B.M.S INSTITUTE OF TECHNOLOGY & MANAGEMENT

Department of Computer Science & Engineering

LAB MANUAL

MICROPROCESSOR - HARDWARE PART (8086)

Sub Code: 15CSL48

4^{th} Semester CSE

Prepared by:

Shankar. R
Asst. Professor, CSE
BMSIT&M

Reviewed By:

Dr. G Thippeswamy
Professor & Head, CSE
BMSIT&M

Programs

8. a. Design and develop an assembly program to demonstrate **BCD Up-Down Counter (00-99)** on the Logic Controller Interface.

b. Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display **X*Y**.

9. Design and develop an assembly program to display messages **"FIRE" and "HELP"** alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).

10. Design and develop an assembly program to drive a **Stepper Motor** interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).

11. Design and develop an assembly language program to

a. Generate the **Sine Wave** using DAC interface (The output of the DAC is to be displayed on the CRO).

b. Generate a **Half Rectified Sine** waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).

The 8086 Microprocessor Pin Diagram

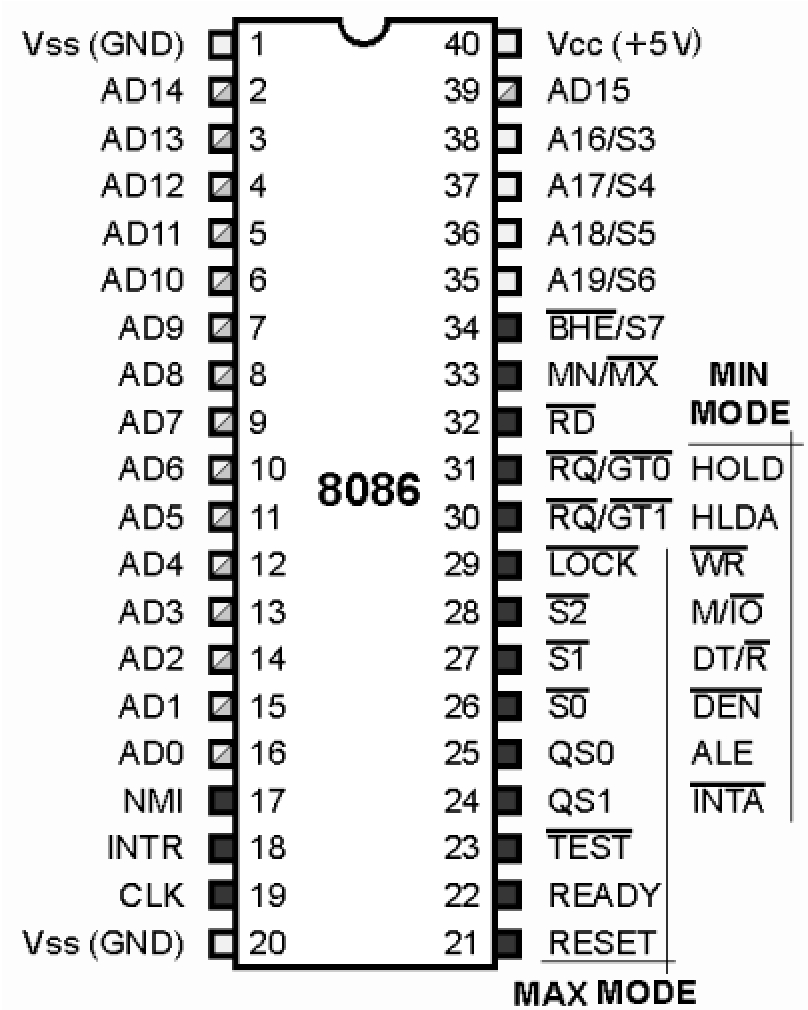
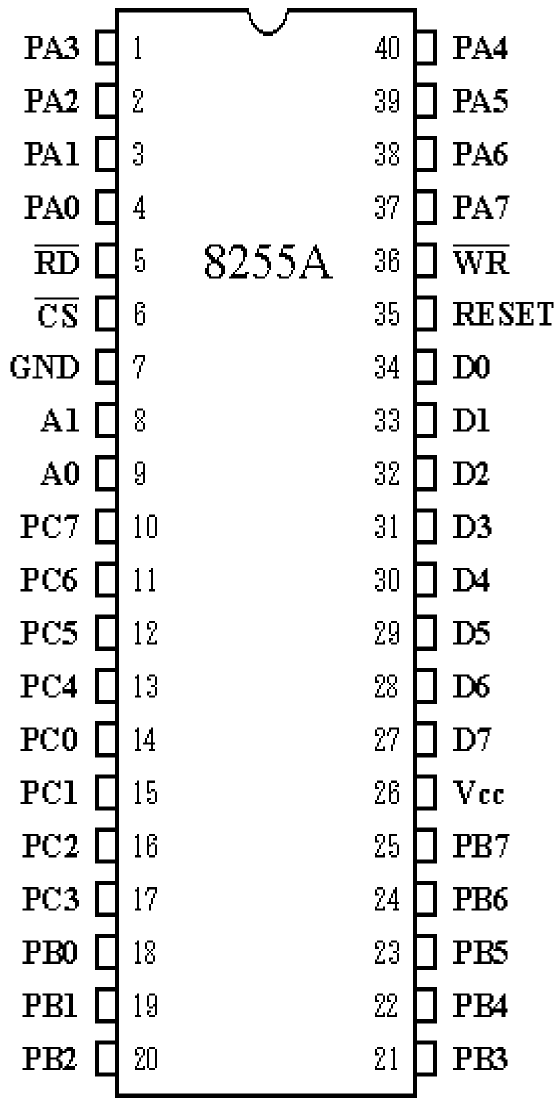


Fig:- The 8086 Microprocessor Pin Diagram

8255 Programmable Peripheral Interface (PPI)



Data Bus Buffer

This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A "low" on this input pin enables the communication between the 8255 and the CPU.

(RD) Read. A "low" on this input pin enables 8255 to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255.

(WR) Write. A "low" on this input pin enables the CPU to write data or control words into the 8255.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

(RESET) Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode.

A1	A0	SELECTION
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	CONTROL

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255. Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

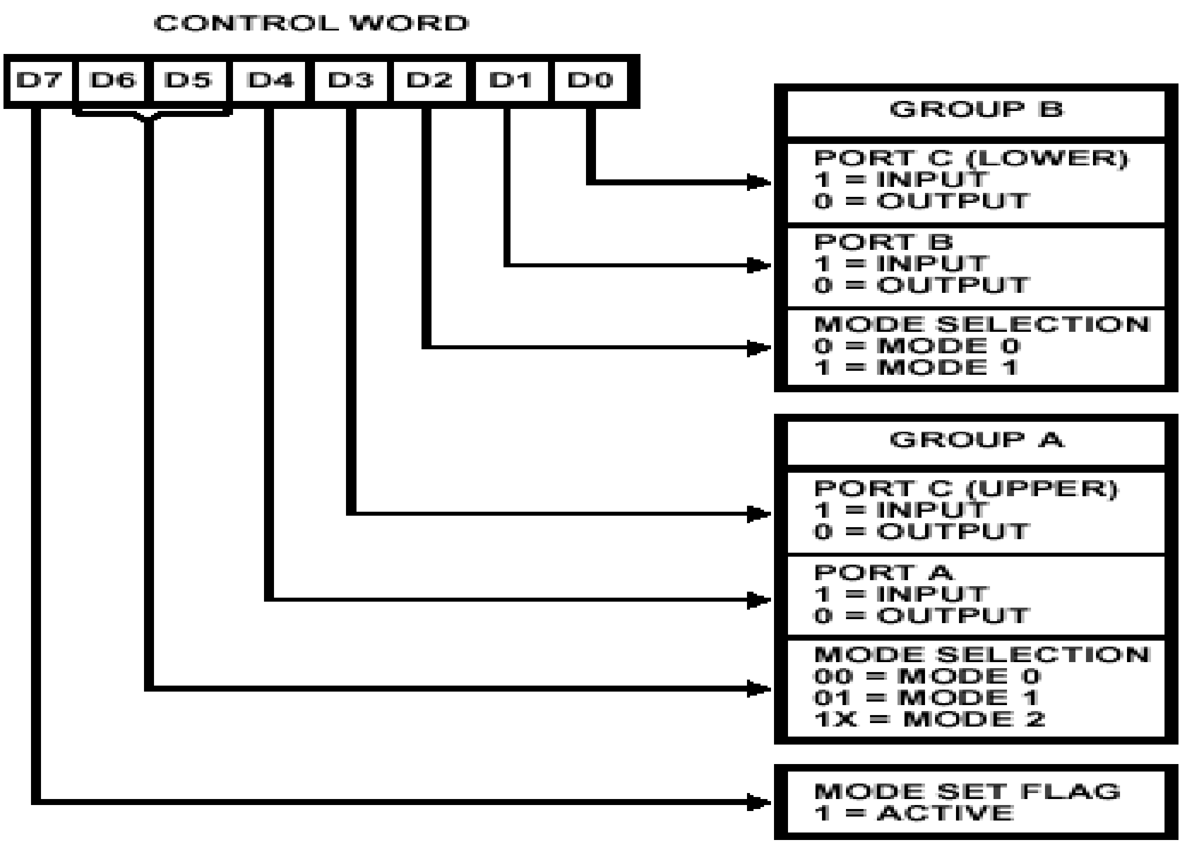
Ports A, B, and C

The 8255 contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both "pull-up" and "pull-down" bus-hold devices are present on Port A.

Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.



<u>Examples and Basics:</u> *Read these thoroughly!*

1. If we want to take input and give output, then we can make any port (out of 3 ports) as input and any other port as output port. So, as an example, let us make **Port A as output and Port B as input.**

Now, fill the above table accordingly for this case. It will be 82h. Well, It's done like this.

;*PORT A as output & PORT B as input*

82 means → 8 | 2

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0
always **1** for I/O mode	always **0** for mode 0	always **0** for mode 0	**0 →PORT A output**	**0** →PORT C (upper) output	always **0** for mode 0	**1 →PORT B input**	**0** →PORT C (lower) output

2. If we want to get 2 outputs, no input at all, then we can make any 2 ports as output ports. So let us make **Port B as output and Port C as output.**

Now, fill the table accordingly for this case. It will be 80h. It's done like this.

;*PORT B as output & PORT C as output as well.*

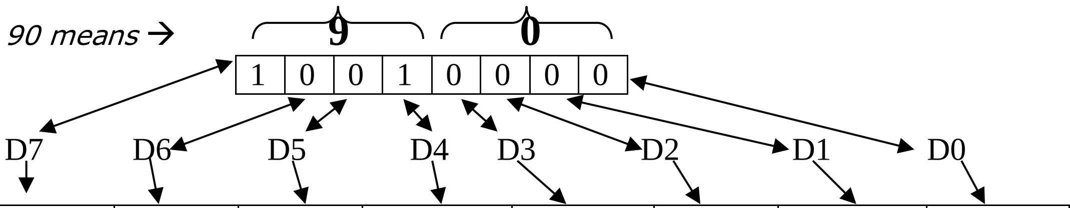
80 means → 8 | 0

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0
always **1** for I/O mode	always **0** for mode 0	always **0** for mode 0	**0 →PORT A output**	**0** →PORT C (upper) output	always **0** for mode 0	**0 →PORT B output**	**0** →PORT C (lower) output

3. Moving on, let us make **Port A as input and Port B as output.**

Now, fill the table accordingly for this case. It will be 90h. It's done like this.

; Port A as input and Port B as output.



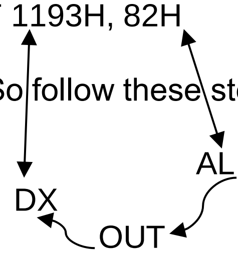
always **1** for I/O mode	always **0** for mode 0	always **0** for mode 0	**1 →PORT A input**	**0** →PORT C (upper) output	always **0** for mode 0	**0 →PORT B output**	**0** →PORT C (lower) output

All these are called as Control Word data. In order to use this, you must **initialize your 8255**. So the following 3 lines help you in initializing 8255 PPI.

Essentially, all you need to do is give this control word (82h or 80h or 90h or whatever) to the control word's address. Let's say that your control word is 82h and the address of it is 1193h. So you should give 82h to 1193h. Technically, you need to do this.

OUT 1193H, 82H

But, the problem is you can't give directly. So follow these steps.

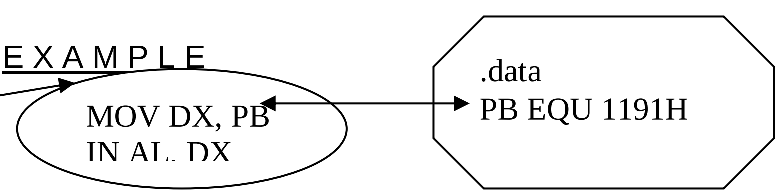


Step1: **MOV AL, 82H**
Step2: **MOV DX, 1193H**
Step3: **OUT DX, AL**

Once you decide which port is what and initialize your 8255, next task is to take input from a port or/and give an output via some port which again depends on your requirement.

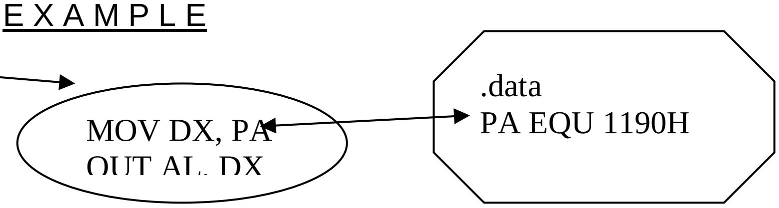
How to give input from a port? <u>E X A M P L E</u>

