Microcontroller Interfacing

LCD Interfacing :-

LCD Operation

Advantages of LCD over LEDs

1. Declining prices of LCDs.
2. Ability to display numbers, characters and graphics. LEDs are limited to numbers and few characters.
3. Incorporation of refreshing controller into LCD, thereby relieving the CPU of the task of refreshing LCD. LEDs must be refreshed by the CPU to keep displaying the data.
4. Ease of programming for characters and graphics.

LCD pin descriptions (14 pin) :-

Pin	Symbol	I/O	Description.
1	V_{SS}	–	Ground
2	V_{CC}	–	+5V power supply
3	V_{EE}	–	Power supply to control contrast.
4	RS	I	RS=0 to select command register, RS=1 to select data register.
5	R/W	I	R/W=0 for write, R/W=1 for read
6	E	I/O	Enable
7	DB0	I/O	8-bit data bus
8	DB1	I/O	–"–
9	DB2	I/O	–"–
10	DB3	I/O	–"–
11	DB4	I/O	–"–

Pin	Symbol	I/O	Description
12	DB5	I/O	-11-
13	DB6	I/O	-11-
14	DB7	I/O	-11-

RS (Register select)

If RS=0, instruction command code register is selected, to send command such as clear display, cursor at home etc.

If RS=1, data register is selected, to send data to be displayed on LCD.

R/W (Read/write)

R/W=1, read or else write.

E (Enable)

High-to-low pulse latches data present on its data pins. It must be 450 ns wide (minimum).

D0-D7

Used to read or send information to LCD's internal registers.

To display (A-Z), (a-z) and (0-9), ASCII codes are sent with RS=1.

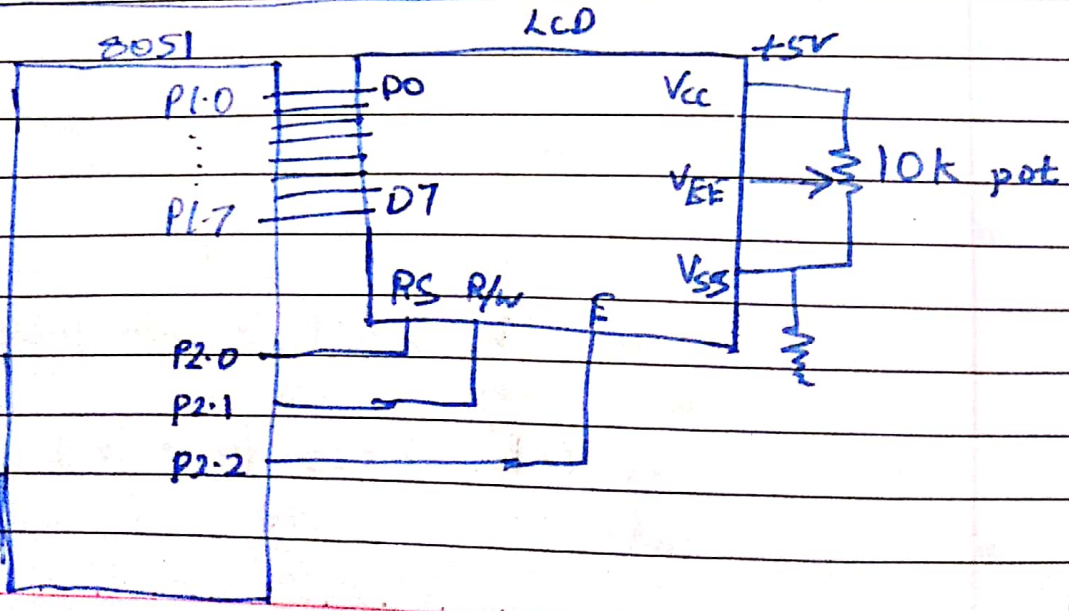
To write data to LCD busy flag D7 has to be checked by making R/W=1 and RS=0. If D7=1, LCD is busy performing internal operation, hence will not receive any information

To send command to LCD, RS=0 and apply high-to-low on pin E to enable internal latch.

LCD command codes.

Code (Hex)	Command to LCD Instruction Register.
1	Clear display screen
2	Return home
4	Decrement cursor (shift left)
6	Increment cursor (shift right).
5	Shift display right.
7	Shift display left.
8	Display off, cursor off.
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	—11—
10	Shift cursor position to left.
14	—11— right
18	Shift entire display to left
1C	—111— right.
80	Force cursor to beginning of first line
C0	—11— Second line
38	2 lines and 5×7 matrix.

Interconnection with 8051



Program to send data or command to LCD with delay

P1.0 - P1.7 are connected to D0-D7, P2.0 is connected to RS, P2.1 to R/w pin, P2.2 to E pin.

```
        org 0h
        Mov a,#38h    ; init. 2 lines, 5x7 matrix
        acall delay
        mov a,#0Eh
        acall commwrt
        acall delay
        Mov a,#01
        acall commwrt
        Acall delay
        mov a,#06h
        acall commwrt
        acall delay
        mov a,#84h
        acall commwrt
        acall delay
        mov a,#'N'
        acall datawrt
        acall delay
        Mov a,#'O'
        Acall data wrt
again:  sjmp again
commwrt: mov P1,A
        clr P2.0    ; Rs=0 for command
        clr P2.1    ; R/w=0 for write
        setb P2.2   ; E=1 for high pulse
        acall delay
        clr P2.2    ; E=0 for H-to-L pulse
        ret
```

```
datawrt: mov P1, A
         setb P2.0  ; RS=1 for data
         setb P2.2  ; E=1 for high pulse.
         acall delay
         clr P2.2   ; E=0 for H-to-L pulse.
         ret
delay:   mov R3, #50 ; 50 or higher for fast CPUs
herez:   mov R4, 255
here:    djnz R4, $
         djnz R3, herez
         ret
         end
```

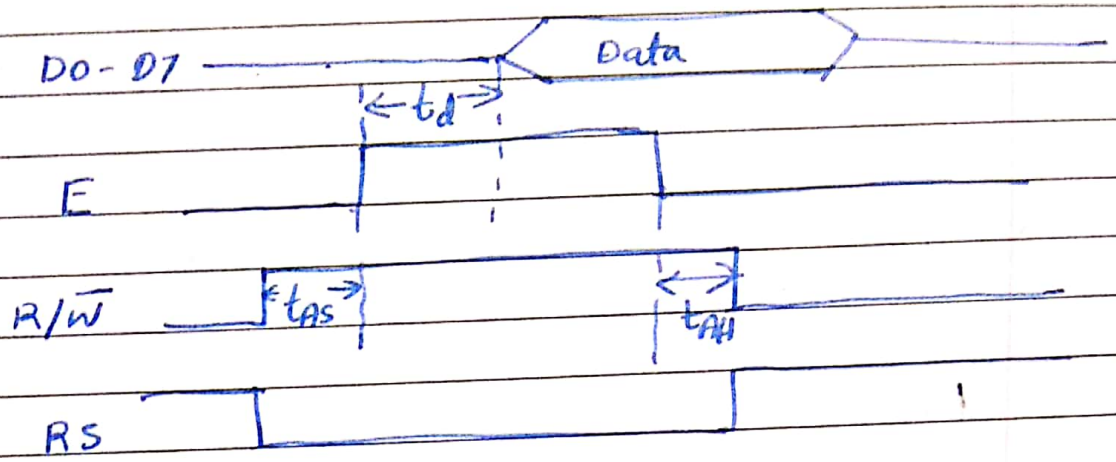
Above code sends commands to LCD without checking busy flag. Following code checks busy flag before sending data or command to LCD.

```
mov a, #38h  ; init 2 lines 5X7 matrix.
acall command
mov a, #0Eh  ; LCD on, cursor on
acall command
mov a, #01h  ; clear LCD
acall command
mov a, #06h  ; shift cursor right.
acall command
mov a, #86h  ; cursor line 1, pos. 6
acall command
mov a, #'N'
acall data_display
mov a, #'O'
acall data_display
here: sjmp $
```

```
command:    acall   ready     ; LCD ready?
            mov     P1, A     ; issue command code
            clr     P2.0      ; RS=0 for command
            clr     P2.1      ; R/W=0 to write to LCD
            setb    P2.2      ; E=1 for H-to-L pulse
            clr     P2.2      ; E=0, latch in
            ret

data_display: acall ready     ; LCD ready?
            mov     P1, A     ; issue data
            setb    P2.0      : RS=1 for data
            clr     P2.1      ; R/W=0 to write to LCD
            setb    P2.2      ; E=1 for H-to-L pulse
            acall   delay
            clr     P2.2      ; E=0, latch in
            ret

ready:      setb    P1.7      ; P1.7 input port
            clr     P2.0      ; RS=0, access command reg
            setb    P2.1      ; R/W=1, read command reg

back        : clr   P2.2 ;  E=0 for L-to-H pulse
            acall   delay
            setb    P2.2 ;  E=1 for L-to-H pulse
            jb      P1.7, back ; wait till busy flag=0
            ret
            end.
```

In the above program no delay subroutine is used since busy flag is checked. D7=1 LCD is busy, D7=0 LCD can accept data or command.

LCD timing for Read. (L-to-H for E line)

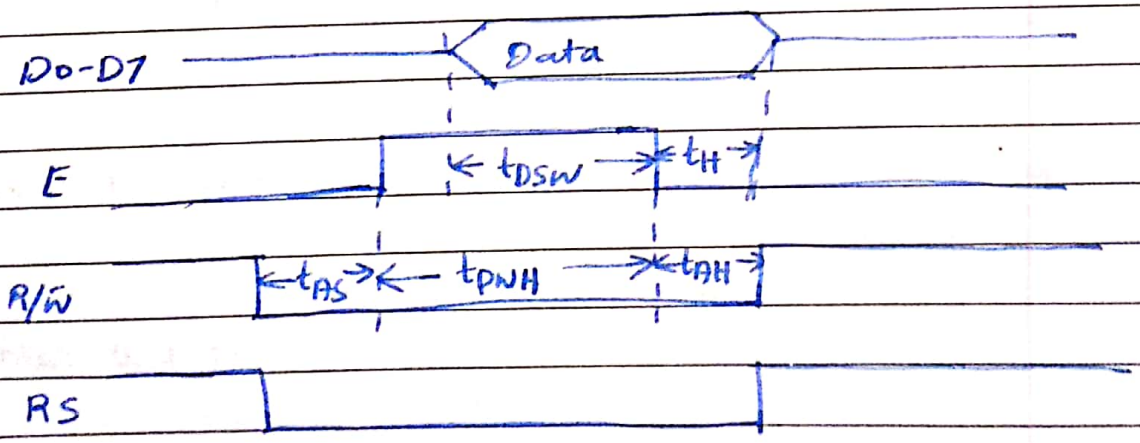


t_d – data output delay time

t_{AS} – setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)

t_{AH} – hold time after E has come down for both RS and R/W = 10 ns (minimum).

