ANNAMALAI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BE (CSE) - V SEMESTER
CSCP509-MICROPROCESSOR LAB
LIST OF EXPERIMENTS

Exp. No.	**CYCLE-I**
1.	Study of 8086 Microprocessor registers and instruction sets.
2 a).	Perform 16-bit Addition operation.
2 b).	Perform 16-bit Subtraction operation.
3 a).	Perform 16-bit Multiplication operation
3 b).	Perform 16-bit Division operation
4.	Calculate the length of a string.
5.	Find the sum of the numbers in a word array.
6.	Sorting number in ascending order of an unsorting array
7.	Move a byte String from source to destination.
CYCLE- II	
8.	Square wave generation using 8253IC.
9.	Stepper motor interface using 8255IC.
10.	Data transfer using USART.
11.	Message display 8279IC.
12.	Simulation of traffic light control signal.

Lab Incharges : Dr.P.Sudhakar (A1-Batch), Dr.G.Prabakaran (A2-Batch)

Dr.M.Arulselvi (B1- Batch), Dr.S.Mohan (B2- Batch)

Ex. No : 1 **STUDY OF 8086 MICROPROCESSOR**
Date : **REGISTERS AND INSTRUCTION SETS**

AIM:

To study 8086 microprocessor registers in 8086 and its instruction set.

DESCRIPTION:

The Intel 8086 is a 16-bit microprocessor used as a CPU in a microcomputer it has a 16-bit data bus. So that it can read 16 data from (or) write 16 data to memory. The 8086 has a 20-bit address bus and can address any one of the 2^{20} memory location words will be stored in bus consecutive memory location. If the 1st byte of a word is at an even address then the 8086 can read entire word in one operation. If the 1st byte of a word is at an odd address then the 8086 will read first byte in one operation.

INTERNAL ARCHITECTURE:

The 8086 CPU is divided into 2 independent functional part the bus interface unit and execution unit.

BUS INTERFACE UNIT:

The bus interface unit sends out addresses, fetches instructions from memory, reads data from ports and memory and writes data into ports and memory. It handles all transfer of data and addresses on the buffer for execution.

SEGMENT REGISTERS:

The BIU contains four 16-bit segment registers. They are

1. The code segment register.
2. The stack segment register.
3. The extra segment register.
4. The data segment register.

QUEUE:

To speed up program execution the BIU fetches as many as six instructions ahead of time from memory. The projected instruction bytes are held for the EU in a queue fetching the next instruction while the current instruction execution is called pipelining.

INSTRUCTION POINTER:

This register holds the 16-bit address of the next code byte within this code segment. The value contained in the instruction pointer is called OFFSET.

EXECUTION UNIT:

The EU of the 8086 tells the BIU where to fetch the instruction or data from decoder instruction to execute the instruction. The component of the EU are control circuitry, instruction decoder and ALU. The EU has 16-bit ALU which can Add, Subtract, AND, OR, XOR and INC etc.

FLAG REGISTER:

A 16-bit flag register in the EU contains active flags. A flag is a Flip-flop, which indicates some condition, produced by the execution of an instruction.

They are,

(i) C-Carry (ii) A-Auxiliary carry (iii) S-Sign (iv) P-Parity (v) Z-Zero

(vi) T-Trap (vii) I-Interrupt (viii) D-Direction (ix) O-Overflow.

GENERAL PURPOSE REGISTERS (GPR):

The EU has 8 general-purpose registers to store 8-bit data. They can be combined to store 16-bit data for 8-bit data storage; the registers are AH, AL, BH, BL, CH, CL, DH, and DL. For 16-bit data the registers are AX, BX, CX, DX, where AX is the accumulator.

POINTER AND INDEX REGISTER:

The EU register contains 16-bit source index (SI) 16-bit destination index registers (DI) and 16-bit base pointer registers. These can be used for temporary storage of data. But their main use is to hold the 16-bit offset of a data word in one of the registers.

ADDRESSING MODES:

The way in which the operand is specified is called an addressing mode. The addressing modes supported by 8086 are:

1. Immediate addressing mode
2. Direct addressing mode
3. Register addressing mode
4. Register indirect addressing mode
5. Register relative addressing mode
6. Based index addressing mode
7. Relative based indexed addressing mode.

8086 INSTRUCTION SET:

After we get the structure of a program worked and written down, the next step is to determine the instruction statements required to do each part of the program.

DATA TRANSFER INSTRUCTIONS:

MOV : Copy byte from specified source to destination.

PUSH : Copy specified used to top of stack.

POP : Copy word from top of stack.

XCHG : Exchange bytes.

XLAT : Translate a byte in ALU using a table a memory.

INPUT/OUTPUT PORT TRANSFER INSTRUCTIONS:

IN Copy a byte from port to accumulator.

OUT Copy a byte from accumulator to port.

SPECIAL ADDRESS TRANSFER INSTRUCTIONS:

LEA Load effective address of operand into register.

LDS Load DS register and other register from memory.

LES Load ES register and other register from memory.

FLAG TRANSFER INSTRUCTIONS:

LAHF Load alt with the low byte of the register.

SAHF Store alt register to low byte of flag register.

ARITHMETIC INSTRUCTIONS:

ADD Add specified byte to byte.

ADC Add byte+byte+carry flag.

INC Increment specified byte.

SUBTRACTION INSTRUCTIONS:

SUB Subtract byte from byte.

SBB Subtract byte and carry flag from byte.

CMP Compare two specified bytes or two specified words.

MULTIPLICATION INSTRUCTIONS:

MUL Multiply unsigned byte by byte.

IMUL Multiply signed byte by byte.

DIVISION INSTRUCTIONS:

DIV Divide unsigned byte by byte.

IDIV Divide signed byte by byte.

LOGICAL INSTRUCTIONS:

NOT Invert each bit of a word.

AND AND each bit in a byte with the cell. But in another byte.

OR OR each bit in a byte with the cell. But in another byte.

SHIFT INSTRUCTIONS:

SHL/SAL Shift bit of word left, put zeroes in LSB.

SHR Shift bit of byte right, put zeroes in MSB.

SAR Shift bits of word right, copy old MSB to LSB.

ROTATE INSTRUCTIONS:

ROL Rotate bits of byte, left MSB to LSB and CF.

ROR Rotate bits of byte, right, MSB to LSB and CF.

STRING INSTRUCTIONS:

REP An instruction prefix. Repeat following instruction until CX=0.

REPE An instruction prefix. Repeat following instruction until CX=0,

and ZF=/!.

OUTS/OUTSB/OUTSW Output string byte or word to port.

PROGRAM EXECUTION TRANSFER INSTRUCTIONS:

(i) Uncondition Instruction Transfer:

CALL Call a procedure save return address on stack.

RET Return from procedure to calling program.

JMP Goto specified address to get next instruction.

(ii) Conditional Transfer Instructions:

JA Jump if above.

JAE Jump if above or equal.

JBE Jump if below or equal.

JC Jump if carry.

JE Jump if equal.

JNC Jump if no carry.

ITERATION CONTROL INSTRUCTIONS:

These instructions can be used to execute a series of instructions and number of iterations that recall the specified instruction.

LOOP Loop three a sequence of instruction until LX=0.

JNZ Jump to specified address if CX=0.

INTERRUPT INSTRUCTIONS:

INT Interrupt program execution call service procedure.

INTO Interrupt program execution if OF=1.

HIGH LEVEL LANGUAGE INTERFACE INSTRUCTION:

ENTER Enter procedure.

LEAVE Leave procedure.

BOUND Check if effective address within specified array bound.

EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS:

HLT: Halt until interrupt or reset.

WAIT: Wait until signal on the test pin.

ESC: Escape to external coprocessor.

RESULT:

Thus, the internal architecture and instruction set of 8086 microprocessor registers and instruction sets have been studied.