Answer: **MOV DX, *port address***
IN AL, DX

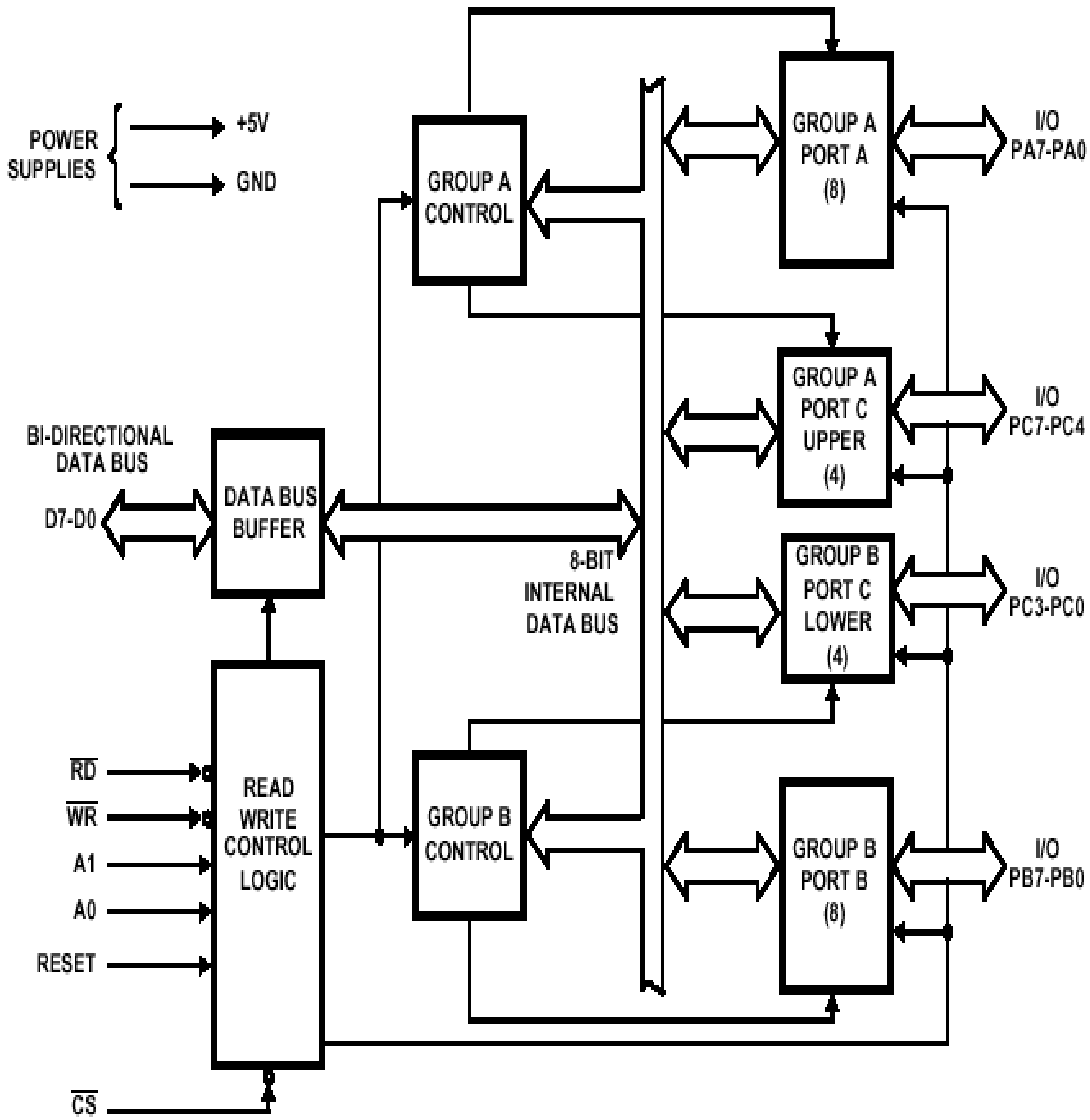


How to give output to a port? <u>E X A M P L E</u>

Answer: **MOV DX, *port address***
OUT DX, AL



8255 PPI Internal Architecture



Microprocessor Hardware Interface Devices

LOGIC CONTROLLER INTERFACE *(for 8a & 8b pgms)*

THEORY

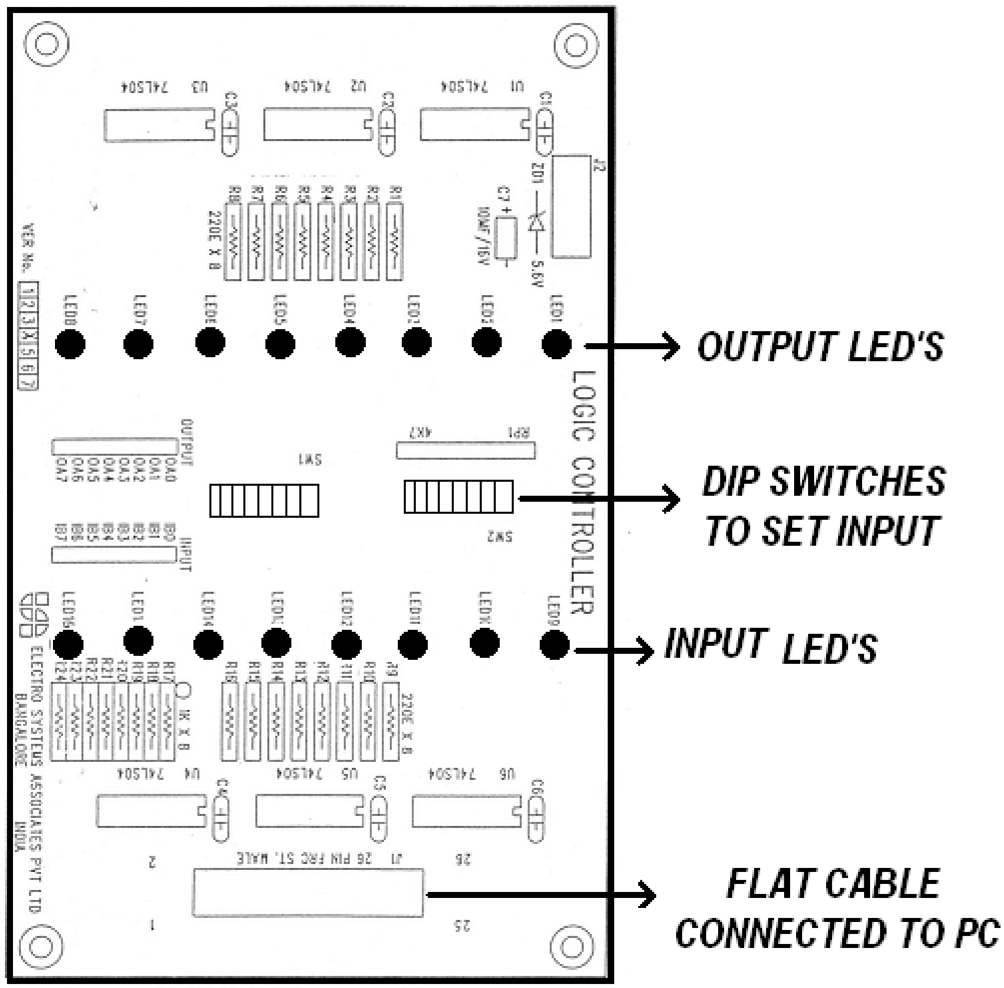
The interface consists of 8 TTL buffered outputs and 8 TTL buffered inputs. The logic state of each input/output is indicated by a corresponding LED (ON/OFF). The inputs can be read through PORT -B and the outputs can be controlled through PORT A. The inputs and outputs brought to 26-pin controller. Inputs LED's are controlled through Dipswitches.

This interface allows the user to perform experiments to understand some of the basic programming techniques involved in a logic controller. The software exercises could include combinational controllers, sequential controllers, programmable counters, etc.

OPERATION.

Logic controller module is used as an input and output device. The 26-pin line plat ribbon cable (FRC) from the DIO card is connected to this module. Port A pins of 8255 are connected to 8 LEDs. When the Port A is high (1), than LED glows and when it is low (0), then LED is switched OFF. The port B-of 8255 is connected to logic controller toggle (DIP) switch. The toggle switch in turn is connected to an LED to indicate the state of the switch. When the switch is opened the Led is turned OFF and when switch is closed, the LED is ON.

LOGIC CONTROLLER HARDWARE MODULE DIAGRAM



8. a. Design and develop an assembly program to demonstrate BCD Up-Down Counter (00-99) on the Logic Controller Interface.

```
.model small

initds macro
    mov ax,@data        ; initializing the data segment
    mov ds,ax           ; it is ds, not dx
endm

init8255 macro
    mov al,cw        ; initialization of 8255 using control word
    mov dx,cr          by passing 82h to control reg.
    out dx,al          (to make port A as output)
endm

outpa macro
    mov dx,pa           ; initialization of port A as output
    out dx,al
endm

printf macro msg
    lea dx,msg          ; load the effective address to dx
    mov ah,9            ; function number is 9
    int 21h             ; using dos interrupt 21h
endm

getchar macro
    mov ah,1            ; this macro takes 1 key input,
    int 21h             ; its ascii value in hex stores in al
endm

exit macro
    mov ah,4ch          ; to terminate
    int 21h
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.data
    pa equ 1190h      ; setting the port address for port A
    cr equ 1193h      ; setting the port address for control reg
    cw db 82h         ; control word is 82 (PORT A is O/P)

    select db 10,13,"select 1: up counter 2: down counter $"

    exitmsg db 10,13,"press any key to exit $"

.code
    initds          ; initialize data segment
    init8255        ; initialize 8255
```

```
printf select  ; print the choice
getchar        ; input the choice to AL          ; or cmp al,31h

cmp al,'1'     ; if your input is 1, go to upcounter
je upcounter
                                                 ; or cmp al,32h
cmp al,'2'     ; if your input is 2, go to downcounter
je downcounter

exit                 ; well, upon any other input, just exit.

upcounter:
    mov al,0          ; initial value of up counter is 0
    up:
        outpa        ; display the contents of al on the interface
        call delay    ; have some delay (let the user see the o/p)
        call keyboardhit    ; if you press any key, then exit.
        add al,1      ; increment the count
        daa           ; daa-decimal adjust after addition
        cmp al,99h   ;compares with 99 in order to count till 99
        jne up       ;upon adding 1, if not equal to 99, go to up
        exit          ; if it crosses 99, exit.

downcounter:
    mov al,99h        ; initial value of down counter is 99
    down:
        outpa         ; down counter starts
        call delay    ; have some delay (let the user see the o/p)
        call keyboardhit    ; if you press any key, then exit.
        sub al,1      ; decrement the count
        das           ; daa-decimal adjust after subtraction
        cmp al,0      ;compares with 0 in order to count till 0
        jne down     ;upon subtracting 1,if not equal to 0,go to down
        exit          ; if it crosses 0, exit.
```

```
delay proc
    mov bx,0fffh      ; do a waste job waste number of times!!!!
    outerfor:
        mov cx,0ffffh

    innerfor:
        loop innerfor
      dec bx
      jnz outerfor
      ret

delay endp
```

```
for (bx = bignumber; bx >= 0; bx --)
{
    for(cx = bignumber; cx >= 0; cx --)
    {
        Do nothing;
    }
}

basically, keep decrementing a huge
number till zero huge number of times.

By the time, microprocessor does this
huge decrements, you can actually see
your front-end output.
```

```
keyboardhit proc
    push ax          ;save your precious ax value
    mov ah,1         ;checks if any key is pressed in between the count
    int 16h          ;if you press any key, it becomes non-zero. so go
    jnz done             to done and exit.

    pop ax           ;if you don't press any key, it becomes zero. so
                        take out your precious value and return.
    ret

done:
     exit            ;so you have pressed a key, go to exit.

keyboardhit endp

end
```

****

OUTPUT:

select 1: up counter 2: down counter

Corresponding choice output is observed on the module

NOTE:

DAA means Decimal Adjust after Addition. Let's say AL=0. Keep adding 1 to it. It becomes 1,2,3,,,,9, after 9 it becomes A. But we don't want A, we want 10. So do DAA once it crosses 9. So it will now be 10. Now keep adding 1 to it. It becomes 11,12,13,14,,,,,19, after 19 it becomes 1A. So do DAA now because we don't want 1A, we want 20. So in a nutshell, once AL crosses 9, adjust the decimal after addition i.e. do DAA.