LCD timing for write (H-to-L for E line)



t_{PWH} – enable pulse width = 450 ns (minimum)

t_{DSW} – data setup time = 195 ns (minimum)

t_H – data hold time = 10 ns (minimum)

t_{AS} – setup time prior to E (going high) for both RS and R/W = 140 ns (minimum).

t_{AH} – hold time after E has come down for both RS and R/W = 10 ns (minimum).

LCD data sheet

In LCD data can be stored at any location.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

AAAAAAA = 0000000 to 0100111 for line1 and 1000000 to 1100111 for line 2.

For 40 character wide, upper address range = 0100111 (39)
For 20 character wide, upper address range = 010011 (19)

Cursor address

16 X 2	80	81	through	8F
	C0	C1	through	CF
20 X1	80	81	through	93
20 X2	80	81	through	93
	C0	C1	through	D3
40 X2	80	81	through	A7
	C0	C1	through	E7

Write an 8051 c program to send letters 'M' 'D' and 'E' to LCD using delays.

```
#include <reg51.h>
sfr ldata = 0x90;  // P1= LCD data pins
sbit rs = P2^0;
sbit rw = P2^1;
sbit en = P2^2;
void main()
{ lcdcmd(0x38);
  msdelay(250);
  lcdcmd(0x0E);
  ms delay(250);
```

```
lcdcmd (0x01);
msdelay (250);
lcdcmd (0x06);
msdelay (250);
lcdcmd (0x86);
msdelay (250);
lcddata ('M');
msdelay (250);
lcddata ('D');
msdelay (250);
lcddata ('E'); }
void lcdcmd(unsigned char value)
{   ldata = value;
    rs = 0;
    rw=0;
    en=1;
    msdelay(1);
    en=0;
    return;
}
void lcddata(unsigned char value)
{   ldata = value;
    rs=1;
    rw=0;
    en=1;
    msdelay(1);
    en=0;
    return; }
void msdelay(unsigned int itime)
{ unsigned int i,j;
  for (i=0;i<itime;i++)
  for (j=0;j<1275;j++); }
```

Above program is rewritten using busy flag method.

```
#include<reg51.h>
sfr ldata = 0x90;
sbit rs = P2^0;
sbit rw = P2^1;
sbit en = P2^2;
sbit busy = P1^7;
void main()
{ lcdcmd (0x38);
  lcdcmd (0x0E);
  lcdcmd (0x01);
  lcdcmd (0x06);
  lcdcmd (0x86);
  lcddata ('M');
  lcddata ('D');
  lcddata ('E'); }
void lcdcmd (unsigned char value)
{ lcdready ();
  ldata = value;
  rs = 0;
  rw = 0;
  en = 1
  msdelay(1);
  en = 0;
  return; }
void lcddata(unsigned char value)
{ lcdready();
  ldata = value;
  rs = 1;
  rw = 0;
  en = 1
  msdelay (1);
```

```
en=0;
return; }
Void lcdready()
{ busy=1;    // P1.7 input pin
  rs=0;
  rw=1;
  while (busy==1)
  { en=0;
    msdelay(1);
    en=1; }
  return; }
Void msdelay(unsigned int itime)
{ unsigned int i,j;
  for(i=0; i<itime,i++)
  for(j=0 ;j<1275;j++) ;}
```

Keyboard Interfacing

Keyboards are organized in a matrix of rows and columns. The CPU access both rows and columns through ports. Two ports connects 8X8 Matrix of keys. When a key is pressed a row and column make contact. Microcontroller scan the keys continuously, identify which one has been activated.

Scanning and identifying key :-

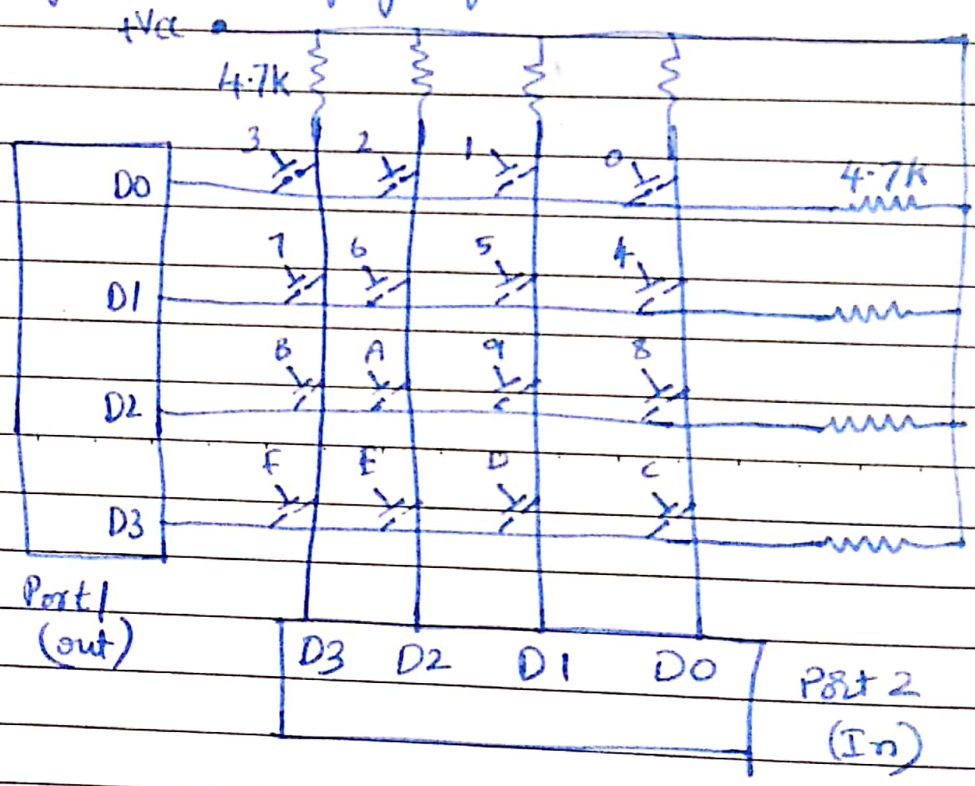


Figure above shows 4X4 matrix connected to two ports. Rows are connected to output ports and columns are connected to input port. All columns are connected to +Vcc. If no key is pressed, all input port pin reads 1's. If all the rows are grounded and a key is pressed, that particular column reads 0, because it is grounded via the row.

To detect a pressed key, microcontroller grounds all rows by providing 0 to the output latches. Then it reads the columns.

If data read from column, D3-D0 = 1111, no key has been pressed.
If D3-D0 = 1101, a key in 01 column has been pressed.
Microcontroller identifies the key by grounding row to zero.
If all column read 1's then no key in that row is pressed. If one of the bit in the column reads 0, then the row has a key pressed.

For example:-
Row D3-D0 = 1110, column D3-D0 = 1011.
In row D0 = 0, and column D2 = 0, the key pressed = 2.

8051 ALP for detection and identification of key activation.
Keyboard subroutine sends ASCII code for key pressed to P0.1.
P1.0 to P1.3 connected to rows, P2.0 to P2.3 connected to columns.

```
        MOV P2,#0FFH        ; P2 input port.
K1:     MOV P1,#00          ; ground rows.
        MOV A,P2            ; read all columns.
        ANL A,#00001111B    ; masked unused bits.
        CJNE A,#00001111B,K1 ; check till all keys are released
K2:     ACALL delay         ; 20 ms delay
        MOV A,P2            ; see if any key is pressed.
        ANL A,#00001111B    ; mask unused bits
        CJNE A,#00001111B   ; key pressed, await closure
        SJMP K2             ; check if key pressed.
OVER:   ACALL delay         ; wait 20ms debounce time
        MOV A,P2            ; check key closure.
        ANL A,#00001111B    ; masks unused bits
        CJNE A,#00001111B,OVER1 ; key pressed, find row
        SJMP K2             ; if not, keep polling
```

```
OVER1: MOV P1,#11111110B   ; ground row 0
       MOV A,P2            ; read all columns
       ANL A,#00001111B    ; mask unused bits
       CJNE A,#00001111B,ROW_0 ; Key row 0, find column
       MOV P1,#11111101B   ; ground row 1
       MOV A,P2            ; read all columns
       ANL A,#00001111B    ; mask unused bits
       CJNE A,#00001111B,ROW_1 ; Key row 1, find column
       MOV P1,#11111011B   ; ground row 2
       MOV A,P2            ; read all columns
       ANL A,#00001111B    ; mask unused bits.
       CJNE A,#00001111B,ROW_2 ; Key row 2, find col.
       MOV P1,#11110111B   ; ground row 3
       MOV A,P2            ; read all columns
       ANL A,#00001111B    ; mask unused bits
       CJNE A,#00001111B,ROW_3 ; Key row 3, find column
       ljmp K2             ;if none false input, repeat

ROW_0: MOV DPTR,#KCODE0    ; set DPTR = start of row 0
       SJMP FIND           ;find column, key belongs to
ROW_1: MOV DPTR,#KCODE1    ;
       SJMP FIND
ROW_2: MOV DPTR,#KCODE2
       SJMP FIND
ROW_3: MOV DPTR,#KCODE3
FIND:  RRC A               ; see if any CY bit is low
       JNC MATCH           ; if zero, get the ASCII code
       INC DPTR            ; point to next column address
       SJMP FIND           ; keep searching
```

```
match: clr A          ; set A=0 (a match is found)
       movc A,@A+DPTR ; get ASCII code from table
       mov P0,A       ; display pressed key
       ljmp K1
; ASCII look up table for each row
       org 300H
KCODE0: DB '0','1','2','3'     ; row 0
KCODE1: DB '4','5','6','7'     ; row 1
KCODE2: DB '8','9','A','B'     ; row 2
KCODE3: DB 'C','D','E','F'     ; row 3
       end
```

① To make sure that preceding key has been ~~pressed~~ released, 0's are output to all rows at once and the columns are read and checked repeatedly until all the columns are high. When all columns are high, the program waits for a short amount of time before next stage of waiting for a key to be pressed.

