jump on no zero</td><td>Z=0</td></tr><tr><td>JPE</td><td>Jump on parity even</td><td>P=1</td></tr><tr><td>JPO</td><td>Jump on parity odd</td><td>P=0</td></tr></table></td><td>16-bit address</td><td>Jump conditionally</td><td>The program sequence is transferred to the memory address given in the operand based on the specified flag of the PSW.</td></tr>
<tr><td colspan="3"><table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>CC</td><td>Call on Carry</td><td>CY=1</td></tr><tr><td>CNC</td><td>Call on no Carry</td><td>CY=0</td></tr><tr><td>CP</td><td>Call on positive</td><td>S=0</td></tr><tr><td>CM</td><td>Call on minus</td><td>S=1</td></tr></table></td><td>16-bit address</td><td>Unconditional subroutine call</td><td>The program sequence is transferred to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack.</td></tr>
</table>

<table><tr><td>CZ</td><td>Call on zero</td><td>Z=1</td></tr><tr><td>CNZ</td><td>Call on no zero</td><td>Z=0</td></tr><tr><td>CPE</td><td>Call on parity even</td><td>P=1</td></tr><tr><td>CPO</td><td>Call on parity odd</td><td>P=0</td></tr></table>			
RET		None	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program.
<table><tr><td>Opcode</td><td>Description</td><td>Flag Status</td></tr><tr><td>RC</td><td>Return on Carry</td><td>CY=1</td></tr><tr><td>RNC</td><td>Return on no Carry</td><td>CY=0</td></tr><tr><td>RP</td><td>Return on positive</td><td>S=0</td></tr><tr><td>RM</td><td>Return on minus</td><td>S=1</td></tr><tr><td>RZ</td><td>Return on zero</td><td>Z=1</td></tr><tr><td>RNZ</td><td>Return on no zero</td><td>Z=0</td></tr><tr><td>RPE</td><td>Return on parity even</td><td>P=1</td></tr><tr><td>RPO</td><td>Return on parity odd</td><td>P=0</td></tr></table>		None	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW and the program execution begins at the new address.
PCHL		None	Load the program counter with HL contents	The contents of registers H & L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the loworder byte.
RST		0-7	Restart	The RST instruction is used as

software instructions in a program to transfer the program execution to one of the following eight locations.

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has additionally 4 interrupts, which can generate RST instructions internally and doesn't require any external hardware. Following are those instructions and their Restart addresses −

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

Following is the table showing the list of Arithmetic instructions with their meanings.

Opcode	Operand	Meaning	Explanation
ADD	R M	Add register or memory, to the accumulator	The contents of the register or memory are added to the contents of the accumulator and the result is stored in the accumulator. Example − ADD K.
ADC	R M	Add register to the accumulator with	The contents of the register or memory & M the Carry flag are added to the contents

		carry	of the accumulator and the result is stored in the accumulator. Example − ADC K
ADI	8-bit data	Add the immediate to the accumulator	The 8-bit data is added to the contents of the accumulator and the result is stored in the accumulator. Example − ADI 55K
ACI	8-bit data	Add the immediate to the accumulator with carry	The 8-bit data and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. Example − ACI 55K
LXI	Reg. pair, 16bit data	Load the register pair immediate	The instruction stores 16-bit data into the register pair designated in the operand. Example − LXI K, 3025M
DAD	Reg. pair	Add the register pair to H and L registers	The 16-bit data of the specified register pair are added to the contents of the HL register. Example − DAD K
SUB	R M	Subtract the register or the memory from the accumulator	The contents of the register or the memory are subtracted from the contents of the accumulator, and the result is stored in the accumulator. Example − SUB K
SBB	R M	Subtract the source and borrow from the accumulator	The contents of the register or the memory & M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. Example − SBB K
SUI	8-bit data	Subtract the immediate from the accumulator	The 8-bit data is subtracted from the contents of the accumulator & the result is stored in the accumulator. Example − SUI 55K
SBI	8-bit data	Subtract the immediate from the accumulator with borrow	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example − XCHG
INR	R M	Increment the register or the memory by 1	The contents of the designated register or the memory are incremented by 1 and their result is stored at the same place. Example − INR K
INX	R	Increment register	The contents of the designated register pair