Similarly, **DAS** – Decimal adjust after Subtraction.

*8b. Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display X*Y.*

```
.model small

initds macro
    mov ax,@data        ; initializing the data segment
    mov ds,ax           ; it is ds, not dx
endm

init8255 macro
    mov al,cw        ; initialization of 8255 using control word
    mov dx,cr          by passing 82h to control reg.
    out dx,al          (to make port A as output & port B as input)
endm

inpb macro
    mov dx,pb          ; initialization of port B as input
    in al,dx
endm

outpa macro
    mov dx,pa        ; initialization of port A as output
    out dx,al
endm

printf macro msg
    lea dx,msg          ; load the effective address to dx
    mov ah,9            ; function number is 9
    int 21h             ; using dos interrupt 21h
endm

getchar macro
    mov ah,1            ; this macro takes 1 key input,
    int 21h             ; its ascii value in hex stores in al
endm

exit macro
    mov ah,4ch          ; to terminate
    int 21h
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.data
    askx db 10,13,"set value for x,then press any key $"
    asky db 10,13,"set value for y,then press any key $"

    pa equ 1190h      ; setting the port address for port a
    pb equ 1191h     ; setting the port address for port b
    cr equ 1193h      ; setting the port address for control reg
    cw db 82h     ; control word is 82 (PORT A is O/P, PORT B is I/P)
```

```
.code
    initds
    init8255

    printf askx             ; ask x
    getchar                 ; press any key
    inpb                    ; reads 1st value i.e. x, which is set
                              through logic controller module, value
                                will be automatically stored in al
    mov bl,al               ; contents of al is copied to bl

    printf asky             ; ask y
    getchar                 ; press any key
    inpb                    ; reads 1st value i.e. x, which is set
                              through logic controller module, value
                                will be automatically stored in al

    mul bl   ; the contents of al is multiplied with contents of bl
                   Result is stored in AX
    outpa
    call delay

    mov al,ah

    outpa
    call delay

    exit

 delay proc
    mov bx,0ffffh     ; do a waste job waste number of times!!!!
    outerfor:
        mov cx,0ffffh

    innerfor:
        loop innerfor
      dec bx
      jnz outerfor
      ret

delay endp


end
```

the result of multiplication is stored in AX reg i.e. AL will be having first 8 bits result, AH – next 8 bits. AL is displayed first on the output module, after some delay, rest 8 bits which are in AH is copied to AL and then displayed on the module.

```
for (bx = bignumber; bx >= 0; bx --)
{
    for(cx = bignumber; cx >= 0; cx --)
    {
        Do nothing;
    }
}
```

basically, keep decrementing a huge number till zero huge number of times.

By the time, microprocessor does this huge decrements, you can actually see your front-end output.

****

SEVEN SEGMENT DISPLAY *(for 9th pgm)*

INTRODUCTION:

This interface provides a four digit 7-segmcnt display driven by the output of four cascaded shift registers. Data to be displayed is transmitted serially (bit by bit) to the interface. Each bit is clocked into the shift register by providing a common clock through a port line (PC4 in ALS module). Thus one port line PB0 provides the data.

This interface allows the user to study seven segment display control technique, code conversion methods, etc. the software exercise could include procedures for table lookup, a variety of bit manipulation operations, etc.

CIRCUIT DECSRIPTION:

This interface allows display of up to 4 digits. The technique adopted uses four 8 bit serial in parallel out (SIPO) shift registers. The 8 outputs of the shift register arc connected to the seven-segment display through register. Each character is represented by an 8 bit 5 code and shifting is done by applying the MSB of the code corresponding to the last digit on the right to the data input of the 1st shift register and applying a clock pulse. The next most significant bit follows this till all the bits for that digit are exhausted. The MSB of the next digit from the right is then fed to the data input and this process is continued till all the digits are displayed. A total of 32 clock pulses are required to display the four digits.

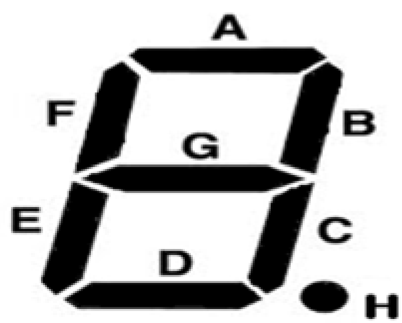
The code corresponding to the four digits can be stored in consecutive locations in RAM to be accessed by the output routine. A look up-table is used to convert these characters to their corresponding output code. The codes for the characters 0 to 9 and A to F are given in the table. With this scheme it is possible to output any special characters as the user can very easily write the corresponding output code.

This interface uses MAN72 Common Anode seven-segment display. I lcnce low (0) inputs must be given to each segment to glow (ON) and high to blank (OFF). The circuit diagram of the seven-segment display interface is provided at the beginning.

Design of Seven-Segment Code (SSC)

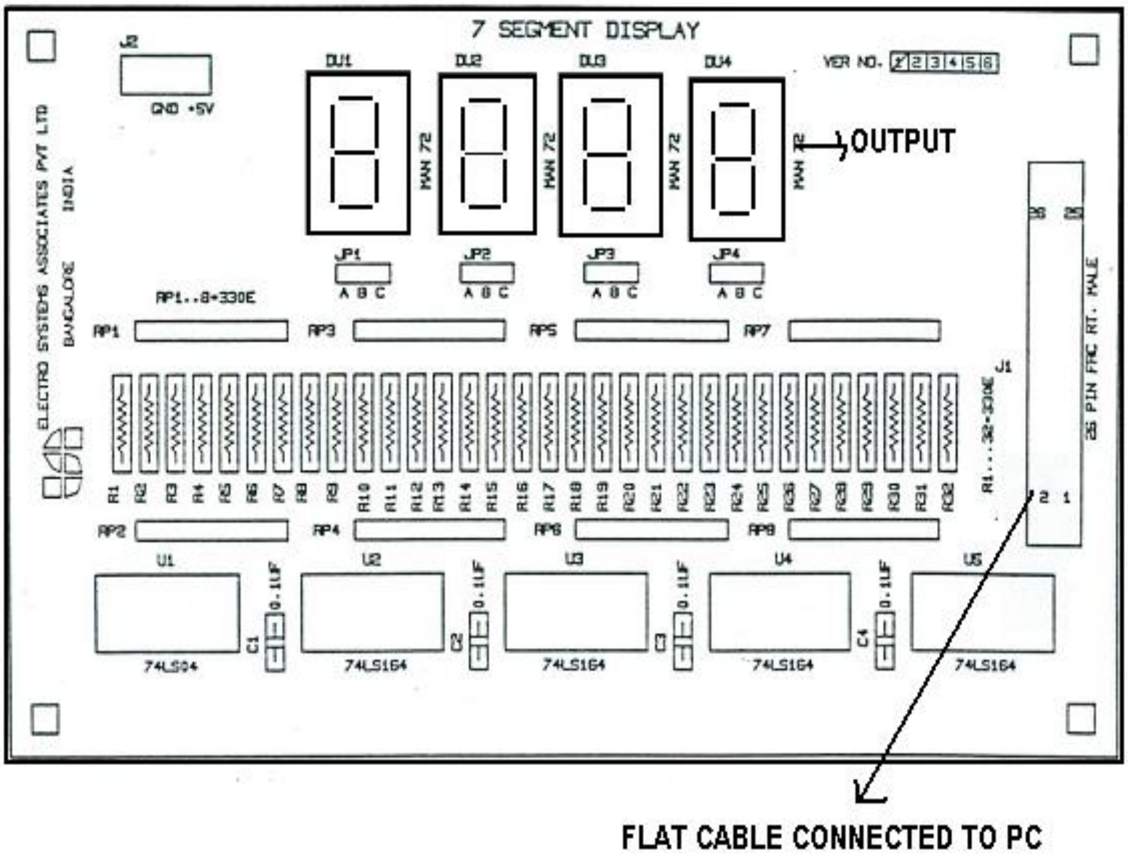
O=ON, 1=OFF.
Each display has seven segment and a dot (a, b, c, d, c, f, g and h). it is as shown in fig . For displaying any character the corresponding segment must be given low (0) inputs.
SSC to display Hex-digits are given in the table.