② To see if any key pressed, the columns are scanned over and over again until one of them has a '0' on it. All the rows are grounded via latches connected to them. After keypress detection, microcontroller waits 20 ms for the bounce and scans the column again. The purpose is to
(i) ensure that the first key press detection was not an erroneous one due to spike noise, and
(ii) 20 ms delay prevents the same key press from being interpreted as a multiple key press. If after 20 ms delay the key is still pressed, it goes to the next stage to detect which row it belongs to, otherwise, it goes back to loop to detect a real key press.

(3) To detect which row the keypress belongs to, microcontroller grounds one row at a time, reading the columns each time. If all columns are high, keypress cannot belong to that row. It grounds next row and continues until it finds the row the keypress belongs to. If it finds the row, it sets up the starting address for the look-up table holding the scan code or ASCII value for that row and goes to next stage to identify the key.

(4) To identify key press, microcontroller rotates the column bits, one bit at a time. into the carry flag to check for low condition. If zero, it pulls out the ASCII code for that key from look-up table or else it goes to the next element of the look-up table.

ADC and DAC Interfacing

Analog to Digital Converter :-

Parallel and Serial ADC :-

ADCs are widely used for data acquisition. Digital computers use binary 0 and 1 values but in real world everything is analog. Example include temperature, pressure, humidity and velocity.

Physical signals are converted to analog using transducers or sensors. The analog signals are electrical signal like voltage or current. ADCs are required to convert analog to digital so that the microprocessor or controller can process them.

An ADC has n-bit resolution where n can be 8, 10, 12, 16 or 24 bits. Higher resolution ADC provides smaller step-size. Conversion time of ADC is the time taken by ADC to convert input analog signal to digital signal.

In parallel ADCs, all the digital outputs are available at the same time where as in serial ADCs the output bits are available through only one pin.

ADC 0804 chip :-

It is an 8-bit parallel ADC. It works with +5V volts and has a resolution of 8 bits. Conversion time varies depending on clock signal applied to CLK IN pin but not faster than 110 μs.

PIN Description :-

CS - chip select is an active low input used to activate. This pin must be low to activate.

RD - Read (output enable)

This is an active low input signal. The ADC converts analog input to its digital equivalent and holds it in internal register. RD signal is used to read bits stored in register. With CS=0, H-to-L pulse on RD pin, 8-bit digital output appears at D0-D7 data pins.

WR (write):-

It is an active low signal applied to start conversion. With CS=0, a low-to-high transition converts analog input V_{in} to an 8-bit digital data. Conversion time depends on CLKIN and CLKR values. When conversion is complete INTR goes low.

CLKIN and CLKR:-

CLKIN pin is used to apply external clock source used for timing. To use internal clock generator (self-clocking), CLKIN and CLKR pins are connected to a capacitor and resistor.

$$\text{clock frequency} = \frac{1}{1.1RC}$$

Typically, R = 10K Ω, C = 150pF, f = 606 kHz, conversion time = 110μs.

INTR (Interrupt).

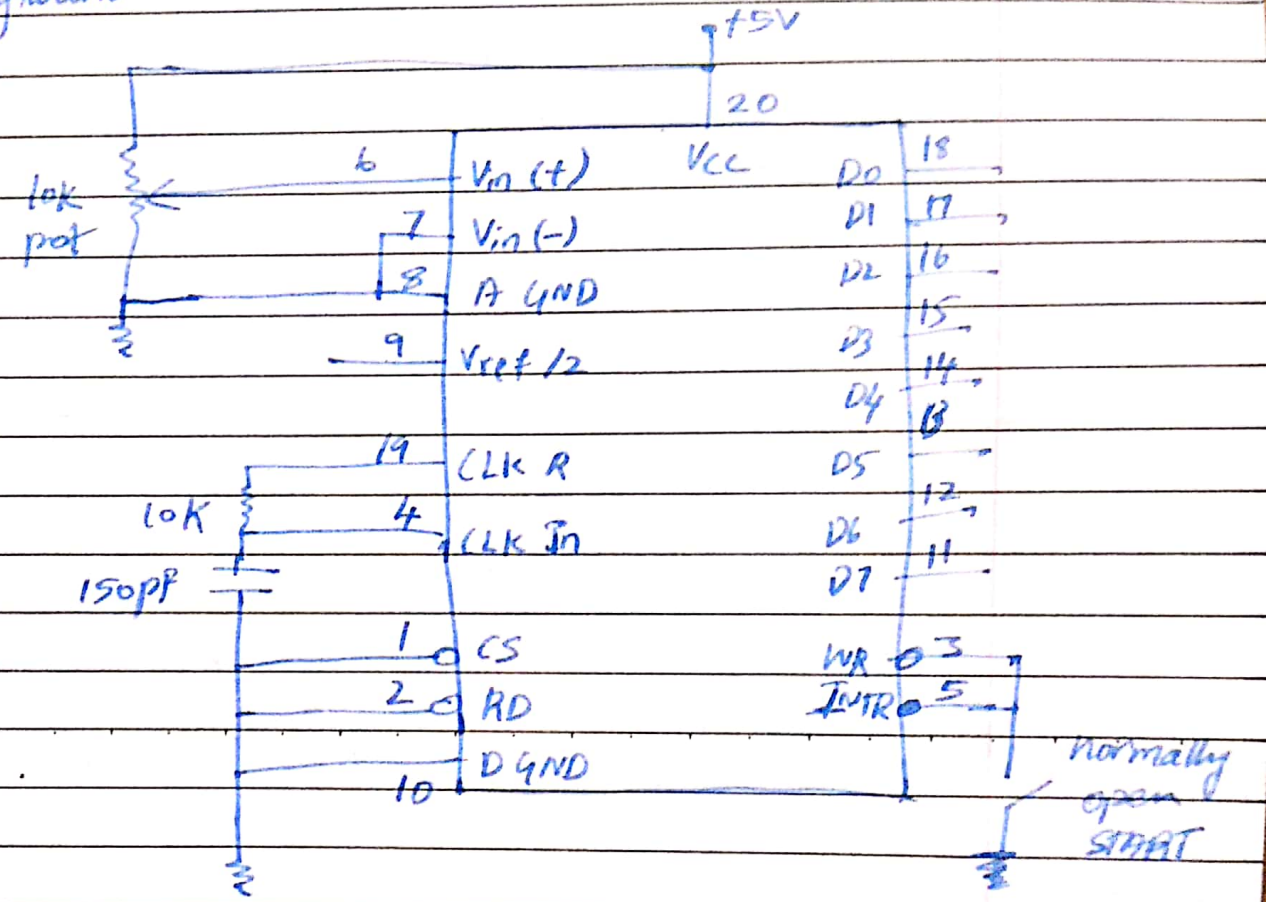
When conversion is complete, pin goes low otherwise will be in high position. Later CS=0 and H-to-L pulse is applied to RD pin to read the digital data output

$V_{in}(+)$ and $V_{in}(-)$:-

They are differential analog input where,

$V_{in} = V_{in}(+) - V_{in}(-)$.

$V_{in}(+)$ is connected to analog input and $V_{in}(-)$ is connected to ground.



ADC 0804 Chip

Vref Relation to Vin Range {Vcc = 5V}

(Vref/2) (V)	Vin (V)	step size (mV)
not connected (open)	0 to 5	5/256 = 19.53
2.0	0 to 4	4/256 = 15.62
1.5	0 to 3	3/256 = 11.71
1.28	0 to 2.56	2.56/256 = 10

Vcc: +5V power supply. Also used as a reference when Vref/2 input (pin 9) is open.

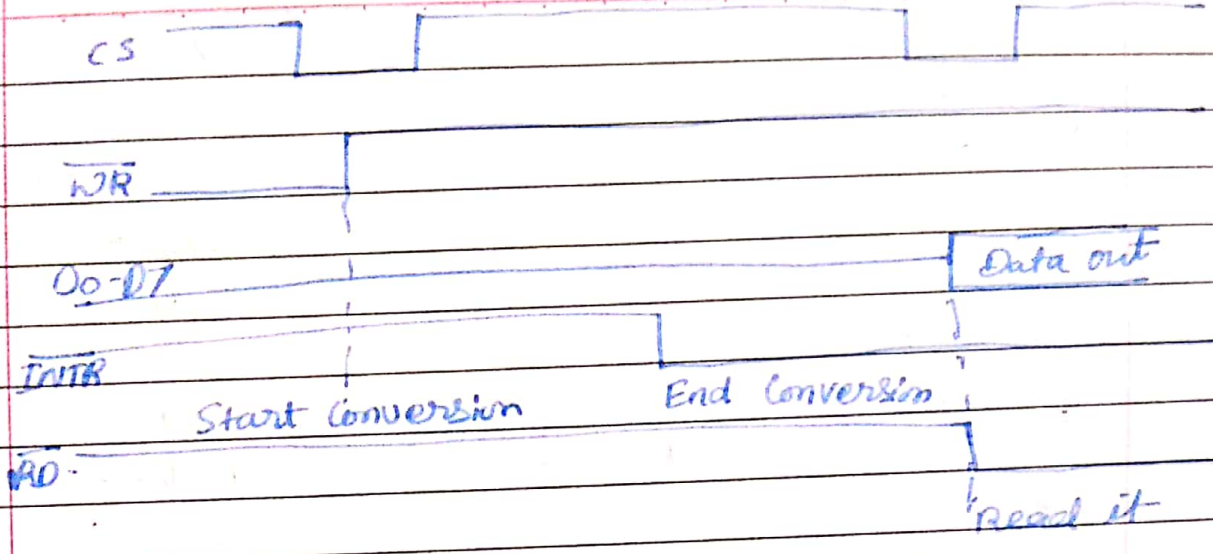
$V_{ref}/2$:- Pin 9 is an input voltage used for reference voltage. If open, analog input voltage is in the range of 0 to 5 volts (same as V_{cc}). Some applications require voltage to be other than 5V. $V_{ref}/2$ is used to implement these voltages. Example. If input range needs to be 0 to 4 volts, $V_{ref}/2$ is connected to 2V.

D0-D7:- Digital data output pins. They are tristate buffered and converted data is accessed when CS=0 and RD forced low. To calculate output voltage

$$D_{out} = \frac{V_{in}}{\text{Step Size}} \quad ; \quad \text{step size} = \frac{\{2 \times V_{ref}/2\}}{256}.$$

Analog ground and digital Ground :-
These are input pins providing ground for analog and digital signal. AGND is connected to ground of analog V_{in} and DGND is connected to ground of V_{cc}. The reason is to separate analog V_{in} signal from transient voltage caused by digital switching of D0-D7. This isolation helps accuracy of digital data output.