		pair by 1	are incremented by 1 and their result is stored at the same place. Example − INX K
DCR	R M	Decrement the register or the memory by 1	The contents of the designated register or memory are decremented by 1 and their result is stored at the same place. Example − DCR K
DCX	R	Decrement the register pair by 1	The contents of the designated register pair are decremented by 1 and their result is stored at the same place. Example − DCX K
DAA	None	Decimal adjust accumulator	The contents of the accumulator are changed from a binary value to two 4-bit BCD digits. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example − DAA

Following is the table showing the list of Data-transfer instructions with their meanings.

Opcode	Operand	Meaning	Explanation
MOV	Rd, Sc M, Sc Dt, M	Copy from the source (Sc) to the destination(Dt)	This instruction copies the contents of the source register into the destination register without any alteration. Example − MOV K, L
MVI	Rd, data M, data	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. Example − MVI K, 55L
LDA	16-bit address	Load the accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. Example − LDA 2034K
LDAX	B/D Reg. pair	Load the accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. Example − LDAX K

LXI	Reg. pair, 16-bit data	Load the register pair immediate	The instruction loads 16-bit data in the register pair designated in the register or the memory. Example − LXI K, 3225L
LHLD	16-bit address	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the address into register L and copies the contents of the next memory location into register H. Example − LHLD 3225K
STA	16-bit address	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example − STA 325K
STAX	16-bit address	Store the accumulator indirect	The contents of the accumulator are copied into the memory location specified by the contents of the operand. Example − STAX K
SHLD	16-bit address	Store H and L registers direct	The contents of register L are stored in the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example − SHLD 3225K
XCHG	None	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example − XCHG
SPHL	None	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers into the stack pointer register. The contents of the H register provide the high-order address and the contents of the L register provide the low-order address. Example − SPHL

XTHL	None	Exchange H and L with top of stack	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1). Example − XTHL
PUSH	Reg. pair	Push the register pair onto the stack	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example − PUSH K
POP	Reg. pair	Pop off stack to the register pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example − POPK
OUT	8-bit port address	Output the data from the accumulator to a port with 8bit address	The contents of the accumulator are copied into the I/O port specified by the operand. Example − OUT K9L
IN	8-bit port address	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example − IN5KL