Fig: Seven Segment Display



Angka	PC.7	PC.6	PC.5	PC.4	PC.3	PC.2	PC.1	PC.0	Hex
	h	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	**C0**
1	1	1	1	1	1	0	0	1	**F9**
2	1	0	1	0	0	1	0	0	**A4**
3	1	0	1	1	0	0	0	0	**B0**
4	1	0	0	1	1	0	0	1	**99**
5	1	0	0	1	0	0	1	0	**92**
6	1	0	0	0	0	0	1	0	**82**
7	1	1	1	1	1	0	0	0	**F8**
8	1	0	0	0	0	0	0	0	**80**
9	1	0	0	1	0	0	0	0	**90**

7-SEGMENT DISPLAY HARDWARE MODULE DIAGRAM



9. Design and develop an assembly program to display messages "FIRE" and "HELP" alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).

```
.model small

initds macro
    mov ax,@data          ; initializing the data segment
    mov ds,ax             ; it is ds, not dx
endm

init8255 macro
    mov al,cw          ; initialization of 8255 using control word
    mov dx,cr            by passing 80h to control reg.
    out dx,al            (to make port B as output & port C as output)
endm

outpb macro
    mov dx,pb             ; initialization of port B as output
    out dx,al
endm

outpc macro
    mov dx,pc             ; initialization of port C as output
    out dx,al
endm

printf macro msg
    lea dx,msg            ; load the effective address to dx
    mov ah,9              ; function number is 9
    int 21h               ; using dos interrupt 21h
endm

exit macro
    mov ah,4ch            ; to terminate
    int 21h
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.data
    pb equ 1191h   ;setting the port address for port B
    pc equ 1192h   ;setting the port address for port C
    cr equ 1193h   ;setting the port address for control reg.

    cw db 80h      ;80h is the value in control word 10000000, which
                   makes port B as output & port C as out put
    anykeytoexit db 10,13,"press any key to exit $"
```

```
            ;F   I   r   E
    fire db 8eh,0cfh,0afh,86h

            ;H   E   L   P
    help db 89h,86h,0c7h,8ch
```

As capital R cannot be displayed we are considering small r (if you want R, go ahead in terms of A - 88H)

variables FIRE & HELP contains hexa decimal values for FIRE & HELP to display on 7-segment display module. You can even try displaying your name with hexa decimal values.

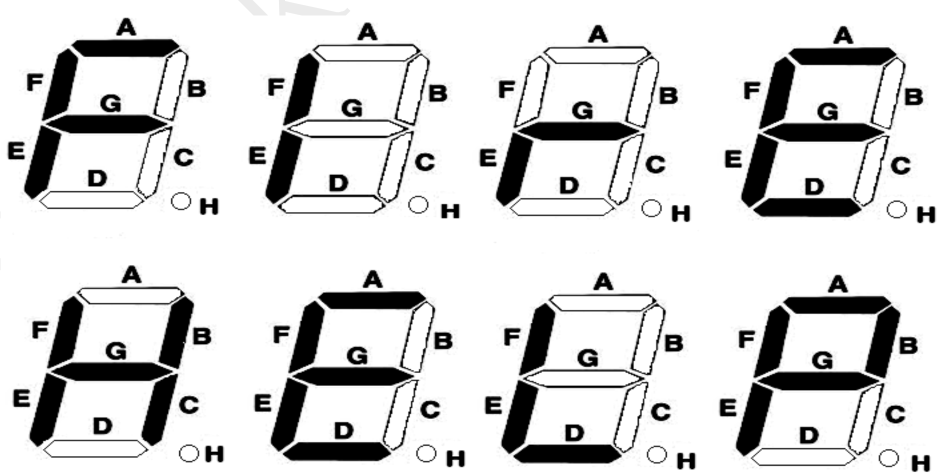
```
.code
    initds
    init8255

    printf anykeytoexit   ; displays press any key to exit

    start:
        lea si,fire       ; loads the address of fire to si
        call disp_msg     ; displays the contents of table form fire
        call delay

        lea si,help       ; loads the address of help to si
        call disp_msg     ; displays the contents of table form help
        call delay

        mov ah,1       ; checks if any key from key board is  pressed
        int 16h
        jz start

        exit              ; terminate program


disp_msg proc      ; displaying char starts from this proc
    mov cx,4 ; count is taken 4 b'coz of 4 char in 1st string i.e. fire
    nextchar:
        mov bl,8          ; bl indicates 8 bits in single char
        mov al,[si]       ; char is moved to al from si which is in
                         the form of 8-bit data
        nextbit:
            rol al,1      ; rotate left will sends data out bit by bit
            outpb         ; sends bit by bit to output module

            push ax       ; keeps copy of ax in stack b'coz next
                               instruction changes it.
            mov al,00h    ; clock pulse 0 given to drive the bits on
             outpc                       led through port c

             mov al,11h   ; clock pulse 1 given to drive the bits on
             outpc                       led through port c
```

```
            pop ax        ; copy is retrieved from stack

            dec bl        ; decrements the bit count
            jnz néxtbit   ; repeats until bit count becomes 0

        inc si            ; si is pointed to next char
        loop nextchar     ; automatically decrements char count (cx)

        ret               ; returns the control to called instruction
disp_msg endp


delay proc

    mov bx,0ffffh     ; do a waste job waste number of times!!!!
    outerfor:
        mov cx,0ffffh

    innerfor:
        loop innerfor
      dec bx
      jnz outerfor
      ret

delay endp


end
```

```
for (bx = bignumber; bx >= 0; bx --)
{
    for(cx = bignumber; cx >= 0; cx --)
    {
        Do nothing;
    }
}

basically, keep decrementing a huge
number till zero huge number of times.

By the time, microprocessor does this
huge decrements, you can actually see
your front-end output.
```

****

OUTPUT ON THE MODULE:



STEPPER MOTOR INTERFACE (for 10th program)

Data acquition and control represents the most popular applications of Microprocessor . Stepper motor control is a very popular application of control area, as stepper motors are capable of accepting pulses directly from the microprocessor and move accordingly.