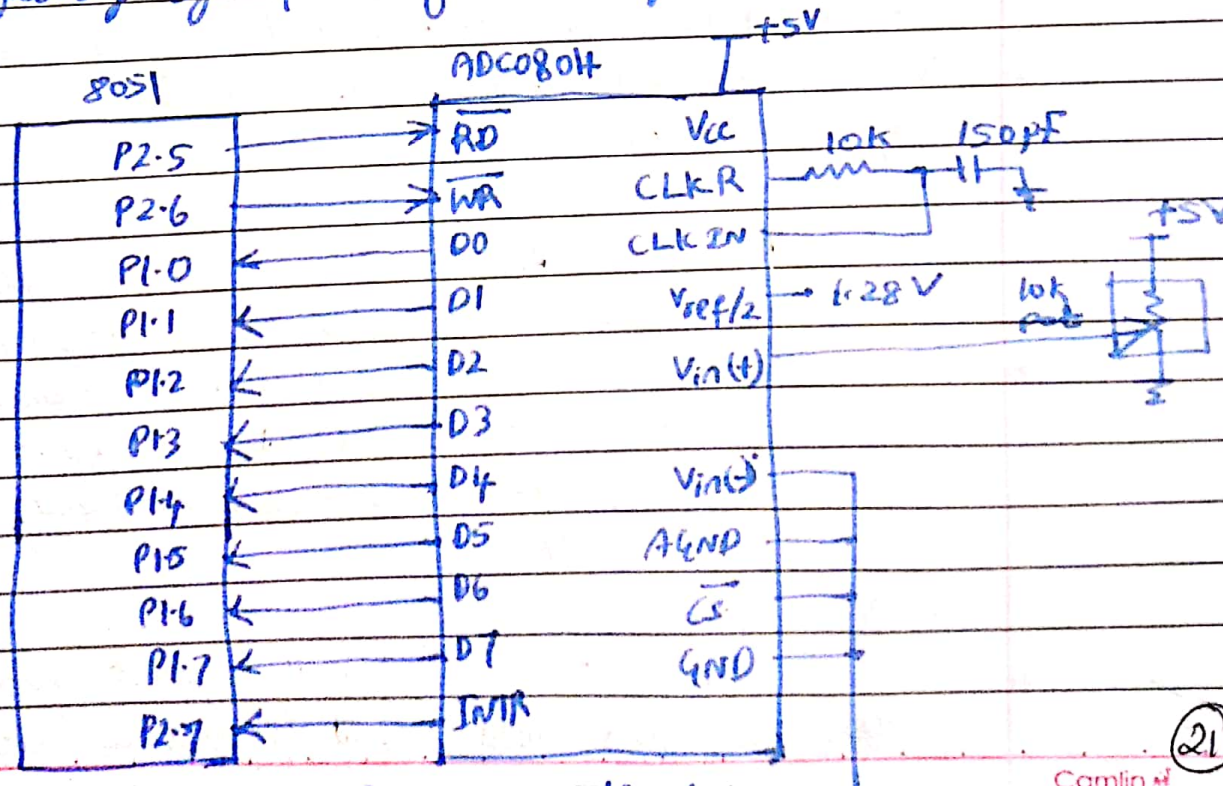
Steps followed when using ADC0804 :-
1. Make CS=0, send L-to-H pulse to WR to start conversion.
2. Keep monitoring INTR pin. If low, conversion finished can jump to next step. If high, keep polling until it goes low.
3. If INTR=0, make CS=0 and send H-to-L pulse to RD pin to get data output.



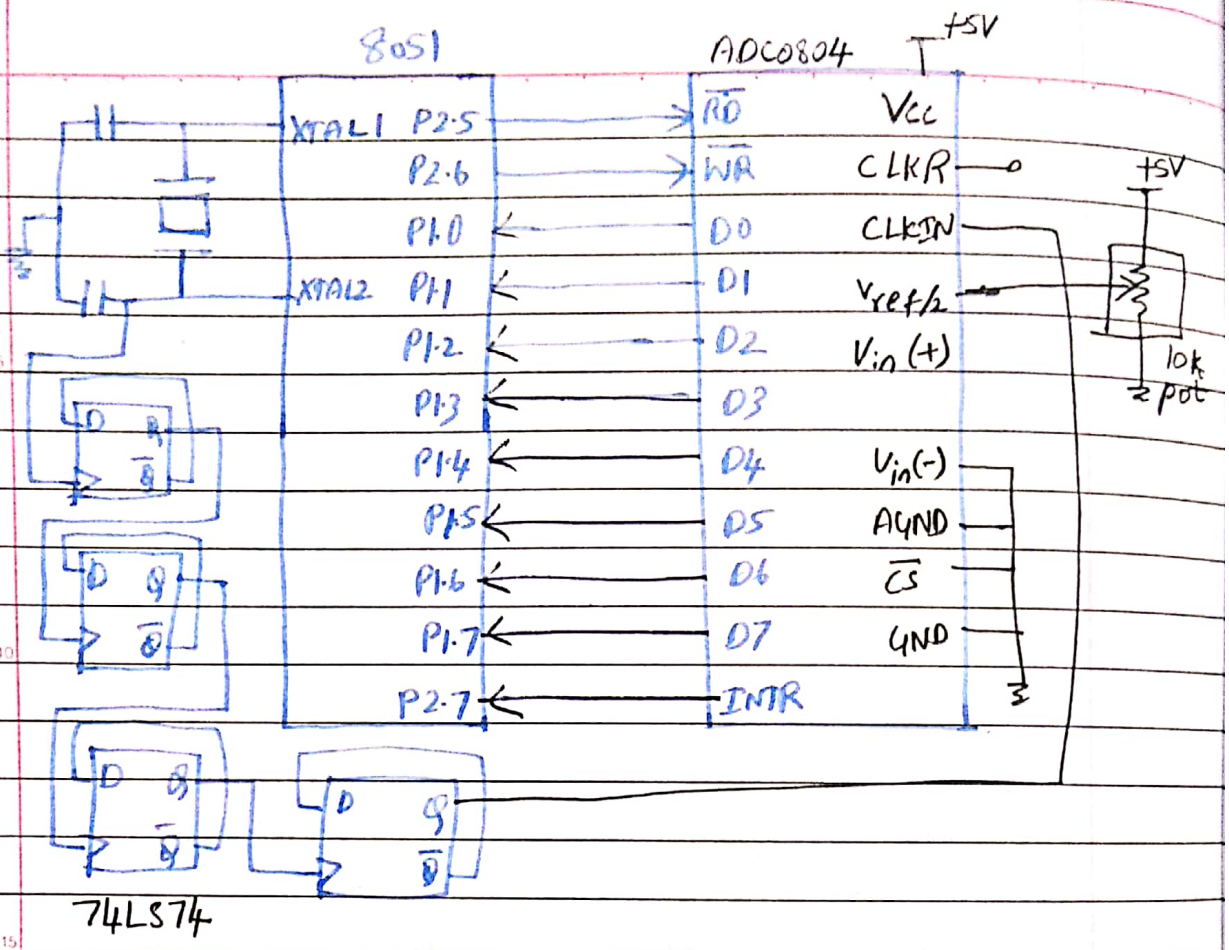
Read and Write Timing for ADC0804

Clock source :-

The conversion speed depends on the clock. Typical operating frequency is 640kHz at 5Volts. There are two ways of providing clock to the ADC0804. Using self clocking or using crystal of the microcontroller. Since microcontrollers frequency is very high, reduce it using D-FF. one DFF reduces frequency by a factor of 2. For higher frequencies use 4 DFFs.



8051 interface to ADC0804 with self clocking.



8051 connection to ADC0804 with clock from XTAL2.

Programming

Write an ALP to monitor INTR pin and bring an analog input into register A. Call hex-to-ASCII conversion and data display subroutines.

```
RD      BIT P2.5
WR      BIT P2.6      ; start conversion
INTR    BIT P2.7      ; end-of-conversion
MYDATA  EQU P1        ; P1.0 to P1.7 = D0 to D7.
        MOV P1, #0FFH ; make P1= input port
        SETB INTR
BACK:   CLR  WR
        SETB WR       ; L-to-H to start conversion.
HERE:   JB   INTR, HERE ; wait for end-of conversion.
```

```
CLR RD          ; enable RD
MOV A, MYDATA   ; read data.
ACALL CONVERSION
ACALL DATA_DISPLAY
SETB RD
SJMP BACK
; hex-to-ASCII conversion.

CONVERSION: RAM_ADDR EQU 40H
            ASCII_RESULT EQU 50H
            COUNT EQU 3
            ORG 0
            ACALL BIN_DEC_CONVERT
            ACALL DEC_ASCII_CONVERT
            SJMP $

; Binary to decimal conversion (00-FF to 000 to 255).
BIN_DEC_CONVERT: MOV R0, #RAM_ADDR ; create pointer
                 MOV A, P1         ; read data from P1.
                 MOV B, #10
                 DIV A,B
                 MOV @R0, B        ; save lower byte
                 INC R0
                 MOV B, #10
                 DIV AB
                 MOV @R0, B        ; save next by digit.
                 INC R0
                 MOV @R0, A
                 RET
```

DEC_ASCI_CONVERT:

```
        MOV  R0, #RAM_ADDR ; address of DEC data
        MOV  R1, #ASCI_RSULT ; address of ASCII data
        MOV  R2, #3        ;count
back:   MOV  A, @R0        ;get DEC digit
        ORL  A, #30H       ;convert it into ASCII
        MOV  @R1, A        ; save it.
        INC  R0            ; next digit
        INC  R1
        DJNZ R2, back
        RET
```

DAC Interfacing

Digital to Analog converter is a device which converts digital pulses to analog signals.
There are two methods used in conversion: binary weighted and R-2R ladder network. DAC0808 uses R-2R method because it gives higher degree of precision.

Important feature of a DAC is resolution, which is a function of number of binary inputs. Example: 8, 10 & 12 bit. The analog output voltage levels depends on number of input bits. For example, an 8-bit DAC provides $2^n = 2^8 = 256$ discrete voltage or current levels at output.