OPCODE SHEET FOR 8085 MICROPROCESSOR

HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC
CE	ACI D8	29	DAD H	7A	MOV A, D	6C	MOV L, H	0F	RRC
8F	ADC A	39	DAD SP	7B	MOV A, E	6D	MOV L, L	C7	RST 0
88	ADC B	3D	DCR A	7C	MOV A, H	6E	MOV L, M	CF	RST 1
89	ADC C	05	DCR B	7D	MOV A, L	77	MOV M, A	D7	RST 2
8a	ADC D	0D	DCR C	7E	MOV A, M	70	MOV M, B	DF	RST 3
8b	ADC E	15	DCR D	47	MOV B, A	71	MOV M, C	E7	RST 4
8c	ADC H	1D	DCR E	40	MOV B, B	72	MOV M, D	EF	RST 5
8d	ADC L	25	DCR H	41	MOV B, C	73	MOV M, E	F7	RST 6
8E	ADC M	2D	DCR L	42	MOV B, D	74	MOV M, H	FF	RST 7
87	ADD A	35	DCR M	43	MOV B, E	75	MOV M, L	C8	RZ
80	ADD B	0B	DCX B	44	MOV B, H	3E	MVI A, D8	98	SBB B
81	ADD C	1B	DCX D	45	MOV B, L	06	MVI B, D8	99	SBB C
82	ADD D	2B	DCX H	46	MOV B, M	0E	MVI C, D8	9A	SBB D
83	ACC E	3B	DCX SP	4F	MOV C, A	16	MVI D, D8	9B	SBB E
84	ADD H	F3	DI	48	MOV C, B	1E	MVI E, D8	9C	SBB H
85	ADD L	FB	EI	49	MOV C, C	26	MVI H, D8	9D	SBB L
86	ADD M	76	HLT	4A	MOV C, D	2E	MVI L, D8	9E	SBB M
C6	ADI D8	DB	IN addr8	4B	MOV C, E	36	MVI M, D8	DE	SBI D8
A7	ANA A	3C	INR A	4C	MOV C, H	00	NOP	22	SHLD addr16
A0	ANA B	04	INR B	4D	MOV C, L	B7	ORA A	30	SIM
A1	ANA C	OC	INR C	4E	MOV C, M	B0	ORA B	F9	SPHL
A2	ANA D	14	INR D	57	MOV D, A	B1	ORA C	32	STA addr16
A3	ANA F	1C	INR E	50	MOV D, B	B2	ORA D	02	STAX B
A4	ANA H	24	INR H	51	MOV D, C	B3	ORA E	12	STAX D
A5	ANA L	2C	INR L	52	MOV D, D	B4	ORA H	37	STC
A6	ANA M	34	INR M	53	MOV D, E	B5	ORA L	97	SUB A
E6	ANI D8	03	INX B	54	MOV D, H	B6	ORA M	90	SUB B
CD	CALL addr16	13	INX D	55	MOV D, L	F6	ORI D8	91	SUB C
DC	CC addr16	23	INX H	56	MOV D, M	D3	OUT addr8	92	SUB D
FC	CM addr16	33	INX SP	5F	MOV E, A	E9	PCHL	93	SUB E
2F	CMA	DA	JC addr16	58	MOV E, B	C1	POP B	94	SUB H
3F	CMC	FA	JM addr16	59	MOV E, C	D1	POP D	95	SUB L
BF	CMP A	C3	JMP addr16	5A	MOV E, D	E1	POP H	96	SUB M
B8	CMP B	D2	JNC addr16	5B	MOV E, E	F1	POP PSW	D6	SUI D8
B9	CMP C	C2	JNZ addr16	5C	MOV E, H	C5	PUSH B	EB	XCHG
BA	CMP D	F2	JP addr16	5D	MOV E, L	D5	PUSH D	AF	XRA A
BB	CMP E	EA	JPE addr16	5E	MOV E, M	E5	PUSH H	A8	XRA B
BC	CMP H	E2	JPO addr16	67	MOV H, A	F5	PUSH PSW	A9	XRA C
BD	CMP L	CA	JZ addr16	60	MOV H, B	17	RAL	AA	XRA D
BE	CMP M	3A	LDA D16	61	MOV H, C	1F	RAR	AB	XRA E
D4	CNC addr16	OA	LDAX B	62	MOV H, D	D8	RC	AC	XRA H
C4	CNZ addr16	1A	LDAX D	63	MOV H, E	C9	RET	AD	XRA L
F4	CP addr16	2A	LHLDaddr16	64	MOV H, H	20	RIM	AE	XRA M
EC	CPE addr16	01	LXI B addr16	65	MOV H, L	07	RLC	EE	XRI D8
FE	CPI D8	11	LXI D addr16	66	MOV H, M	F8	RM	E3	XTHL
E4	CPO addr16	21	LXI H addr16	6F	MOV L, A	D0	RNC		
CC	CZ addr16	31	LXISPaddr16	68	MOV L, B	CO	RNZ		
27	DAA	7F	MOV A, A	69	MOV L, C	F0	RP		
09	DAD B	78	MOV A, B	6A	MOV L, D	E8	RPE		
19	DAD D	79	MOV A, C	6B	MOV L, E	EO	RPO		

Table 1-4. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr,code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr,code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr,code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2

Hex Code	Number of Bytes	Mnemonic	Operands
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr

Hex Code	Number of Bytes	Mnemonic	Operands
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0

8051 Microcontroller Instruction Set

Hex Code	Number of Bytes	Mnemonic	Operands
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr

Hex Code	Number of Bytes	Mnemonic	Operands
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0

Hex Code	Number of Bytes	Mnemonic	Operands
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

ASCII TABLE

Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
0	00	NUL	32	20	SP	64	40	@	96	60	`	128	80	160	A0	192	C0	224	E0
1	01	SOH	33	21	!	65	41	A	97	61	a	129	81	161	A1	193	C1	225	E1
2	02	STX	34	22	"	66	42	B	98	62	b	130	82	162	A2	194	C2	226	E2
3	03	ETX	35	23	#	67	43	C	99	63	c	131	83	163	A3	195	C3	227	E3
4	04	EOT	36	24	$	68	44	D	100	64	d	132	84	164	A4	196	C4	228	E4
5	05	ENQ	37	25	%	69	45	E	101	65	e	133	85	165	A5	197	C5	229	E5
6	06	ACK	38	26	&	70	46	F	102	66	f	134	86	166	A6	198	C6	230	E6
7	07	BEL	39	27	'	71	47	G	103	67	g	135	87	167	A7	199	C7	231	E7
8	08	BS	40	28	(	72	48	H	104	68	h	136	88	168	A8	200	C8	232	E8
9	09	HT	41	29	)	73	49	I	105	69	i	137	89	169	A9	201	C9	233	E9
10	0A	LF	42	2A	*	74	4A	J	106	6A	j	138	8A	170	AA	202	CA	234	EA
11	0B	VT	43	2B	+	75	4B	K	107	6B	k	139	8B	171	AB	203	CB	235	EB
12	0C	FP	44	2C	'	76	4C	L	108	6C	l	140	8C	172	AC	204	CC	236	EC
13	0D	CR	45	2D	-	77	4D	M	109	6D	m	141	8D	173	AD	205	CD	237	ED
14	0E	SO	46	2E	.	78	4E	N	110	6E	n	142	8E	174	AE	206	CE	238	EE
15	0F	SI	47	2F	/	79	4F	O	111	6F	o	143	8F	175	AF	207	CF	239	EF
16	10	DLE	48	30	0	80	50	P	112	70	p	144	90	176	B0	208	D0	240	F0
17	11	DC1	49	31	1	81	51	Q	113	71	q	145	91	177	B1	209	D1	241	F1
18	12	DC2	50	32	2	82	52	R	114	72	r	146	92	178	B2	210	D2	242	F2
19	13	DC3	51	33	3	83	53	S	115	73	s	147	93	179	B3	211	D3	243	F3
20	14	DC4	52	34	4	84	54	T	116	74	t	148	94	180	B4	212	D4	244	F4
21	15	NAK	53	35	5	85	55	U	117	75	u	149	95	181	B5	213	D5	245	F5
22	16	SYN	54	36	6	86	56	V	118	76	v	150	96	182	B6	214	D6	246	F6
23	17	ETB	55	37	7	87	57	W	119	77	w	151	97	183	B7	215	D7	247	F7
24	18	CAN	56	38	8	88	58	X	120	78	x	152	98	184	B8	216	D8	248	F8
25	19	EM	57	39	9	89	59	Y	121	79	y	153	99	185	B9	217	D9	249	F9
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	186	BA	218	DA	250	FA
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{	155	9B	187	BB	219	DB	251	FB
28	1C	FS	60	3C	<	92	5C	\	124	7C	|	156	9C	188	BC	220	DC	252	FC
29	1D	GS	61	3D	=	93	5D	]	125	7D	}	157	9D	189	BD	221	DD	253	FD
30	1E	RS	62	3E	>	94	5E	^	126	7E	~	158	9E	190	BE	222	DE	254	FE
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL	159	9F	191	BF	223	DF	255	FF

APPENDIX - C

HEX CONVERSION TABLE

EX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440