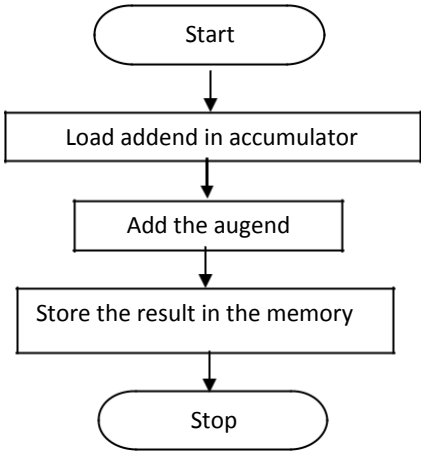
EXP NO:2 a) PERFORM 16-BIT ADDITION OPERATION

Date:

Aim:
To perform 16-bit addition operation using 8086 microprocessor kit

Theory:
Move the first data to accumulator. Then using the add instruction 16 bit addition is performed.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
A2 dw 1102h
A3 dw 1104h
datahere ends
codehere segment
assume cs:codehere, ds:datahere
ORG 1000h
MOV AX, [A1]
ADD AX, [A2]
MOV [A3], AX
HLT
codehere ends
end
```

Opcode Table:

Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 11	MOV AX,[A1]	Addend in AX
1003 1004 1005 1006	03 06 02 11	ADD AX,[A2]	Add
1007 1008 1009	A3 00 12	MOV [A3],AX	Store the result
100A	F4	HLT	Halt

Sample Data:

Input: **Output:**

[1100] =11 [1200]=33

[1101]=11 [1201]=33

[1102]=22

[1103]=22

Result:
The 16-bit Arithmetic operation for addition is performed using 8086 microprocessor kit.

Ex. No:2(b) **Perform 16-BIT SUBTRACTION OPERATION**
Date:

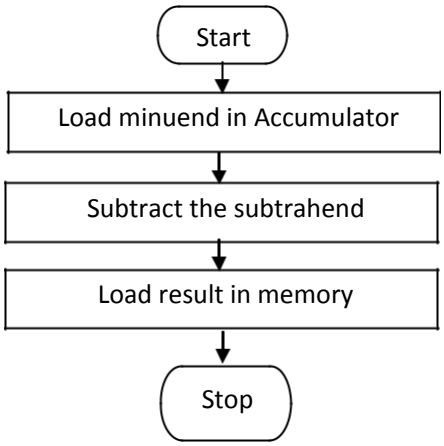
Aim:

To Perform 16-bit subtraction operation using 8086 microprocessor kit.

Theory:

Move the minuend to a register pair. Then using the sub instruction 16 bit subtraction is performed.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
A2 dw 1102h
A3 dw 1104h
datahere ends
codehere segment
assume CS: codehere, ds:datahere
org 1000h
MOV AX, [A1]
SUB AX, [A2]
MOV [A3], AX
HLT
codehere ends
end
```

Opcode Table:

Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 11	MOV AX,[A1]	Minuend in AX
1003 1004 1005 1006	2B 06 02 11	SUB AX,[A2]	Subtract
1007 1008 1009	A3 00 12	MOV [A3],AX	Store the result
100A	F4	HLT	Halt

Sample Data:
Input: Output:

[1100] =33 **[1200]=11**

[1101]=33 **[1201]=11**

[1102]=22

[1103]=22

Result:
The 16-bit Arithmetic operation for subtraction is performed using 8086 microprocessor kit

Ex.N0. 3(a) **PERFORM 16-BIT MULTIPLICATION OPERATION**

Date:

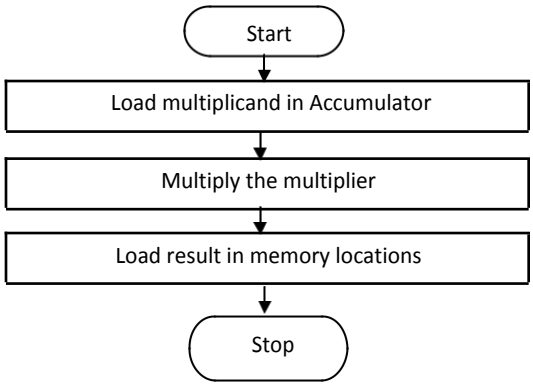
Aim :

To Perform 16-bit multiplication operation using 8086 microprocessor kit.

Theory:

Assign multiplicand and multiplier to memory location. Move the multiplicand to register AX. Multiply with multiplier. Store the product in AX and DX

Flowchart:



MASM code:

```
datahere segment
a dw 1500h
b dw 1502h
c dw 1504h
d dw 1506h
datahere ends
codehere segment
assumecs: codehere, ds:datahere
ORG 1000h
MOV AX,[a]
MUL [b]
MOV [c],AX
MOV[d],DX
HLT
codehere ends
End
```

Opcode Table:

Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 15	MOV AX,[a]	Move content of memory to accumulator
1003 1004 1005 1006	F7 26 02 15	MUL [b]	Multiply the memory content to accumulator
1007 1008 1009	A3 04 15	MOV [c],AX	Move accumulator to memory
100A 100B 100C 100D	89 16 06 15	MOV[d],DX	Move DX content to Memory
100E	F4	HLT	Halt the program

Sample Data :

Input:

1500 – 03h

1501 – 00h

1502 – 02h

1503 – 00h

OUTPUT :

1504 – 06h

1505 – 00h

RESULT:

Thus the 16-bit multiplication is performed using microprocessor.

Ex. No:3(b) **PERFORM 16-BIT DIVISION OPERATION**

Date:

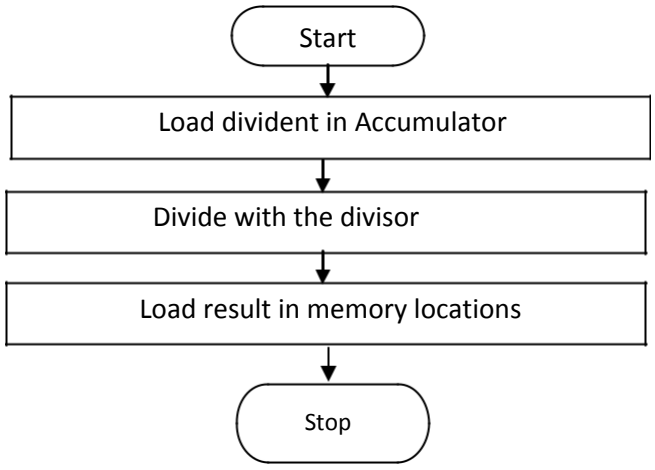
AIM :-

To perform 16-bit division operation using 8086 micropressor kit.

Theory:-

Assign dividend and divisor to memory location. Move the dividend to register AX. Divide using divisor. Store the quotient and remainder in AX and DX.

FLOWCHART:



MASM code :-

```
data segment
a dw1600h
b dw 1602h
c dw 1604h
d dw 1606h
datahere ends codehere segment
assumecs:codehere,ds:datahere
org 2010h

 MOV AX ,[a]
DIV [b]
MOV[c],AX
MOV[d],DX
HLT.
Codehere ends
end
```

OPCODE TABLE:

Address	Opcode	Mnemonics	Comments
2010 2011 2012	A1 00 16	MOV AX,[a]	Move the content of memory to accumulator
2013 2014 2015 2016	F7 36 02 16	DIV [b]	Divide the content of memory with accumulator
2017 2018 2019	A3 04 16	MOV[c],AX	Move accumulator to memory
201A 201B 201C 201D	89 16 06 16	MOV[d],DX	Move the DX content to memory
201E	F4	HLT	Halt the program

SAMPLE DATA :

INPUT :

1600 – 08h

1601 – 00h

1602 – 02h

1603 – 00h

OUTPUT :-

16O4 – 04h

1605 – 00h

RESULT :-

Thus the 16-bit division is performed using microprocessor

Ex. No:4 **Calculate the length of a string**
Date:

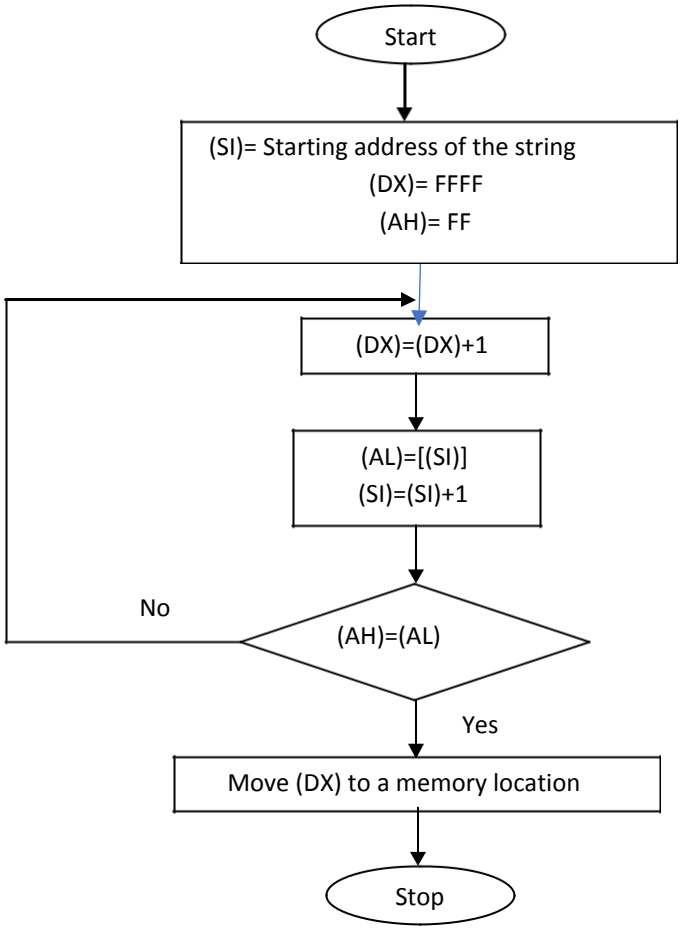
Aim :

To find the number of characters in a string.