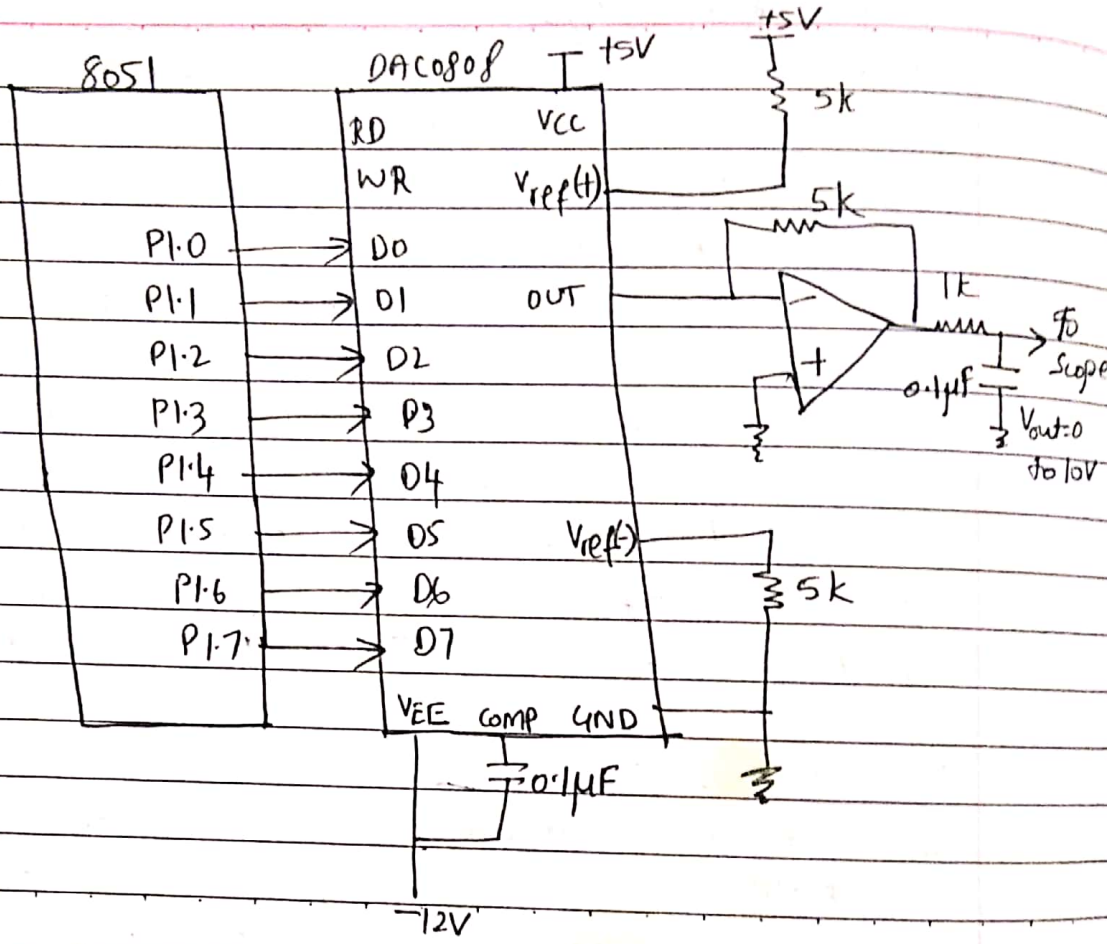
MC1408 DAC (DAC0808) :-

DAC 0808 is a 8-bit DAC. Digital inputs are converted to current at the output. By using resistor, current can be converted to the voltage. The magnitude of output current is proportional to input binary data and reference current.

$$I_{out} = I_{ref} \left\{ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right\}$$

where D0 is LSB and D7 is MSB.
Iref is the input current applied to pin 14. Iref is generally set to 2.0 mA generated using 5V, 1kΩ and 1.5kΩ resistors.

Example: Iref = 2 mA, digital input D0 - D7 = 11111111, then maximum output current = 1.99 mA.



8051 Interfaced to DAC0808

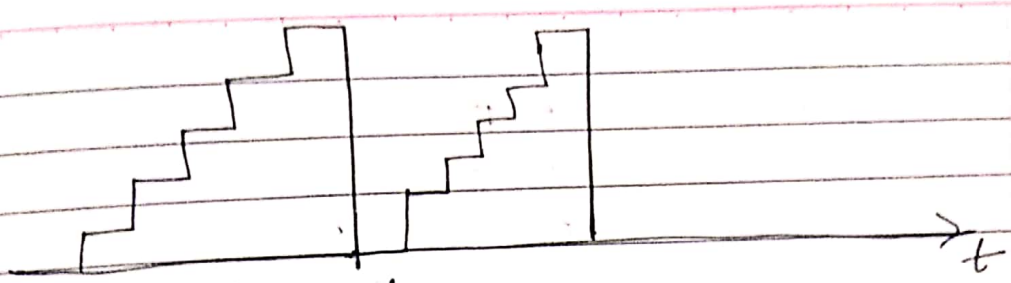
Converting Iout to Voltage :-

To convert output current to voltage, resistor can be employed but causes inaccuracy because load resistance also adds up to this resistor. Hence load is isolated from ~~out~~ Iout using operational amplifier.

Ex :- A switch is connected to P0.0. write an ALP to do the following

(1) when SW=0, DAC output gives a staircase waveform.
(2) when SW=1, DAC output gives triangular waveform.

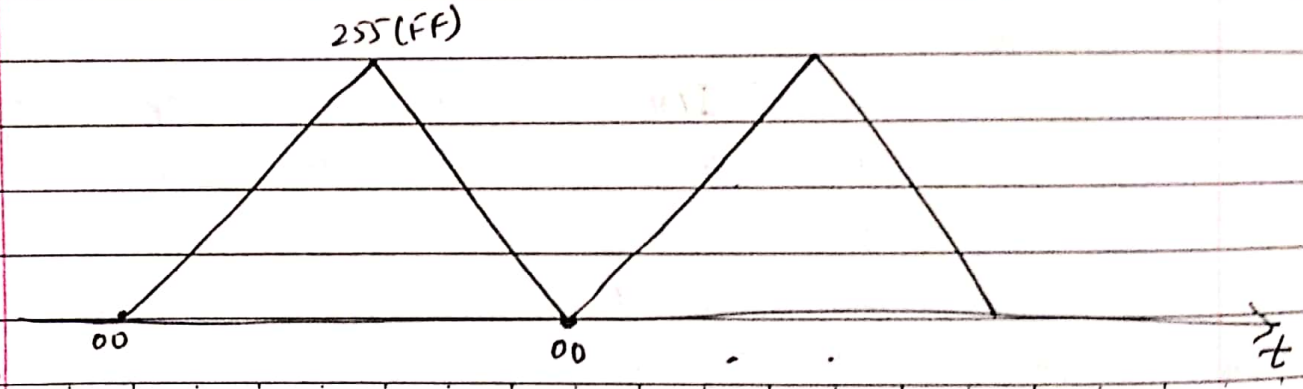
Sol :- For example, staircase waveform has 5 steps. The number of increment between each step = 255/5 = 51.



Output waveform

```
SW    EQU  P0.0
      org  00H
      setb SW        ; configure P0.0 as input pin.
CHECK MOV  C, SW
      JC   TRIANG    ; if SW=1, jump to TRIANG.
      MOV  A, #00    ; generate staircase wave
      MOV  P1, A
      ACALL DELAY
RPT:  ADD  A, #51
      MOV  P1, A
      ACALL DELAY
      CJNE A, #255, RPT  ; Repeat till A=255.
      SJMP CHECK

TRING: MOV A, #00
INCR:  MOV P1, A
       INC A
       CJNE A, #255, INCR
DECR:  MOV P1, A
       DEC A
       CJNE A, #00, DECR
       SJMP CHECK
```

```
DELAY:
RPT1:  MOV R1, #20
RPT2:  MOV R2, # 200
RPT3:  DJNZ R2, RPT3
       DJNZ R1, RPT2
       DJNZ R0, RPT1
       RET
       END
```



Triangular Output

Generating a sine wave :-

The sine waves amplitude varies from −1 to +1 when angle of sine wave varies from 0° to 360°. The 8051 can send only binary values to DAC to generate analog waveform. Hence magnitude of sine wave is represented by integer values for each angle.

Assuming DAC is applied with a full scale voltage of 10V, the output voltage is maximum when all the data inputs are high. To achieve this, output voltage should be equal to:

$$V_{out} = 5V + (5 \times \sin\theta)$$

Angles Vs Voltage magnitude for sine wave

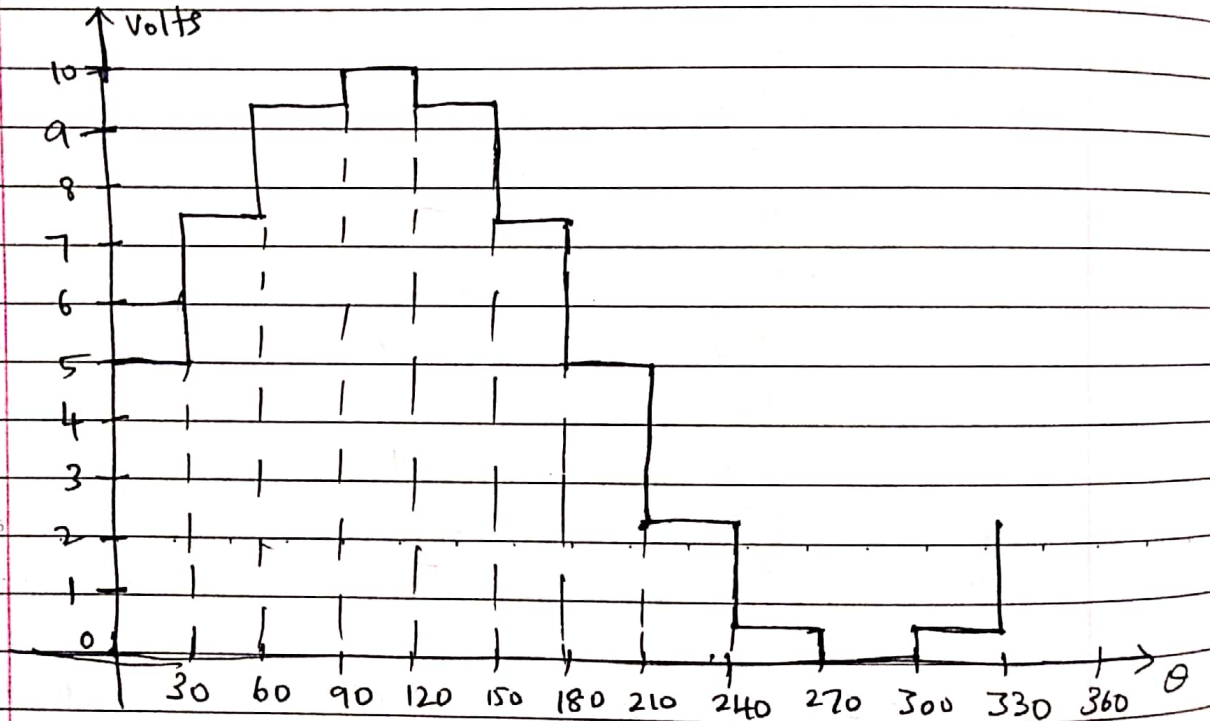
Angle (θ)	Sin θ	$V_{out} = 5V + (5V \times \sin\theta)$	Value sent to DAC (decimal) (Voltage × 25.6)
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	−0.5	2.5	64
240	−0.866	0.669	17
270	−1.0	0	0
300	−0.866	0.669	17
330	−0.5	2.5	64
360	0	5	128

The 8-bit DAC has $2^8 = 256$ steps at the output. For a full scale output of 10V, each voltage has a step size equal to $(256/10) = 25.6$. which is the resolution of DAC.