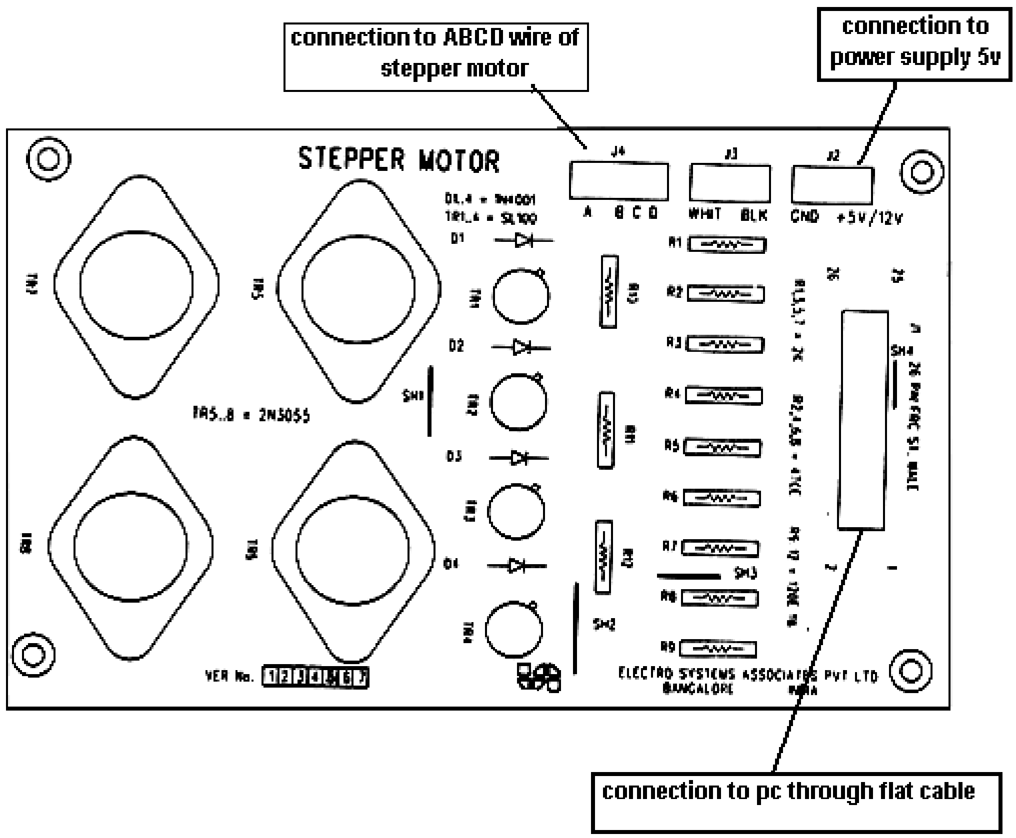
There are several areas of stepper motor applications like instrumentation, computer peripherals and machine tool drives. Tiny stepper motors are used in quartz analog electronics watches for driving the second, minute and hour hands. These motors operate directly within the button cells used in these electronic watches. Bigger stepper motors are used for driving the hands of slave clocks on railway platforms, bus stations, offices, factories , etc computer peripherals form an important areas of stepper motor applications. Card readers/punches, papers tape readers/punches, teleprinters and teletypes represents the first application's areas of stepper motors. Digital X-Y plotters and dot matrix printer's uses stepper motors for driving the arm and pen, and the paper respectively. Stepper motors find application in line printers to drive the paper advance mechanism. Floppy disks and hard/Winchesters disks have their magnetic reading/writing heads positioned by stepper motors.

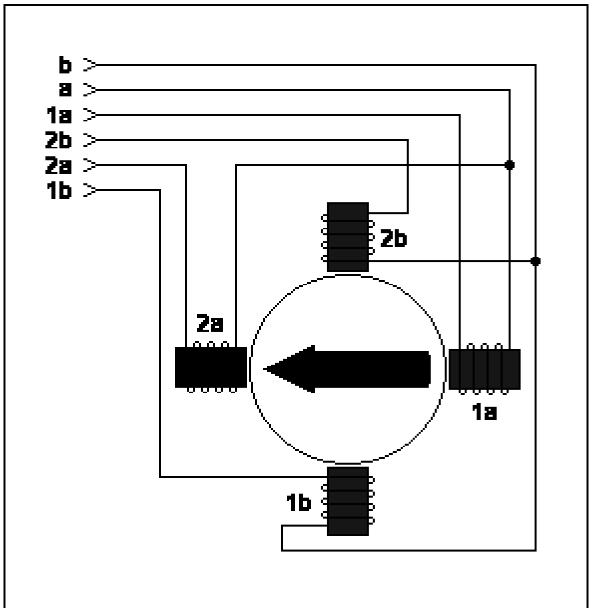
Then main applications areas of stepper motor are in Numerical Control (NC) systems for machine tools. Here they are utilized for driving the cutting tool along x, y, z directions. Another applications in this area is the co-ordinate table. Indexing mechanisms used in muItistation machine tools employ stepper motors for moving either work piece or cutting tools.

SPECI FICATION OF THE STEPPER MOTOR USED

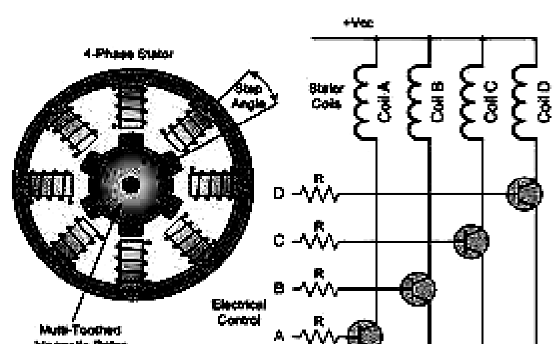
The stepper motor is reversible one with a torque of 3 kgcm. The power requirement is +5DC @1.2 A current per winding at full torque. The step angle is 1.8 degree i.e. tor the every single ex itatiou, the motor shafts rotates by 1.8 degree. for the motor to rotate one full revolution, number of steps required is 200. The stepper motor used has four stator windings which are brought out through colored wires terminated at a 4 pin polarized female connector.

STEPPER MOTOR HARDWARE MODULE DIAGRAM





Stepper Motor Operation (Unipolar, Half step)



10. Design and develop an assembly program to drive a Stepper Motor interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).

```
.model small

initds macro
    mov ax,@data        ; initializing the data segment
    mov ds,ax           ; it is ds, not dx
endm

init8255 macro
    mov al,cw        ; initialization of 8255 using control word
    mov dx,cr          by passing 82h to control reg.
    out dx,al          (to make port A as output)
endm

outpa macro
    mov dx,pa           ; initialization of port A as output
    out dx,al
endm

exit macro
    mov ah,4ch          ; to terminate
    int 21h
endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.data
    pa equ 1190h     ;One is Enough-setting the port address for port A
    cr equ 1193h
    cw db 82h        ; 82h is the value in control word 10000010, which
                         makes port A as output port
    steps db 200     ;step count
.code
    initds
    init8255

    mov al,88h       ; setting value in al 88=10001000
    mov bx,steps     ; taking count as 200 into BX

    rotate:

        outpa        ; perform rotation on port A
        call delay   ; have some delay in between the steps.
```

```
        ror al,01
        dec bx
        jnz rotate

      exit                  ; once the count becomes 0, call exit macro
```

clockwise direction- rotate right contents of al, i.e. 10001000 is rotated towards right by 1 bit. This makes the stepper motor to rotate clock wise direction. Then decrement the count and repeat the rotation process till it becomes 00 (200 times you rotate)

```
delay proc

    mov dx,00ffh
    outer:
        mov cx,0ffffh
    inner:
        loop inner
        dec dx
        jnz outer
        ret

delay endp
 end
```

****

OUTPUT:

Stepper motor is rotated clock wise and anti clock wise according to the step size

Formula to find step angle:

$$\text{Step angle} = \frac{360}{Rp * Sp} = \frac{360}{50*4} = 1.8^{\circ}$$

Rp --- no of rotor poles
Sp --- no of stator poles

For anti-clockwise rotation, do rotate AL towards left by 1 bit.

i.e
ROL AL,01

DUAL DAC INTERFACE (for 11a & 11b programs)