Theory :

Addressing the string is done using SI register, and the DX register is used to store the number of characters. End of string is detected using FF. Hence each character is detected using FF.Hence each character is fetched from memory and is compared with FF. If the zero flag is set, then it denotes end of string, the count have been stored in DX, by incrementing it after each comparison.

Flowchart:



MASM code:

```
Datahere segment
 A1 dw 1100h
Datahere segment ends
Codehere segment
Assume cs:codehere , ds: datahere
        ORG 1000h
        MOV SI,1200h
        MOV DX,0FFFFh
        MOV AH,0FFh
L1      INC DX
         MOV AL,[SI]
         INC SI
         CMP AH,AL
         JNZ  L1
         MOV [A1],DX
         HLT
Codehere ends
 End
```

Opcode Table:

Address	Opcode	Mnemonics	Comments
1000	BE	MOV SI,1200	Load the starting
1001	00		Address of the string in SI
1002	12		
1003	BA	MOV DX,FFFF	Initialize DX
1004	FF		
1005	B4	MOV AH,FF	Load AH with end of the string
1006	FF		

1007	42	NOTEND: INC DX	Increment count
1008	8A	MOV AL,[SI]	Get string character to AL
1009	04		
100A	46	INC SI	Increment String index
100B	38	CMP AH,AL	Compare string with FF
100C	C4		
100D	75	JNZ NOTEND	Jump if not end
100E	F8		
100F	89		
1011	16	MOV [1200],DX	Store the length
1012	00		
1013	11		
1014	F4	HLT	Halt the process

Sample data :

Input :

1200 28

1201 13

1202 10

1203 11

1204 FF

Output: 1100 04

Result:

Thus the program to find the number of character in a string is executed and verified.

Ex. No: 5 FIND THE SUM OF THE NUMBERS IN A WORD ARRAY
DATE:

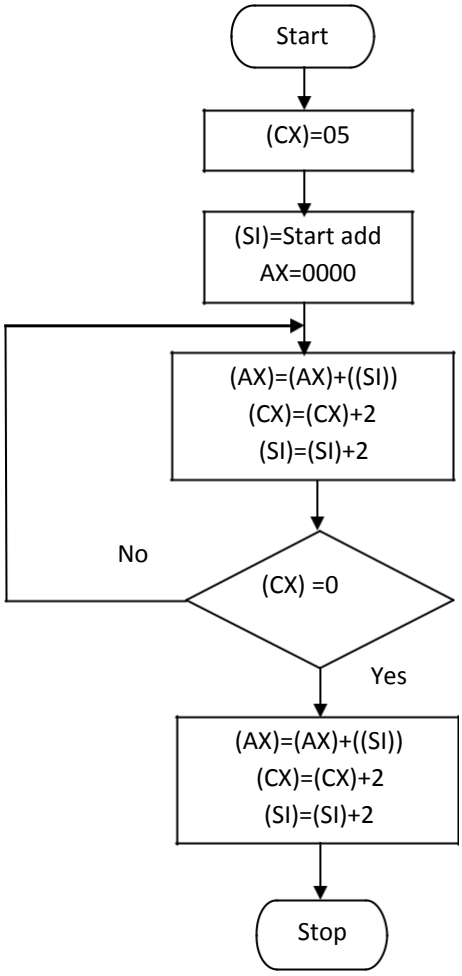
Aim:

To obtain the sum of a 16-bit array in memory, using index register and store the result in memory.

Theory:

Initialize the index register SI with the start address and CX with the length of the array. Clear the accumulator and add the contents of SI in it. Increment the index to point to the next word and decrement CX by 1. Repeat until CX=0 and store the sum in a memory location.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
Sum dw 1200h
datahere ends
codehere segment
assume CS: codehere, DS: datahere
                ORG 1000h
                MOV CX, 05h
                MOV AX, 0
                MOV SI,AX
        L1:     ADD AX, START[A1]
                ADD SI, 2
                LOOP L1
                MOV [Sum], AX
                HLT
  Codehere ends
   end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Remarks
1000 1001 1002	B9 05 00	MOV CX, 05h	CX= 5 No of elements
1003 1004 1005	B8 00 00	MOV AX, 0	Clear AX
1006 1007	8B F0	MOV SI,AX	Initialize SI to start of the array
1008 1009 100A 100B	03 84 00 11	L1: ADD AX, START[A1]	Add accumulator and content of Array
100C 100D 100E	83 C6 02	ADD SI,2	

100F 1010	E2 F7	LOOP L1	Decrement CX and check if zero
1011 1012 1013	A3 00 12	MOV [Sum],AX	Store the result
1014	F4	HLT	Halt the process

Sample Input and output:

Input

Number of Elements = 5

[1100] = 0001

[1102] = 0002

[1104] = 0003

[1106] = 0004

[1108] = 0005

Output : [1200] = 000F

RESULT:

Thus, the sum of the numbers in a word array is obtained.

Ex. No: 6 SORTING NUMBER IN ASCENDING ORDER OF AN UNSORTED ARRAY

DATE:

Aim:

To arrange an array of unsorted words in ascending order.

Theory:

The algorithm used here is bubble sort. Let

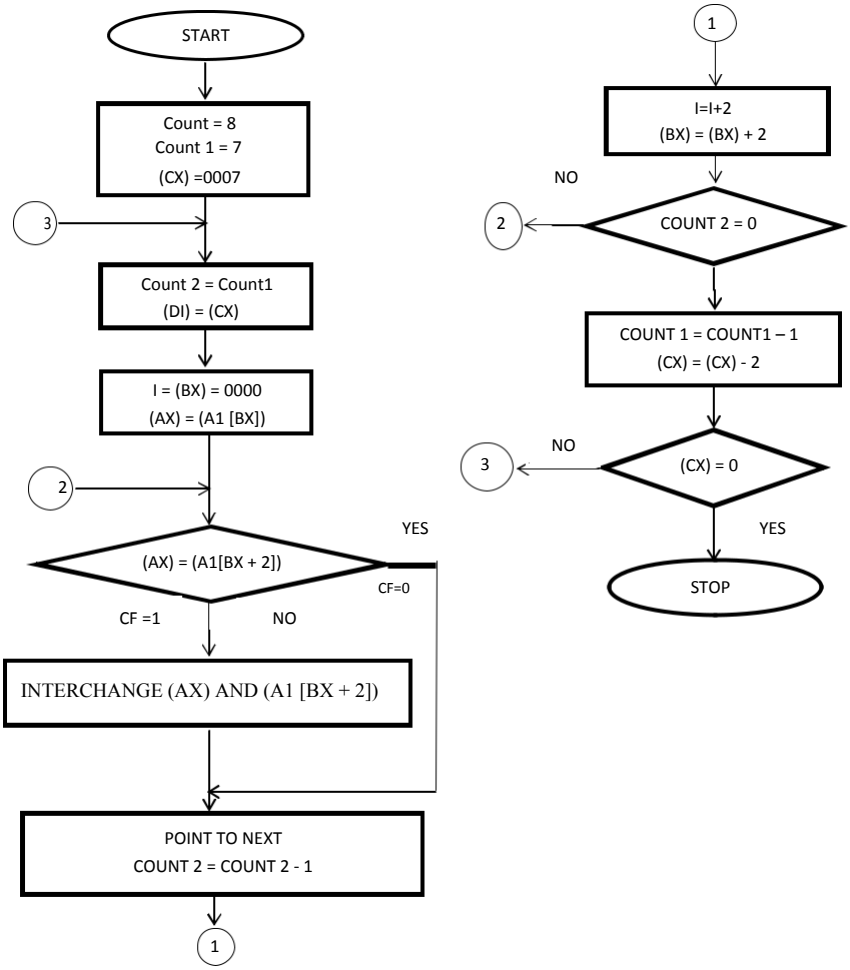
N : Number of elements in the array, content of CX.

A1 : Array name of start address of array.

I : Index in this array, here content of DI.