Write a C program to generate a sine wave.

```
#include <reg51.h>
sfr DATA = P1;
void main()
{ unsigned char VALUE [12] = { 128, 192, 238, 255, 238, 192,
                               128, 64, 17, 0, 17, 64};
  unsigned char x;
  while (1)
```

```
} for (x=0; x<12; x++)
    { DATA = VALUE[0];
    }
}
```



Stepper Motor Interfacing

A stepper motor translates electrical pulses into mechanical movement. In disk drives, dot matrix printers, robotics the stepper motor is used for position control.

They have permanent magnet Rotor (shaft) surrounded by stator. There are reluctance stepper motor that do not have permanent magnet rotor.

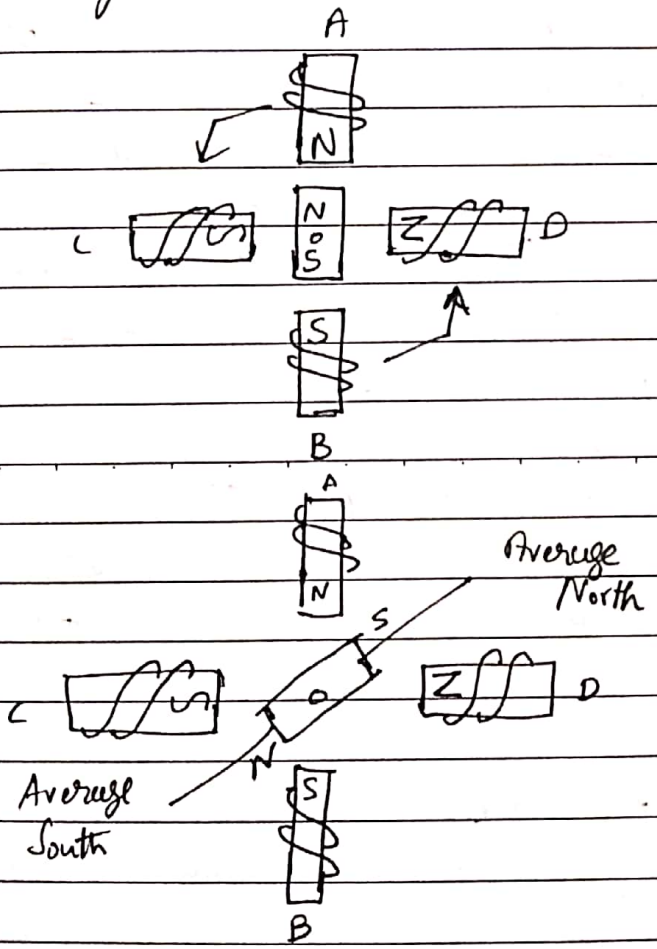
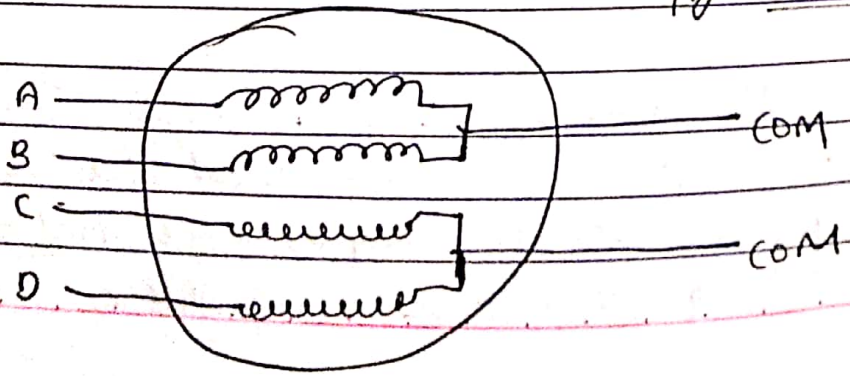


Fig1. Rotor Alignment

Fig2: Stator Windings Configuration



The most common S.M. have four stator windings that are paired with a center-tapped common as in fig 2. This is known as four-phase or unipolar S.M. The centre tap allows a change of current direction in each of two coils. when a winding is grounded, resulting in polarity change of stator.

The S.M. shaft moves in a fixed repeatable increment to move it to a precise position. This is because poles of same polarity repel and opposite poles attract. The direction of rotation is dictated by stator poles. The stator poles are determined by current sent through wire coils. As the direction of current changes, polarity also changes causing reverse motion of rotor.

The S.M. has 6 leads, 4 represents four stator windings and 2 commons for centre-tapped leads. As sequence of power is applied to each stator, rotor will rotate.

Normal 4-step Sequence

Clockwise	Step No.	Winding A	Winding B	Winding C	Winding D	Counter clockwise
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	

Above table shows 2-phase, 4-step stepping sequence. We can start from any sequence but must continue in a particular order. For example, if we start with step 3 (0110) next sequence must be 4, 1, 2, etc.,

Step angle :-

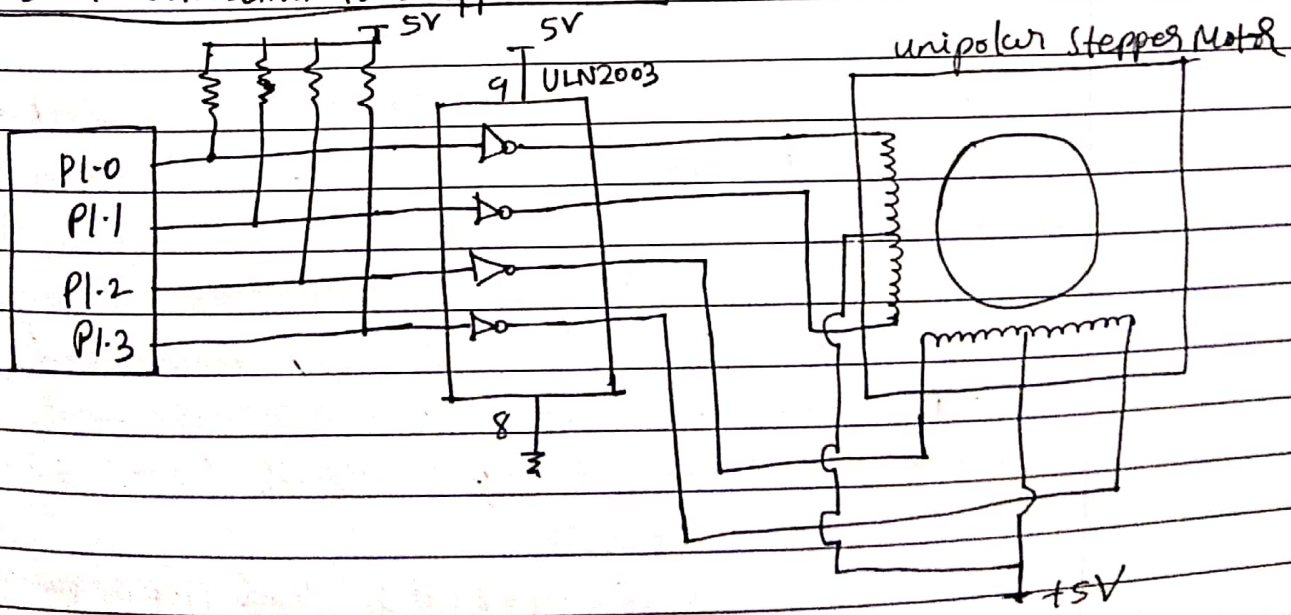
The step angle depends on the internal construction of motor, that is, number of teeth on stator and rotor. It is minimum degree of rotation associated with a single step.

Stepper Motor Step Angles

Step Angle	Steps per Revolution
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24.

Steps per revolution is the total number of steps needed to rotate one complete rotation or 360°(200X1.8°).

8051 Connection to Stepper Motor



The four leads of stator winding are controlled by four bits of 8051 port. Since 8051 lacks sufficient current to drive the stepper motor windings, use driver such as ULN2003 to energize the stator.

Steps per second and rpm relation :-

$$\text{Steps per second} = \frac{\text{rpm} \times \text{Steps per revolution}}{60}$$

The four-step sequence and number of teeth on rotor

In 4-step switching sequence, after 4-steps the same two windings will be 'ON'. After completing 4 steps, the rotor moves only one tooth pitch. Therefore with 200 steps per revolution, the rotor has 50 teeth since $4 \times 50 = 200$ steps. are needed to complete one revolution. Hence minimum step angle is always a function of number of teeth on rotor. Smaller the step angle, the more teeth the rotor passes.