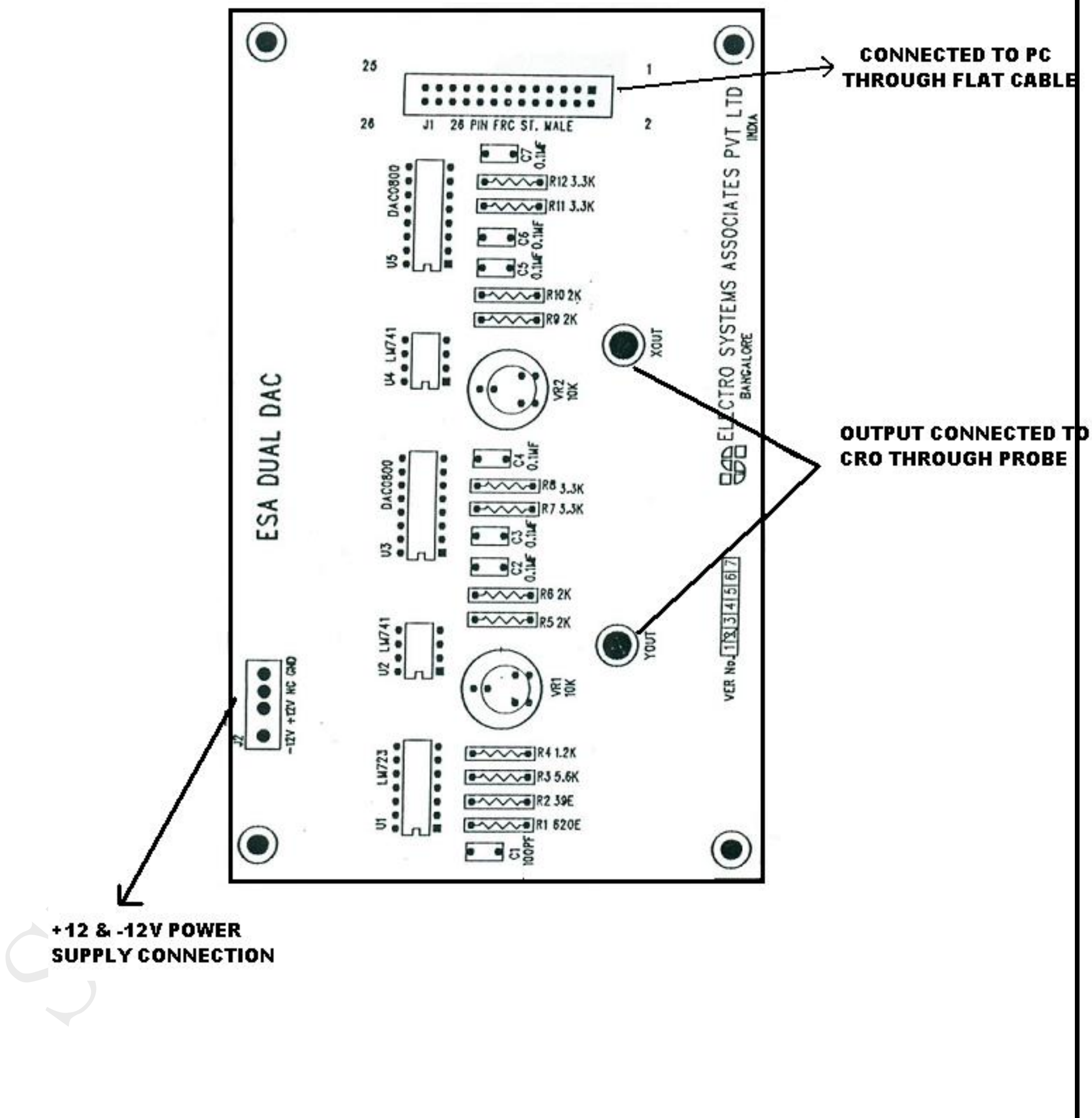
INTRODUCTION

The Dual DAC interface can be used to generate different interesting waveforms using microprocessor. These are two eight-bit digital to analog converters provided based all DAC 0800. The digital inputs to these DAC's arc provided through the port A and port B of 8255 used as output ports. The analog output from the DAC is given to operational amplifiers which act as current to voltage converters.

DESCRIPTION OFTHE CIRCUIT

The port A and port B of 8255 programmable peripheral interfaces are used as output ports. The digital inputs to the DAC s are provided through the port A and port B of 8255. The analog outputs of the DAC s are connected to the inverting inputs of the opamps 741 which act as current to voltage converters. The outputs from the opamps are connected to pins marked Xout & Yout at which the waveforms are observed on a CRO. The reference voltage for the DAC s is derived from an on-board voltage regulator 723. it generates a voltage of about 8V. The offset balancing of the opamps is done by making use of the two 10K pots provided. The output waveforms are observed at Xout & Yout on an oscilloscope.

DUAL DAC INTERFACE MODULE DIAGRAM



11a. Generate the Sine Wave using DAC interface (The output of the DAC is to be displayed on the CRO).

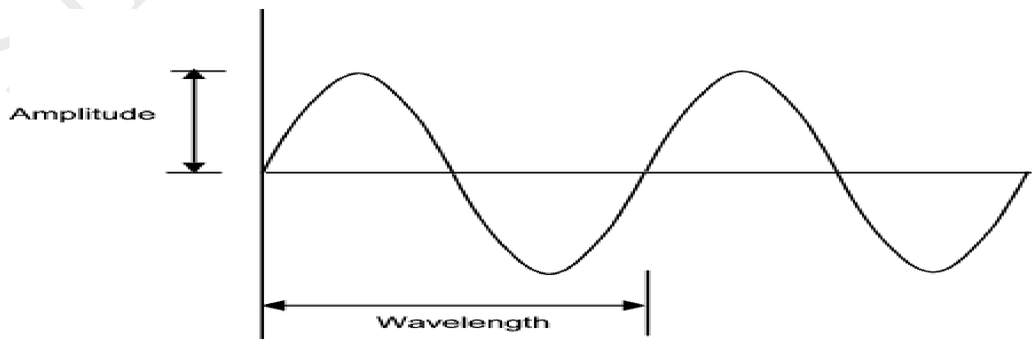
```
.model small

initds macro
    mov ax,@data          ; initializing the data segment
    mov ds,ax             ; it is ds, not dx
endm

init8255 macro
    mov al,cw          ; initialization of 8255 using control word
    mov dx,cr            by passing 82h to control reg.
    out dx,al            (to make port A as output)
endm

outpa macro
    mov dx,pa             ; initialization of port A as output
    out dx,al
endm

printf macro msg
    lea dx,msg            ; load the effective address to dx
    mov ah,9              ; function number is 9
    int 21h               ; using dos interrupt 21h
endm

exit macro
    mov ah,4ch            ; to terminate
    int 21h
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.data
    pa equ 1190h       ;One is Enough-setting the port address for port A
    cr equ 1193h
    cw db 82h          ; 82h is the value in control word 10000010, which
                             makes port A as output port

    table db 80H,96H,0ABH,0C0H,0D2H,0E2H,0EEH,0F8H,0FEH,0FFH;+ve 1st half
          db 0FEH,0F8H,0EEH,0E2H,0D2H,0C0H,0ABH,96H,80H ;+ve 2nd half
          db 69H,54H,40H,2DH,1DH,11H,07H,01H,00H   ;-ve 1st half
          db 01H,07H,11H,1DH,2DH,40H,54H,69H,80H   ;-ve 2nd half

    anykeytoexit db 10,13,"PRESS ANY KEY TO EXIT $"

  .code

    initds
    init8255
```

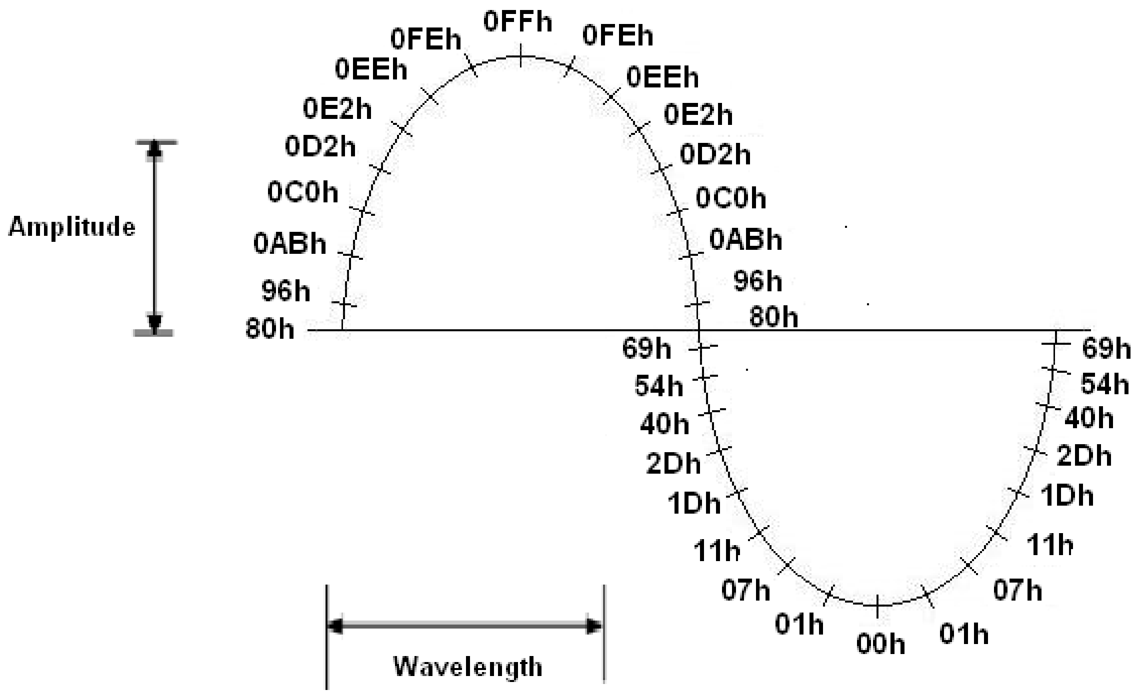
Look at the conversion table at the end of this program. Then you will understand these

```
    printf anykeytoexit         ;or you can use 25h

    start:
        mov cx,37             ;count value is taken 37 bcz the table
                                      contains 37  values
        lea si,table          ; table address is loaded to si
    back:
        mov al,[si]           ;the contents of si is moved to al i.e. single
                                      value of table is moved
        outpa                 ; moved value is sent to hardware module
                                      through port a
        call delay
        inc si                ; si is pointed to the next value of table
        loop back             ; loop repeats until all the contents of table
                                      is moved (till cx becomes 0)

        mov ah,1
        int 16h               ; checks if any key is pressed in keyboard. if
        jz start                       you haven't, then go to start

       exit                 ; if you press any key, just call exit macro


delay proc
    mov bx,0fffh                      ; note: single loop delay is enough
    inner:
        dec bx
        jnz inner                     ; you can't use CX as it is used to
                                      hold the count (37)
        ret
delay endp

end
```