Start at the Beginning of the array, and considering a pair of elements at a time, put the pair in descending order. Thus arrange successive pairs of elements in descending order.After the first pass through the array the smallest element in at the end of the array :Hence during the second pass consider only the first N – 1 element and so on.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
datahere ends
codehere segment
assume CS: codehere, DS: datahere
                org 1000h
                MOV CX, 07h
L1:             MOV D1, CX
                MOV BX, 00h
L2:             MOV AX, A1 [BX]
                CMP AX, A1 [BX + 2]
                JNC PROCEED
                XCHG AX, A1 [BX + 2]
                MOV A1 [BX], AX
PROCEED:        ADD BX, 2
                LOOP L2
                NOP
                MOV CX, DI
                LOOP L1
                HLT
Codehere ends

End
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Remarks
1000 1001 1002	B9 07 00	MOV CX, 07	CX= count -1
1003 1004	8B F9	L1: MOV D1, CX	Save CX in DX
1005 1006 1007	BB 00 00	MOV BX,0	Clear BX
1008 1009 100A	8B 87 00	L2: MOV AX, A1[BX]	

100B	11		
100C 100D 100E 100F	3B 87 02 11	CMP AX, A1 [BX+ 2]	Compare first to elements
1010 1011	73 08	JNC PROCEED	
1012 1013 1014 1015	87 87 02 11	XCHG AX, A1 [BX+ 2]	Interchange if less
1016 1017 1018 1019	89 87 00 11	MOV A1 [BX], AX	
101A 101B 101C	83 C3 02	PROCEED:ADD BX, 2	Increment index
101D 101E	E2 E9	LOOP L2	And proceed if not
101F	90	NOP	
1020 1021	8B CF	MOV CX, D1	Move a copy of count in CX
1022 1023	E2 DF	LOOP L1	Repeat until CX=0
1024	F4	HLT	Halt the process

Sample Input and output:

Number of Elements = 8

Unsorted array Starting from A1 = 1100

[1100] = 0022

[1102] = 0011

[1104] = 0033

[1106] = 0077

[1108] = 0055

[110A] = 0066

[110C] = 0088

[110E] = 0044

Sorted array :

[1100] = 0011

[1102] = 0022

[1104] = 0033

[1106] = 0044

[1108] = 0055

[110A] = 0066

[110C] = 0077

[110E] = 0088

RESULT:

Thus, an array of unsorted words is arranged in ascending order.

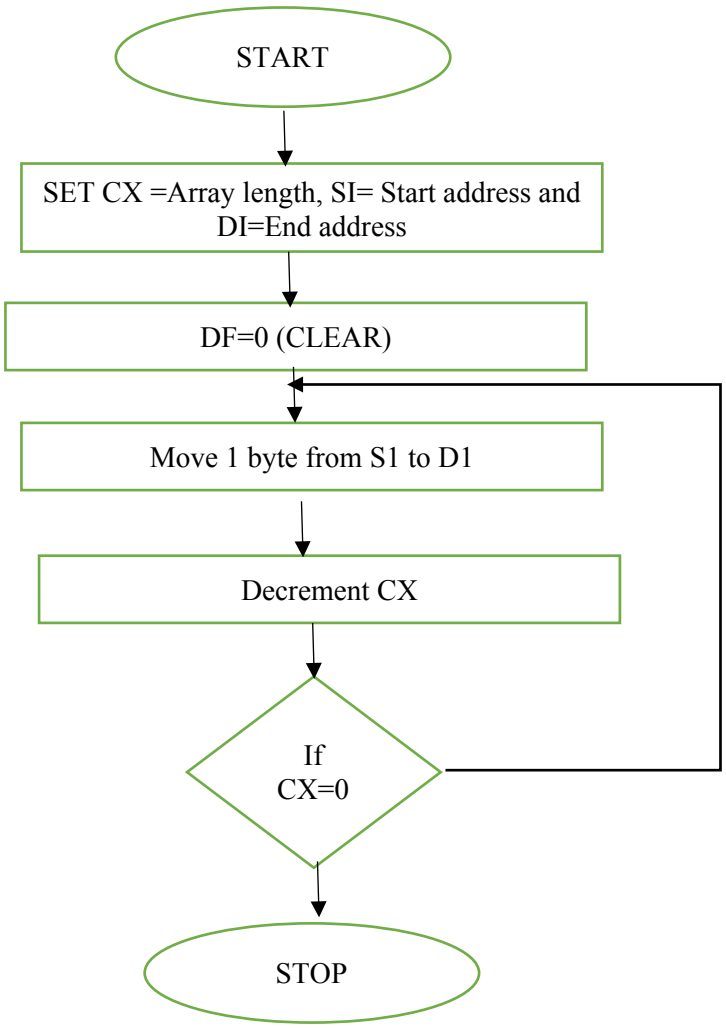
EX. No. 7. MOVE A BYTE STRING FROM SOURCE TO DESTINATION

AIM : To Move a Byte String of length FF from a source to a destination.

THEORY:

String primitives require initialization of the index registers and SI and DI registers are initialized to start of the source and start of the destination array respectively. The direction flag is cleared to facilitate auto incrementing of the index registers. The CX register is used to perform the operation repeatedly. The string primitive used in MOVSB which moves one byte from source operand to destination operand. The SI and DI registers are incremented automatically as DE=0.

FLOW CHART:



MASM Code:

```
Datahere segment
SOU dw 100Eh
DES dw 110Eh
Datahere ends
Codehere segment
```

```
        Assume CS: codehere, DS: datahere
        ORG 1000 h
        MOV SI, [SOU]
        MOV DI, [DES]
        MOV CX, 0FFH
        CLD
MOVE : MOVSB
        Loop  MOVE
        HLT
```

1000 1001 1002	BE 0E 10	MOV SI, [SOU]	Move source address to SI
1003 1004 1005	BF 0E 11	MOV DI,[DES]	MOVE destination address to DI
1006 1007 1008	B9 FF 00	MOV CX, 0Ah	CX=Count=10
1009	FC	CLD	Clear the destination flag
100A	A4	MOVE: MOVSB	CX=Count=10
100B 100C	E2 FD	Loop MOVE	Repeat Until CX=0
100D	F4	HLT	Halt the process

SAMPLE DATA:

INPUT:

Fill the location from 100E to 10 locations with 55

S -Array=100E

D-Array= 110 E

OUTPUT:

[100E] to [110E]= 55

RESULT:

Thus the program to move string from a source to a destination is executed and verified.

EX.NO:8 SQUARE WAVE GENERATION USING 8253IC

Date:

AIM:

To generate a square wave frequency 125 KHz from channel 0 using 8253 IC.

ALGORITHM:

1. Start the process.

2. Move the control word to acc and it in the control.

3. Register to initialize 8253A.

4.Move the LSB of count to channel 0.

5. Stop.

SOURCE CODE:

```
MOV AL, 36H

OUT 16H, AL

MOV AL, DAH

OUT 10H, AL

MOV AL,00H

OUT 10H,AL

HLT
```

CONTROL WORD FORMAT:

Sc_1	Sc_0	RW_1	RW_0	M_2	M_1	M_0	BCD	
0	0	1	1	0	1	1	0	= 36

Sc_0, Sc_1→counter

selection RW_1, RW_0

→read/write

M_2 M_1 M_0→ Mode selection

BCD → Counter type, BCD or Binary (Binary 16 bit counter)

MODE SELECTION:

Mode (0) – Interrupt terminal

Mode (1) – H/W one short

Mode (2) – pulse generator

Mode(3) – square wave

generator Mode(4) – S/W

trigged store Mode(5) – H/W

trigged store

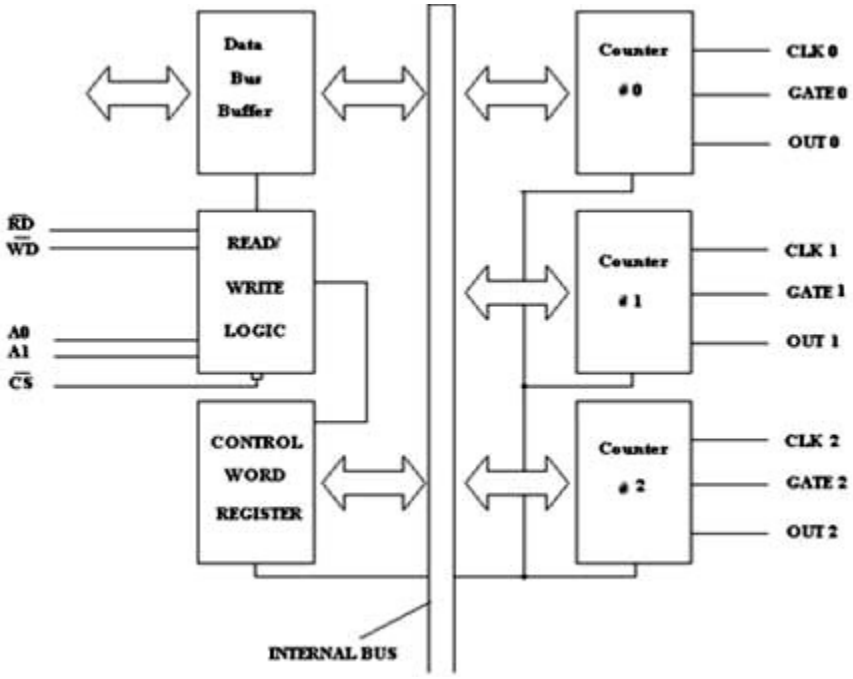
M_2	M_1	M_0
0	0	0
0	0	1
X	1	0
X	1	1
1	0	0
1	0	1

OPCODE TABLE:

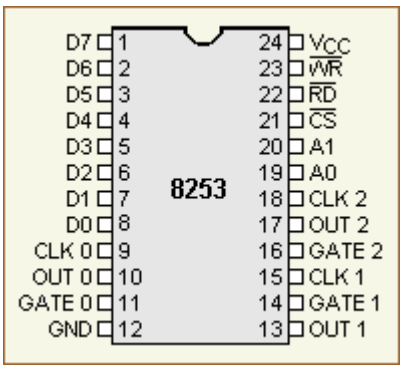
ADDRESS	OPCODE	MNEMONICS	COMMENTS
2000	B0	MOV AL,36H	move 36H to AL
2001	36		
2002	E6	OUT 16H ,AL	Give data to output
2003	16		
2004	B0	MOV AL, 0A	Move 0A to AL
2005	0A		
2006	E6	OUT 10H , AL	Give data to output
2007	10		
2008	B0	MOV AL ,00H	Move 00H to AL
2009	00		
200A	E6	OUT 10H , AL	Give data to output
200B	10		
200C	F4	HLT	Halt

DESCRIPTION:

$D_2 - D_0$	Data bus (8-bit)
CLKN	Counter clock inputs
GATE IN	Counter gate inputs
OUT N	Counter output
$\overline{\square\square}$	RD counter
$\overline{\square\square}$	Write command or data
$A_0 - A_1$	Counter select
VCC	+5 volt
GND	Ground



Block Diagram of 8253 IC



Pin Diagram of 8253 IC

SAMPLE CALCULATION:

Read (01 out counter – LSB only)

Mode section – mode3

Counter type – binary

Time period:

Time period=horizontal distance x time/

division Frequency = 1 / Time period

Time = (T on + T off) X Time/Division

= (2.2+2) x $20x10^{-6}$

= $84 X 10^{-6}$

Frequency = 1 / T

= $1 / 84X10^{-6}$

= 119 KHZ

Design of count:

Count=system clock frequency / Required Frequency

= 1250/125 KHZ

= 10

RESULT:

Thus the square wave of the retrieval frequency was obtained which was almost equal to observed frequency.

Ex. No : 9

Date :

STEPPER MOTOR INTERFACE USING 8255IC

AIM:

To control the movement of stepper motor using 8255IC.

ALGORITHM:

1. Initialize 8255 in the input mode.
2. Move count to CX register and it may be 4 or 8 steps.
3. Move data to AL register and output in at part A.
4. Call a delay routine and increment count.
5. Repeat 3 and 4 until count=0.
6. Repeat step through 4 infinitely.
7. Stop.

PROGRAM:

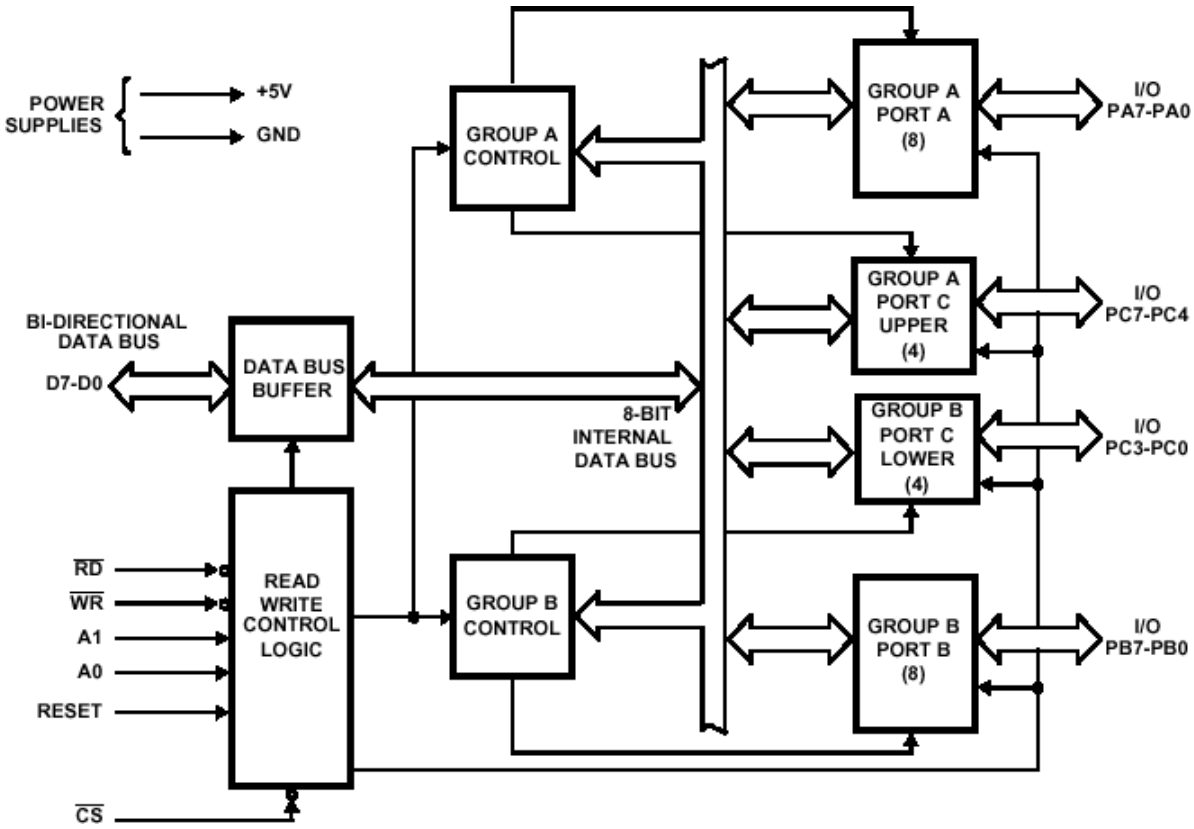
```
Start MOV DI , 1018
      MOV CL ,O4
      MOV AL, [DI]
      OUT CO,AL
      MOV DX , 1010
      DEC DX
      JNC 100F
      INC DI
      LOOP 1007
      JMP 1000
```

OPCODE TABLE:

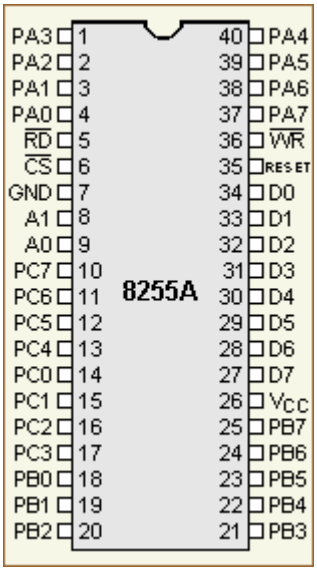
ADDRESS	OPCODE	MNEMONICS	COMMENTS
1000	C7	MOV DI, 1018	Move 1018 to DI
1001	C7		
1002	18		
1003	10		
1004	C6	MOV CI,04	Move 04 to CI
1005	C7		
1006	04	MOV AI [DI]	Move DI to AL
1007	8A		
1008	05		
1009	E6	OUT CO , AL	
100A	C0		
100B	C7	MOV DX 1010	Move 1010 to DX
100C	C2		
100D	10		
100E	10	DEC DX	Decrement DX
100F	4A	JNC DI	
1010	75		
1011	FD		
1012	47	INC DI	Increment DI
1013	E2	L00P 1007	
1014	F2		
1015	E9		
1016	E8	JUMP 1000	Jump to LOOP 1000
1017	FF		

SAMPLE DATA:

CLOCKWISE	ANTI-CLOCKWISE
1018 – 09	1018 – 0A
1019 – 05	1019 – 06
101A – 06	101A – 05
101B – 0A	101B – 09

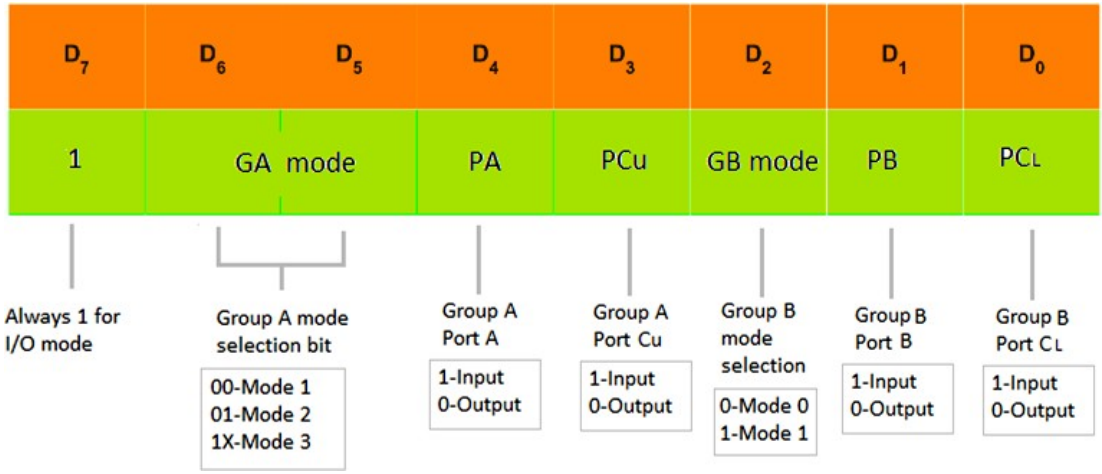


Block Diagram of 8255 IC



Pin Diagram of 8255 IC

Control Word Format:



RESULT:

Thus the stepper motor control using 8255IC was performed.