Q. Write a program to rotate a motor 64° in clockwise direction. The motor has step angle of 2°. Use 4-step sequence.

Sol Step angle = 2°, steps per revolution = 180, No. of rotor teeth = 45 (4 X 45), movement per 4-step sequence = 8° (2 X 4). To move the rotor 64°, send eight consecutive 4-step sequences, $8 \times 4 = 32$ steps.

```
        ORG 00
        MOV A,#66H ; load step sequence
        MOV R0,#32
back:   RR A       ; rotate right clockwise
```

```
MOV P1,A   ; issue sequence to motor
ACALL Delay
DJNZ R0, back
~~End~~

Delay: MOV R2, #100
l1:    MOV R3, #255
l2:    DJNZ R3, l2
       DJNZ R2, l1
       Ret
       end
```

Half-Step 8-step Sequence

Clockwise Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
1	1	0	0	1	
2	1	0	0	0	
3	1	1	0	0	
4	0	1	0	0	
5	0	1	1	0	
6	0	0	1	0	
7	0	0	1	1	
8	0	0	0	1	

Wave-Drive 4-step Sequence:-

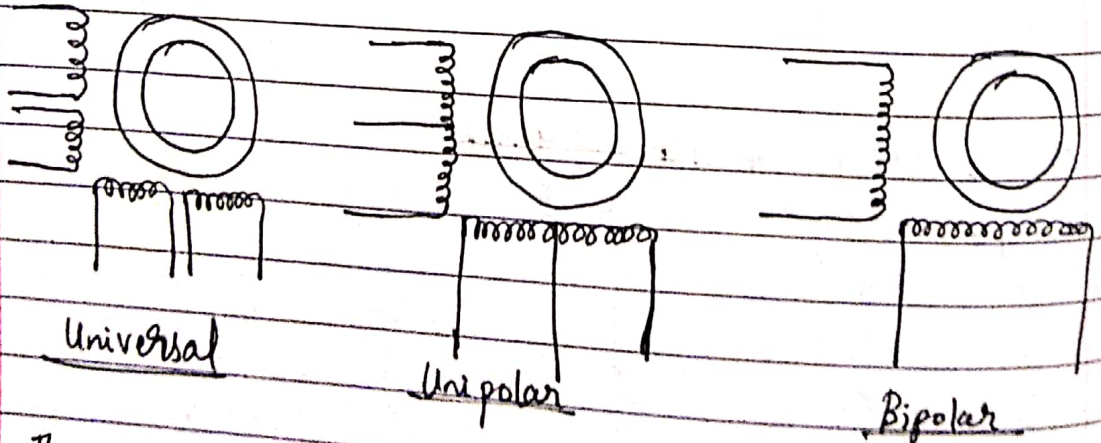
Clockwise Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
1	1	0	0	0	
2	0	1	0	0	
3	0	0	1	0	
4	0	0	0	1	

Motor Speed:-
Measured in steps per second, is a function of the switching rate. In delay subroutine by changing the length of time delay loop, various rotation speeds can be achieved.

Holding torque:-
Definition:- With motor shaft at standstill or zero rpm condition, the amount of torque from an external source, required to break away the shaft from its holding position. This is measured with rated voltage and current applied to the motor. The unit is ounce-inch or kg-cm.

Wave Drive 4-step Sequence:-
This type of sequence is in addition to 8-step sequence and 4-step sequence. In fact, 8-step sequence is a combination of 4-step normal sequence and 4-step wave Drive sequence.

Common stepper Motor types:-

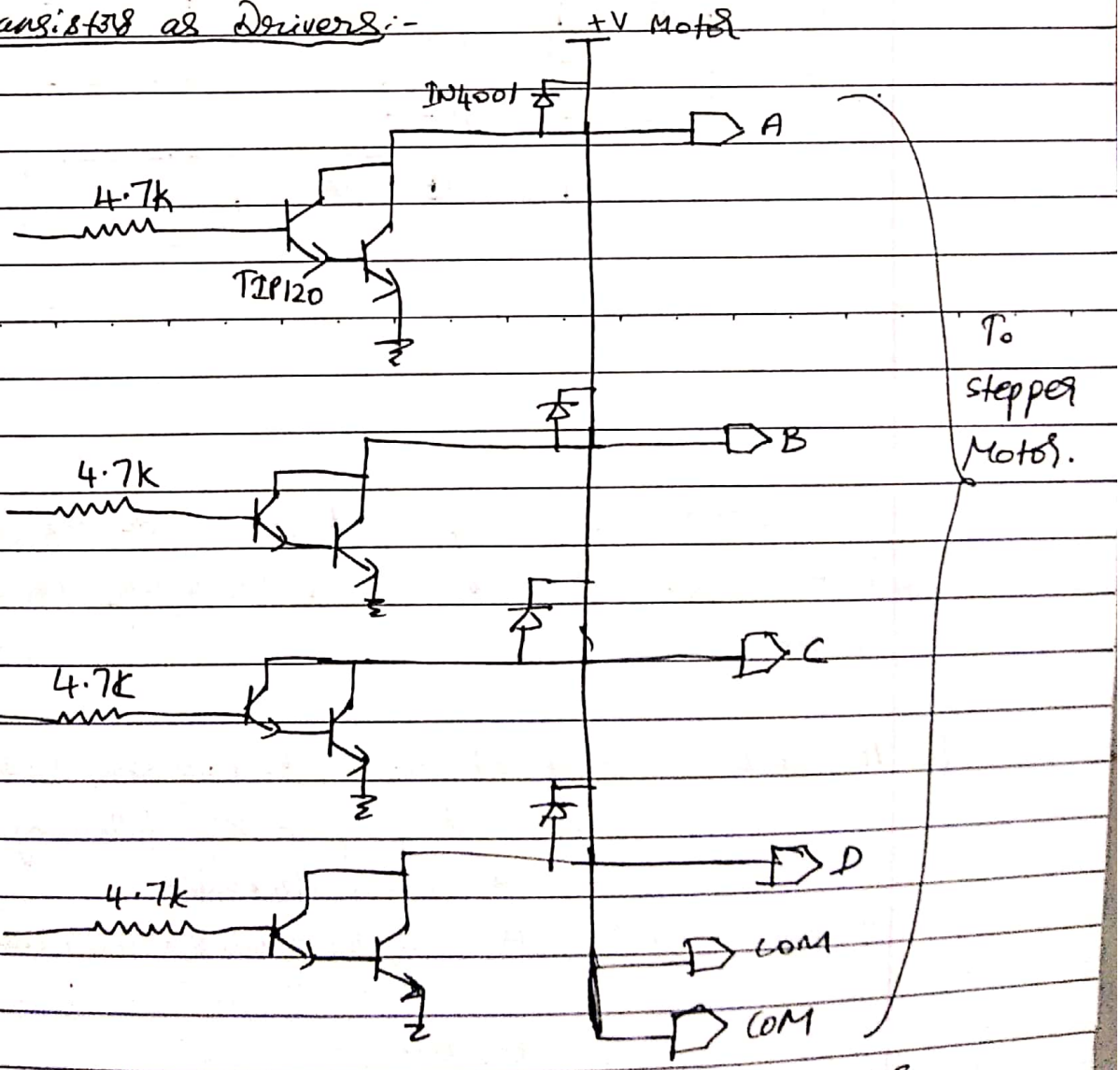


Three types of stepper motor: universal, unipolar and Bipolar. They are identified by number of connections to the motor.

A universal type has eight, unipolar has six and bipolar has four connections. The universal type can be configured for all three modes while unipolar can be either unipolar or bipolar. Bipolar cannot be configured for universal or unipolar.

Unipolar type can be controlled using transistors. Bipolar type requires H-Bridge circuitry. It requires higher operational current than unipolar but can have higher holding torque.

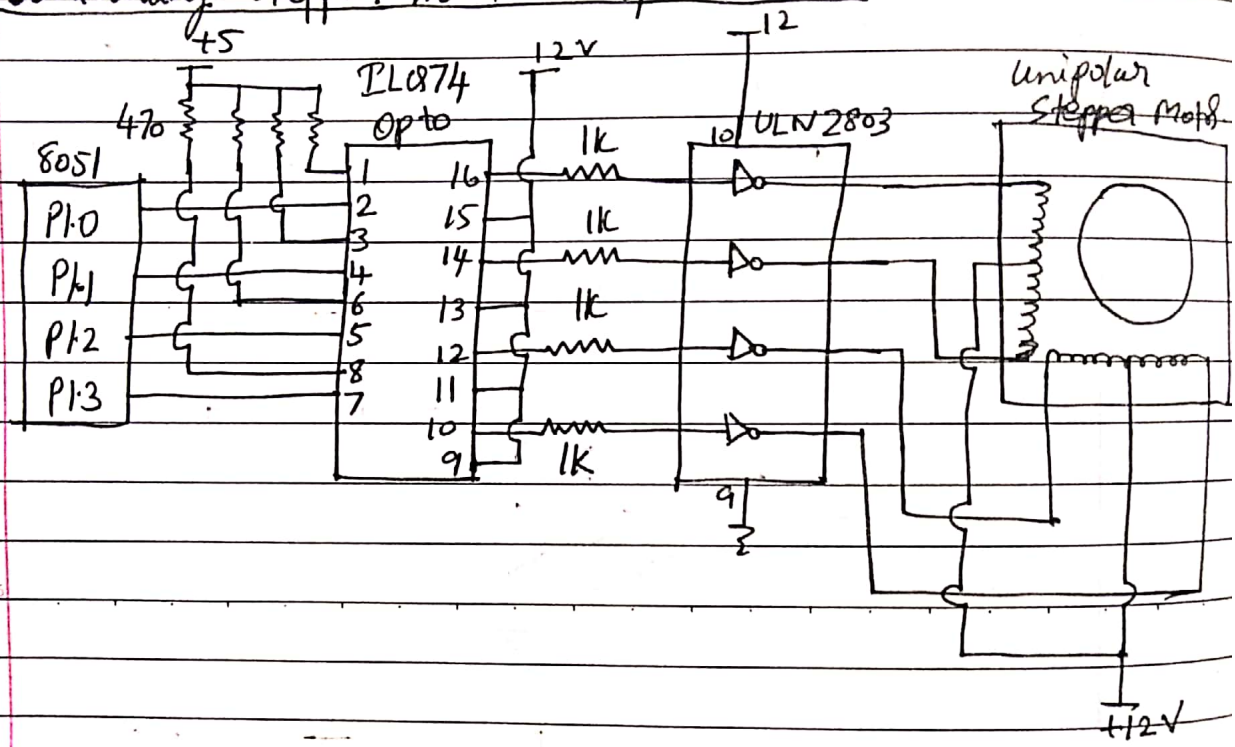
Using Transistors as Drivers:-



Interface to unipolar Steppers Motor using Transistors

Diodes are used to reduce back EMF spike created when coils are energized and de-energized. TIP transistors can be used to supply higher current to the motor.

Controlling stepper motor via optoisolator:-



Optoisolators are widely used to isolate the stepper motor's EMF voltage and keep it from damaging the digital/ microcontroller system.