****

OUTPUT:





conversion table for producing sin wave							
Formula V=128*128 sin θ							
+ve Values				**-ve Values**			
θ	**sinθ**	**v=128+(128*sinθ)**	**Hex**	**θ**	**sinθ**	**v=128+(128*sinθ)**	**Hex**
0	0	128	**80h**	0	0	128	**80h**
10	0.1736	150.22	**96h**	-10	-0.1736	105.78	**69h**
20	0.342	171.78	**0Abh**	-20	-0.342	84.22	**54h**
30	0.5	192.00	**0C0h**	-30	-0.5	64.00	**40h**
40	0.6428	210.28	**0D2h**	-40	-0.6428	45.72	**2Dh**
50	0.766	226.05	**0E2h**	-50	-0.766	29.95	**1Dh**
60	0.866	238.85	**0Eeh**	-60	-0.866	17.15	**11h**
70	0.9397	248.28	**0F8h**	-70	-0.9397	7.72	**07h**
80	0.9848	254.05	**0Feh**	-80	-0.9848	1.95	**01h**
90	1	255.00	**0FFh**	-90	-1	255.00	**0FFh**

11b. Generate a Half Rectified Sine waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).

```
.model small

initds macro
    mov ax,@data        ; initializing the data segment
    mov ds,ax           ; it is ds, not dx
endm

init8255 macro
    mov al,cw        ; initialization of 8255 using control word
    mov dx,cr          by passing 82h to control reg.
    out dx,al          (to make port A as output)
endm

outpa macro
    mov dx,pa             ; initialization of port a as output
    out dx,al
endm

printf macro msg
    lea dx,msg          ; load the effective address to dx
    mov ah,9            ; function number is 9
    int 21h             ; using dos interrupt 21h
endm

exit macro
    mov ah,4ch          ; to terminate
    int 21h
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.data
    pa equ 1190h     ;One is Enough-setting the port address for port A
    cr equ 1193h
    cw db 82h        ; 82h is the value in control word 10000010, which
                           makes port A as output port

    table db 80H,96H,0ABH,0C0H,0D2H,0E2H,0EEH,0F8H,0FEH,0FFH;+ve 1st half
          db 0FEH,0F8H,0EEH,0E2H,0D2H,0C0H,0ABH,96H,80H ;+ve 2nd half
          db 80H,80H,80H,80H,80H,80H,80H,80H,80H   ;all zeros (T-OFF)
          db 80H,80H,80H,80H,80H,80H,80H,80H,80H   ;all zeros (T-OFF)

    anykeytoexit db 10,13,"PRESS ANY KEY TO EXIT $"

.code

    initds
    init8255
```

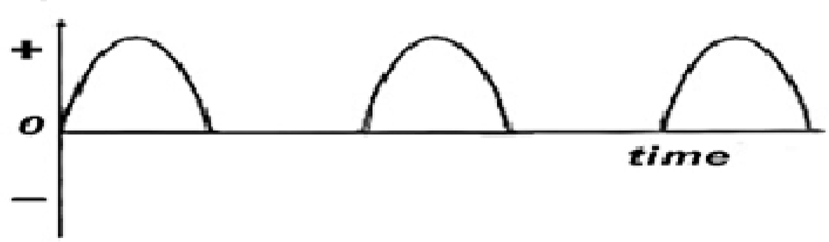
This is the only change in this pgm

Look at the conversion table at the end of this program. Then you will understand these

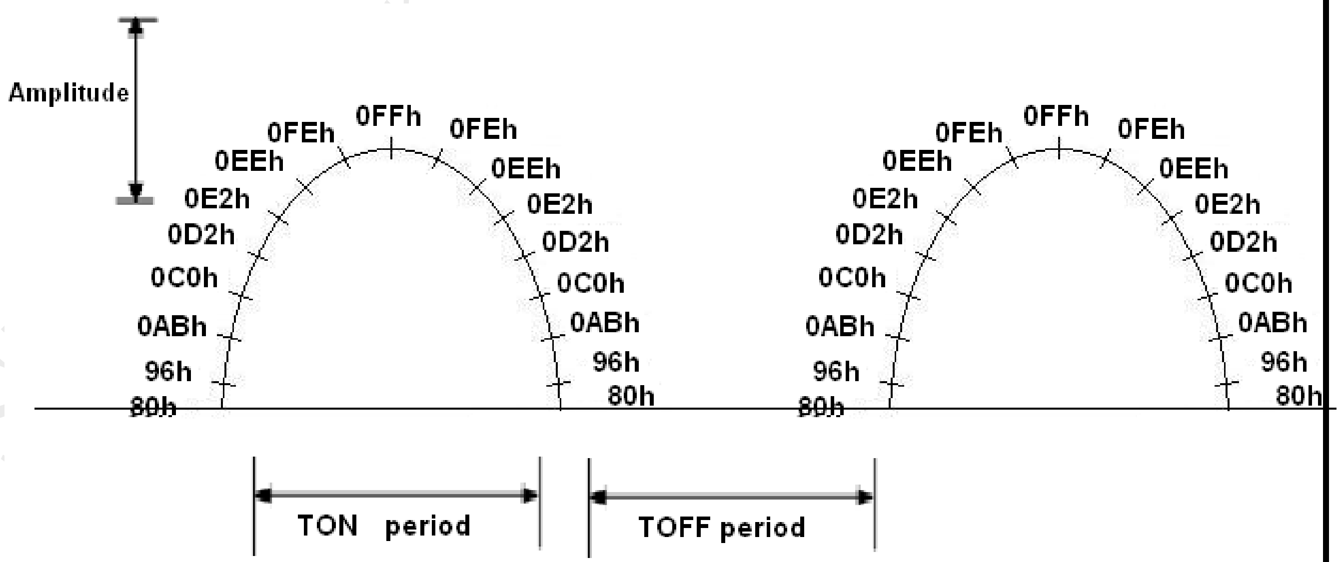
```
    printf anykeytoexit         ;or you can use 25h

    start:
        mov cx,37              ;count value is taken 37 bcz the table
                                    contains 37  values
        lea si,table           ; table address is loaded to si
    back:
        mov al,[si]            ;the contents of si is moved to al i.e. single
                                    value of table is moved
        outpa                  ; moved value is sent to hardware module
                                    through port a
        call delay
        inc si                 ; si is pointed to the next value of table
        loop back              ; loop repeats until all the contents of table
                                    is moved (till cx becomes 0)

        mov ah,1
        int 16h                ; checks if any key is pressed in keyboard. if
                                     you haven't, then go to start
        jz start
        exit                   ; if you press any key, just call exit macro


 delay proc
    mov bx,0fffh               ; note: single loop delay is enough
    inner:
        dec bx
        jnz inner               ; you can't use CX as it is used to hold the
                                    count (37) in our above program

        ret
delay endp

end
```

****

OUTPUT:



Conversion table for producing sin wave			
Formula V=128*128 sin θ			
θ	**sinθ**	**v=128+(128*sinθ)**	**Hex**
0	0	128	80h
10	0.1736	150.22	96h
20	0.342	171.78	0ABh
30	0.5	192.00	0C0h
40	0.6428	210.28	0D2h
50	0.766	226.05	0E2h
60	0.866	238.85	0EEh
70	0.9397	248.28	0F8h
80	0.9848	254.05	0FEh
90	1	255.00	0FFh



*************** All the best guys! **************