Ex. No : 10

Date :

DATA TRANSFER USING USART

AIM:

To check the transmission and receiving of a block of data using 8251.

ALGORITHM:

1. Initialize 8253 and 8251A.
2. Load CX register with the number 06 data to transfer.
3. Read the status register and check if transmitter is ready than read data to output part the repeat until transmitter in ready.
4. If count CX_1=0 repeat step3 and step4.
5. Else stop the process.

SOURCE CODE:

```
MOV AL, 36H

OUT 16H, AL

MOV AL ,08H

OUT 10H , AL

XOR AL ,AL

OUT 10H ,AL

MOV AL ,40H

OUT 12H, AL

MOV AL, 37H

OUT 12H, AL
```

```
        MOV DI ,1300H
        MOV CX,[DI]
        MOV SI,1500H
        MOV DI,1600H
TX:     IN AL, 0AH
        AND AL, 04H
        JZ  TX
        MOV AL, [SI]
        OUT 08H,AL
LI:     IN AL 0AH
        AND AL ,02H
        JZ  L1
        JN AL,08H
        MOV [DI],AL
        INC SI
        INC DI
        LOOP TX
        HLT
```

OPCODE TABLE:

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1000	B0	MOV AL,36H	Move 36H to AL
1001	36		
1002	E6	OUT 16H,AL	Display
1003	16		
1004	B0	MOV AL, 08H	Move 08H to AL
1005	08		
1006	E6	OUT 10H, AL	Display
1007	10		
1008	32	XOR AL, AL	XOR AL content
1009	C0		
100A	E6	OUT 10H, AL	Display
100B	10		
100C	B0	MOV AL,40H	Move 40H to AL
100D	40		
100E	E6	OUT 12H AL	Display
101F	12		
1010	B0	MOV AL ,4EH	Move 4EH to AL
1011	4E		
1012	E6	OUT 12H, AL	Display
1013	12		
1014	BF	MOV DI, 1300	Move 1300 to DI
1015	00		
1016	13		
1017	8B	MOV CX, [DI]	Move DI value to CX
1018	0D		
1019	BE	MOV SI, 1500H	Move 1500 to SI
101A	00		

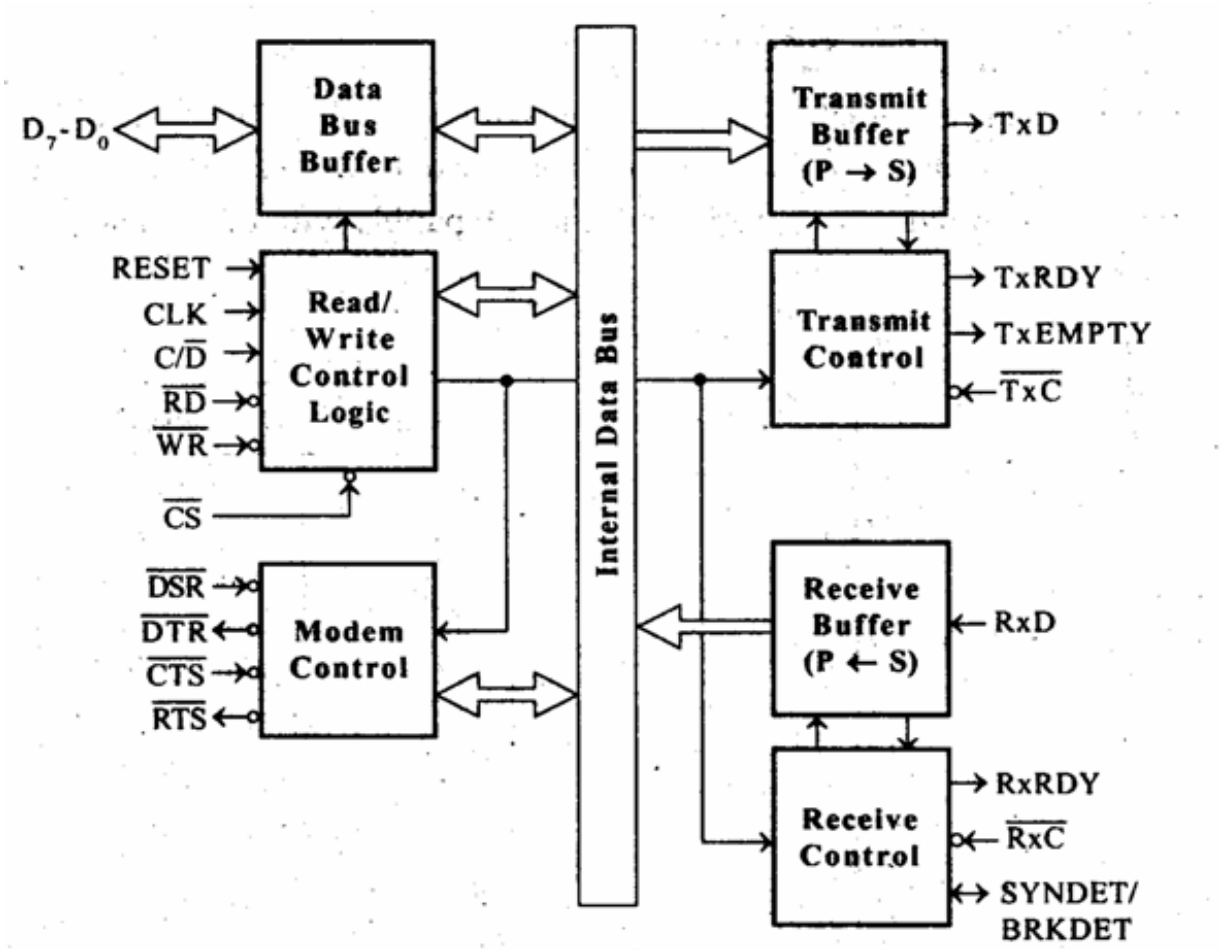
101B	15		
101C	BF	MOV DI 1600	Move 1600 to DI
101D	00		
101E	16		
101F	E4	TX: INAL ,0AH	Move 0AH to AL
1020	0A		
1021	24	AND AL,04H	AND AL with 04
1022	04		
1023	74	JZ TX	Jump to zero to tX
1024	FA		
1025	8A	MOV AL, [SI]	Move SI value to AL
1026	04		
1027	E6	OUT 08H,AL	Display
1028	08		
1029	E4	L1: IN AL,0A	Move 0A in AL
102A	0A		
102B	24	AND AL,02	AND 02 with AL
102C	02		
102D	74	JZ L1	Jump on zero to L1
102E	FA		
102F	E4	INAL ,08H	Move 08H in AL
1030	08		
1031	88	MOV [DI] ,AL	Move AL to DI
1032	05		
1033	46	INC SI	Increment SI
1034	47	INC DI	Increment DI
1035	E2	LOOP TX	Goto loop TX
1036	E8		
1037	F4	HLT	Halt the program

SAMPLE DATA:

1300 = 0A [Number of data to be transferred**]**

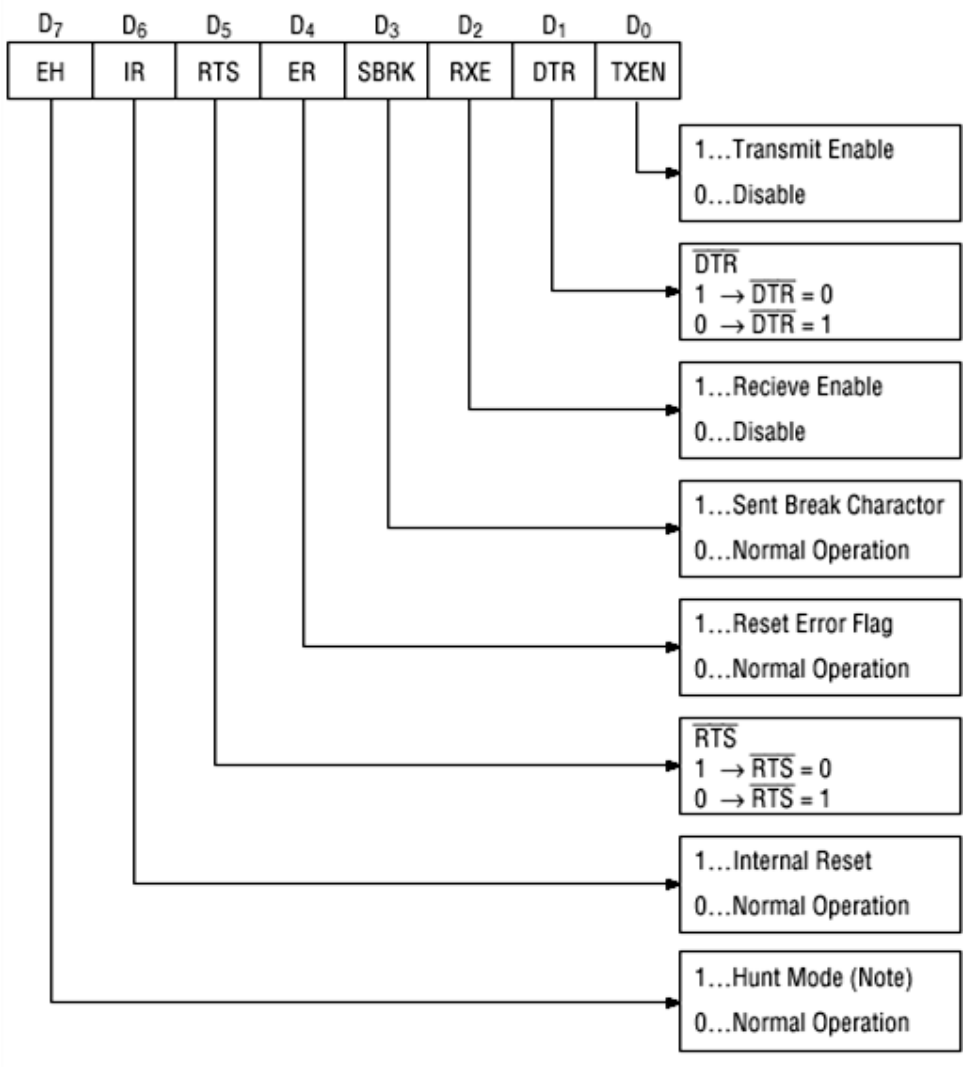
Input:	**Output:**
1500H = 11H	1600H = 11H
1501H = 22H	1601H = 22H
1502H = 33H	1602H = 33H
1503H = 44H	1603H = 44H
1504H = 55H	1604H = 55H
1505H = 66H	1605H = 66H
1506H = 77H	1606H = 77H
1507H = 88H	1607H = 88H
1508H = 99H	1608H = 99H
1509H = AAH	1609H = AAH

Pin Diagram for 8251 IC



Block Diagram for 8251 IC

USART Control Format:



RESULT:

Thus the block of data transferred successfully using **USART**.

Ex. No : 11

Date :

MESSAGE DISPLAY USING 8279IC

AIM:

To display a message using keyboard display (8279 IC).

ALGORITHM:

1. To display a message 8279 must be reset and initialized the initial that must done other reset for 8279 are

2. Write the keyboard display mode replay

3. Clear the display RAM.

4. Initialize low address to which such register writing one to be done.

SOURCE CODE:

```
        MOV AL , 00H
        OUT 02H , AL
        MOV AL , 00H
        OUT 02H , AL
        MOV AL , 90H
        OUT 02H , AL
        MOV CL , 05H
        MOV DI , 1100H
LOOP: MOV AL, [DI]
        OUT 00H , AL
```

INC DI

DEC CL

JNZ LOOP

HLT

Control word format:

1. **Keyword/display mode setup:**

00	08	8bit	character display left centre
11	16	8bit	character display left centre
10	08	8bit	character display left centre
11	10	8bit	character display left centre

2. **Write display RAM:**

1	0	0	A_1	A	A	A	A	$A A_1$ – auto increment flag

If A_1=0 the row add select will be increment offer each read/write to display

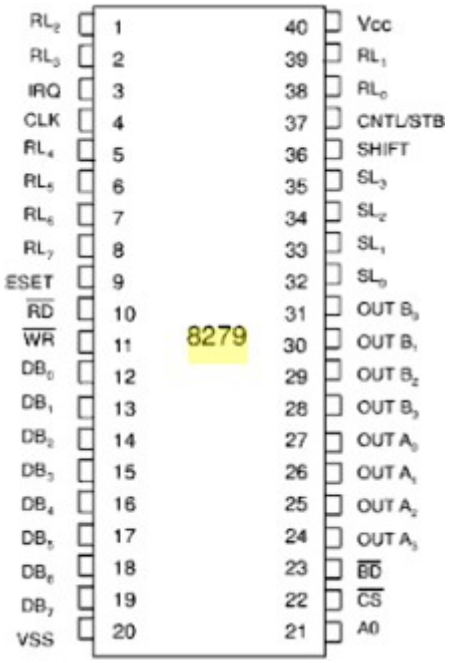
3. **Char display RAM:**

1	1	0	CD_2	CD_1	CD_0	CF	CA	CD_1	CD_0

0	X	A_0-A_0	B_0-B_3 (0000-0000)
1	0	A_0-A_3	B_0-B_3 (0000-0000)
1	1	A_0-A_3	B_0-B_3 (1111-1111)

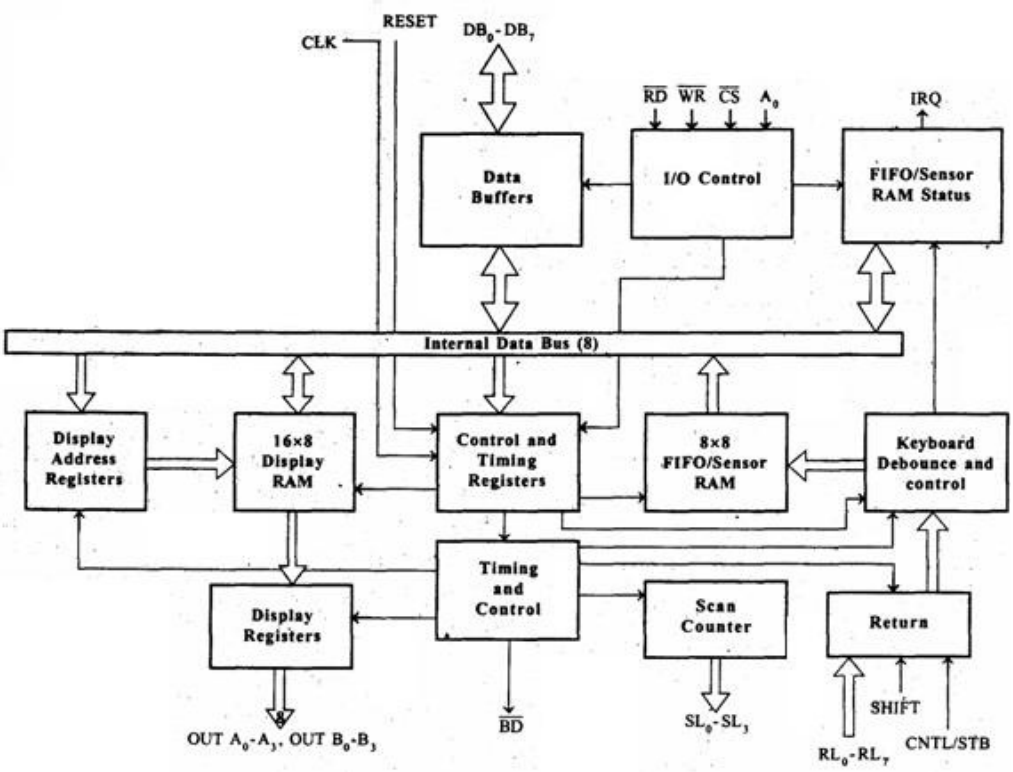
OPCODE TABLE:

ADDRESS	OPCODE	MNEMONICS	COMMENTS
1000	B0	MOV AL , 00H	Move 00H to AL
1001	00		
1002	E6	OUT 02H , AL	Display
1003	02		
1004	B0	MOV AL ,00H	Move 00H to AL
1005	00		
1006	E6	OUT 02H ,AL	Display
1007	02		
1008	B0	MOV AL ,90H	Move 90H to AL
1009	90		
100A	E6	OUT 02H, AL	Display
100B	02		
100C	B1	MOV CL , 05H	Move 05 to CL
100D	05		
100E	BF	MOV DI, 1100H	Move 1100H to DI
100F	00		
1010	11		
1011	8A	MOV AL, [DI]	Move DI to AL
1012	05		
1013	E6	OUT OOH, AL	Display
1014	00		
1015	47	INC DI	Increment DI
1016	FE	DEC CL	Decrement CL
1017	C9		
1018	75	JNZ LOOP	Jump on non zero
1019	F7		
101A	F4	HLT	Halt



Pin Diagram for 8279 IC

Block Diagram for 8279



SAMPLE INPUT

Letter	Dp	G	E	F	D	C	B	A	equation
P	1	0	0	0	1	1	0	0	8C
E	1	0	0	0	0	1	1	0	86
P	1	0	0	0	1	1	0	0	8C
S	1	0	0	1	0	0	1	0	92
I	1	1	1	1	0	0	0	1	79

INPUT:

1100H = 8CH

1101H = 86H

1102H = 8CH

1103H = 92H

1104H = 79H

Output:

PEPSI

RESULT:

Thus a describe message display on microprocessor kit is successfully executed.