Ex:- A switch is connected to pin P2.7. Write an ALP to monitor the status of SW and perform the following:

(a) If SW=0, the S.M. moves clockwise
(b) If SW=1, the S.M. moves counter clockwise.

```
        ORG 00H
Main:   setb P2.7      ; configure P2.7 as i/p port
        mov a, #66h    ; Starting phase value
        mov P1, A      ; send value to P1.
```

```
turn: jnb   P2.7, CW   ; check switch result
      rr    a          ; Rotate right
      acall delay
      mov   P1, a      ; send value to P1
      sjmp  turn       ; repeat
CW:   rl    a          ; rotate left
      acall delay
      mov   P1, a      ; send value to P1.
      sjmp  turn       ; repeat
delay: mov  R2, #100
h1:   mov   r3, #255
h2:   djnz  r3, h2
      djnz  r2, h1
      ret
      end
```

Ex:- Rotate S.M. continuously ① clockwise using wave drive 4-step sequence, ② clockwise using the half-step 8-step sequence.

Sol ① The sequence values are saved in ROM locations starting from 0100H.

```
      org   00
start: mov  r0, #04
      mov   dptr, #0100h
rpt:  clr   a
      movc  a, @a+dptr
      mov   P1, a
      acall delay
      inc   dptr
      djnz  r0, rpt
      sjmp  start
```

```
org 0100h
db 8,4,2,1   ; db -> define byte
end
```

(2) Sequence values are save in ROM locations starting from 0200h.

```
        org 00
start : mov r0, #08
        mov dptr, #0200h
rpt:    clr a
        movc a, @a+dptr
        mov P1, a
        acall delay
        inc dptr
        djnz r0, rpt
        sjmp start
        org 0200h
dB:     09,08, 0Ch, 04, 06, 02, 03, 01
        end
```

Ex:- Write a C program to monitor the status of switch connected to P2.7 and perform the following.
(a) If SW=0, the S.M. moves clockwise
(b) If SW=1, the S.M. moves counterclockwise.

Sol

```
#include <reg51.h>
sbit sw = P2^7; // definite single bit of Port2
void main()
{ sw=1;
 while(1)
 { if (sw==0)
```

```
{  P1 = 0X66;          // 66h = 0110 0110 b
   msdelay(100);
   P1 = 0xCC;          // CCh = 1100 1100 b
   msdelay(100);
   P1 = 0X99;          // 99h = 1001 1001 b
   msdelay(100);
   P1 = 0X33;          // 33h = 0011 0011 b
   msdelay(100); }
else {
   { P1 = 0X66;
     msdelay(100);
     P1 = 0X33;
     ms delay(100);
     P1 = 0X99;
     msdelay(100);
     P1 = 0XCC;
     msdelay(100);
     na }
     }
}

void msdelay(unsigned int value)
{ unsigned int x, y;
  for(x=0; x<1275; x++)
   for(y=0; y<value; y++);
}
```

DC Motor Interfacing and PWM

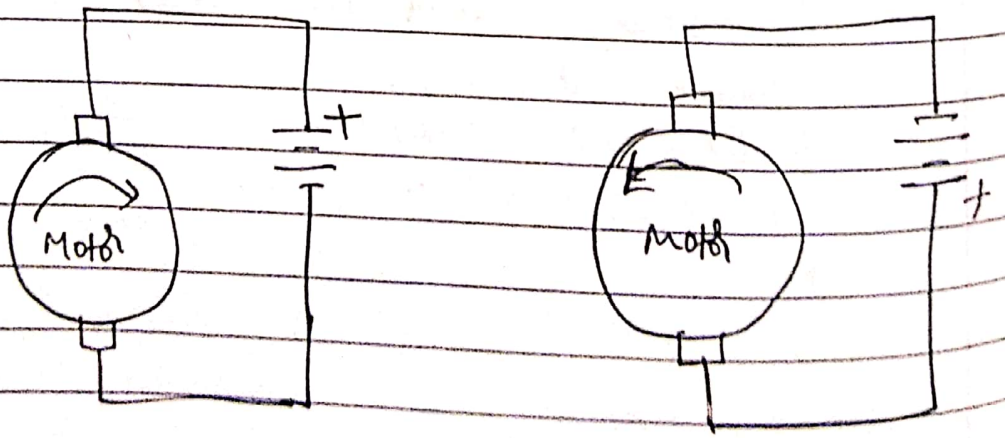
DC Motors:-

A direct current motor is a device that translates electrical pulses into mechanical movement. It has '+' and '-' leads. Connecting them to a DC voltage source moves the motor in one direction. For example CPU fan is a DC Motor.

Once powered, the DC motor moves continuously. The maximum speed is measured in rpm (rotations per minute). and will be specified in the data sheet. It has two RPM, no load and loaded. The manufacturer's data sheet gives no-load rpm. It can varry from few thousand to tens of thousands. The speed depends on load, more the load, lesser the speed.

The voltage range of DC motor can be from 1 to 150V. As the voltage increases, speed also increases. The current rating is the current consumption when nominal voltage is applied with no load and can be from 25 mA to few amperes. If it is overloaded, it will stall and can damage the motor due to heat generated and high current consumption.

Unidirectional Control:-

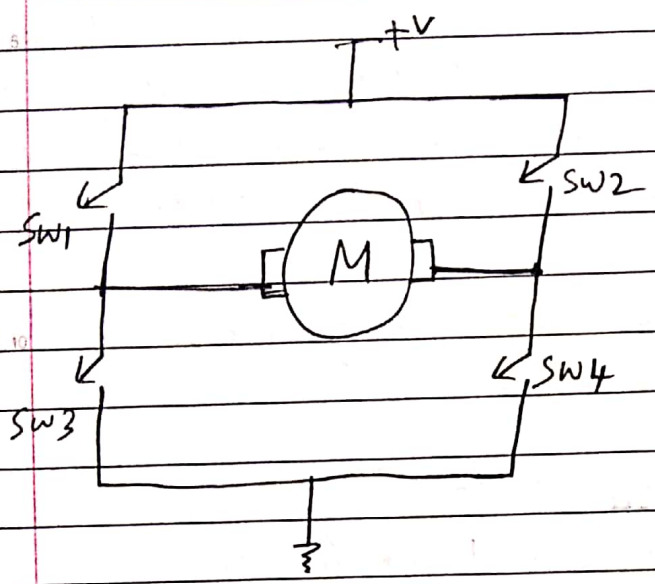


Clockwise rotation

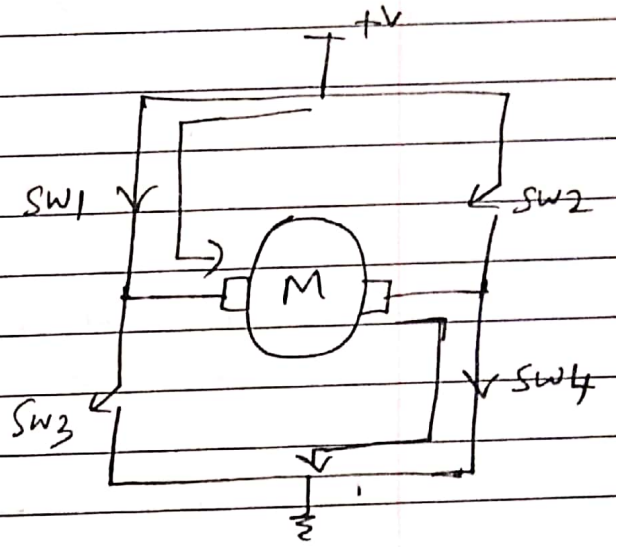
Counter clockwise rotation

The direction of rotation of motor depends on the polarity of supply applied to the motor.

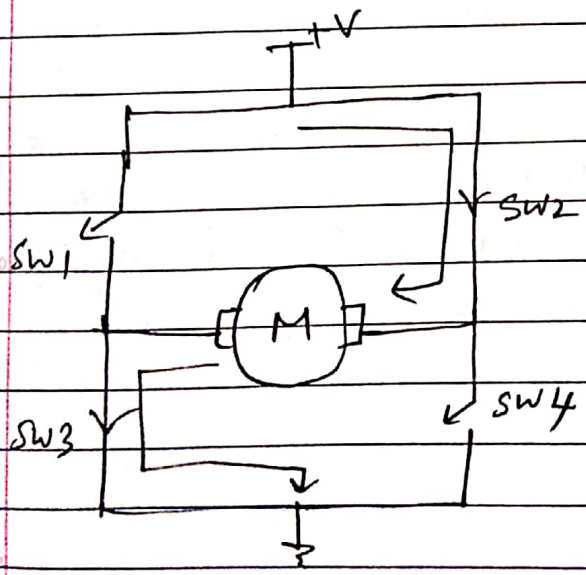
Bidirectional control :- (H-Bridge Motor Configuration)



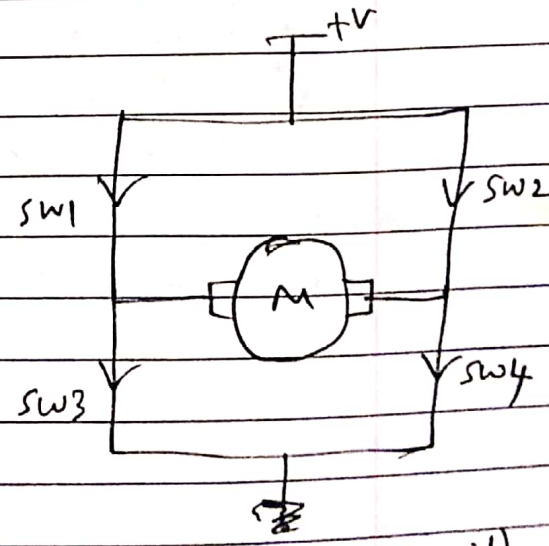
(a) Motor in OFF condition.



(b) Clockwise rotation



(c) Counter clockwise rotation



(d) Invalid state (short circuit)

With the help of relays or specially designed chips we can change the direction of DC motor rotation. Above figure sho four possible H-Bridge Configuration.

In fig ⓐ, when all the switches are open, motor will be OFF condition. In fig ⓑ, when switches 1 and 4 are closed motor rotates in clockwise direction. In fig ⓒ, when switches 2 and 3 are closed, motor rotates in counter clockwise direction. In fig ⓓ, when all the switches are closed, current flows directly to ground. which is the case when switches 1 and 3 are closed or switches 2 and 4 are closed.

H-Bridge control can be created using relays, transistors, or single IC such as L293.

Pulse Width Modulation (PWM):-

The speed of motor depends on three factors
ⓐ Load, ⓑ Voltage, and ⓒ current.
For a given fixed load we can maintain a steady speed by using a method called PWM. By changing (modulating) the width of pulse, we can increase or decrease the amount of power applied there by varrying the speed. The voltage has fixed amplitude but variable duty cycle. Wider the pulse higher is the speed of motor.