Ex. No : 12

Date :

SIMULATION OF TRAFFIC LIGHT CONTROL SIGNAL

AIM:

To simulation the traffic light controller signal red, green, yellow using 8255 IC's.

INTRODUCTION:

The VIRAF board is a simple construction of a traffic control system where in the signaling lights are simulated by the blanking or ON-OFF control of emitting diode. A mode of a four lead leave junction the board has green , yellow and RED LED's which are the green , yellow and red signals of a actual system third two such IED's are used to the board.

Circuit description:

The control of LED's is a follow the VIRAF board communication which the microprocessor trainer by means of a 26 core cable which is connected to the output pins of 8255 programmable principal interface chip. The outputs are the input to the buffer 741507 where output drive the diode whether it is ON or OFF.

Software:

The simulation software is a below and is based on successive outputting of appropriate byte to the 8255 followed by a pro-calculated delay and so on and the software of self-ex-plantary.

ALGORITHM:

1. Start
2. Load the signals and counter with the number of pair of data
3. Load the pot of data A, B, C consecutively with the signals calling the delay before the shift to the next signal.
4. Decrement the counter
5. Stop the execution.

SOURCE CODE:

```
        ORG 1000H
        MOV AL, 80H
        OUT 26H, AL
M1:     MOV SI, 1500H
        MOV CX, 0003H
BE:     MOV AL, [SI]
        OUT 20H, AL
        INC SI
        MOV AL, [SI]
        OUT 22H, AL
        INC SI
        MOV AL, [SI]
        OUT 24H, AL
        MOV BX, 000FH
L2:   MOV DX , 0FFFH
L1:   DEC DX
        JNC LI
```

DEC BX
JNC L2
INC SI
LOOP BE
JMP SI
HLT

OPCODE TABLE:

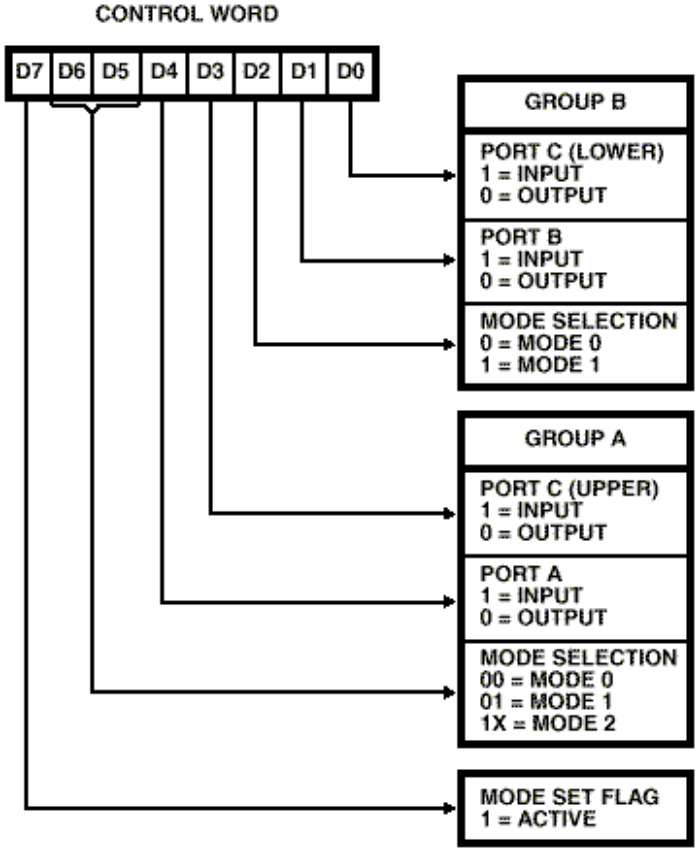
ADDRESS	OPCODE	MNEMONICS	COMMENTS
1000	B0 80	MOV AL ,80H	Move 80 H to AL.
1002	E6 26	OUT 26H AL	Display
1004	BE 1500	Start MOV SI	Move 1500H to SI
1007	B9 0003	MOV CX, 0003H	Move 0003 to CX
100A	8A 04	BE: MOV AL, [SI]	Move SI to AL
100C	E6 20	OUT 20,AL	Display
100E	46	INC SI	Increment SI
100F	84 04	MOV AL, [SI]	Move SI to AL
1011	E6 22	OUT 22H, AL	Display
1013	46	INC SI	Increment SI
1014	BB OOOF	MOV BX,000F	Move 000F to BX
1016	BA OFFFF	L2:MOV DX,0FFFH	Move 0FFF to DX
1018	4A	L1: DEC DX	Decrement DX
101B	75 FD	JNZ L1	Jump on non zero to L1
101D	4B	DEC BX	Decrement BX
101E	75 F7	JNZ L2	Jump on non zero to L2
1020	46	INC S1	Increment S1
1021	E2 E7	LOOP BE	Loop to memory address
1022	EB DF	JMP START	Jump to SI
1023	F4	HLT	Halt.

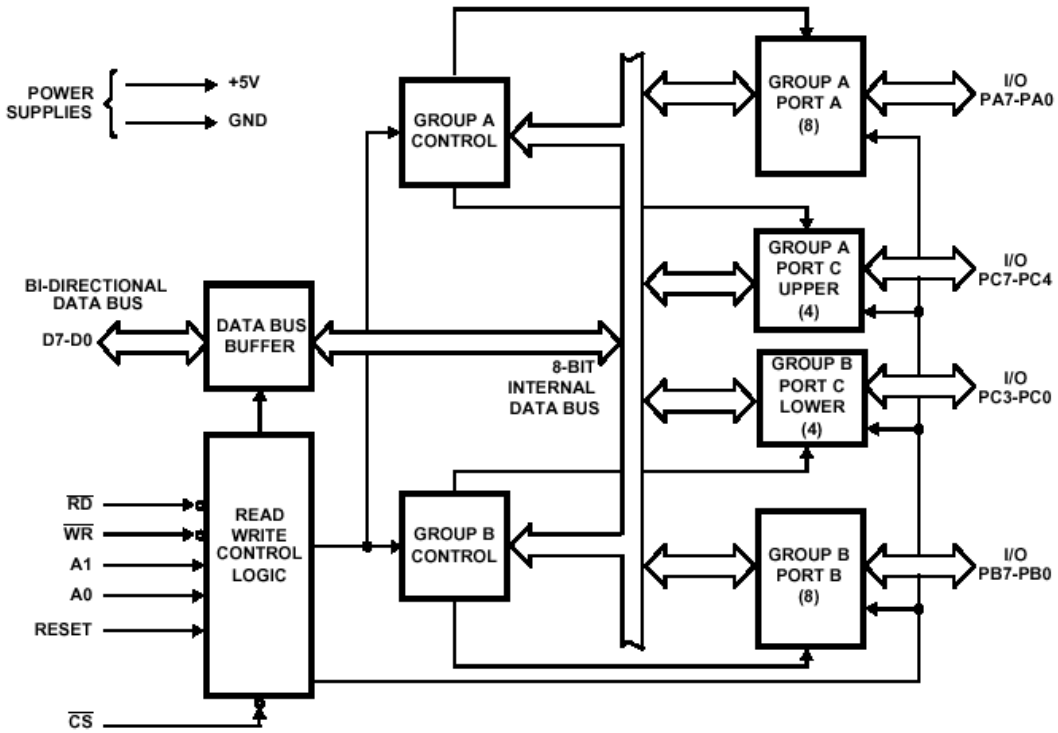
DATA: S_0 A_2 C_4 2A 11 22 33 CC 11 0C

SIGNAL:

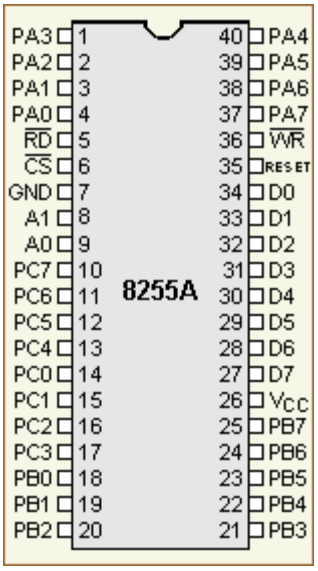
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	1	0	0	0	1	0	A_2
0	0	0	1	0	0	0	1	11
1	1	0	0	1	1	0	0	CC
B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
1	1	0	0	0	1	0	0	C4
0	0	1	0	0	0	1	0	22
0	0	0	1	0	0	0	1	11
C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	
0	0	1	0	1	0	1	0	2A
0	0	1	1	0	0	1	1	33
0	0	1	0	1	1	0	0	2C

Control Word Format:





Block Diagram of 8255 IC



Pin Diagram of 8255 IC

RESULT:

Thus the program for traffic light signal control using 8255 IC has been performed successfully.