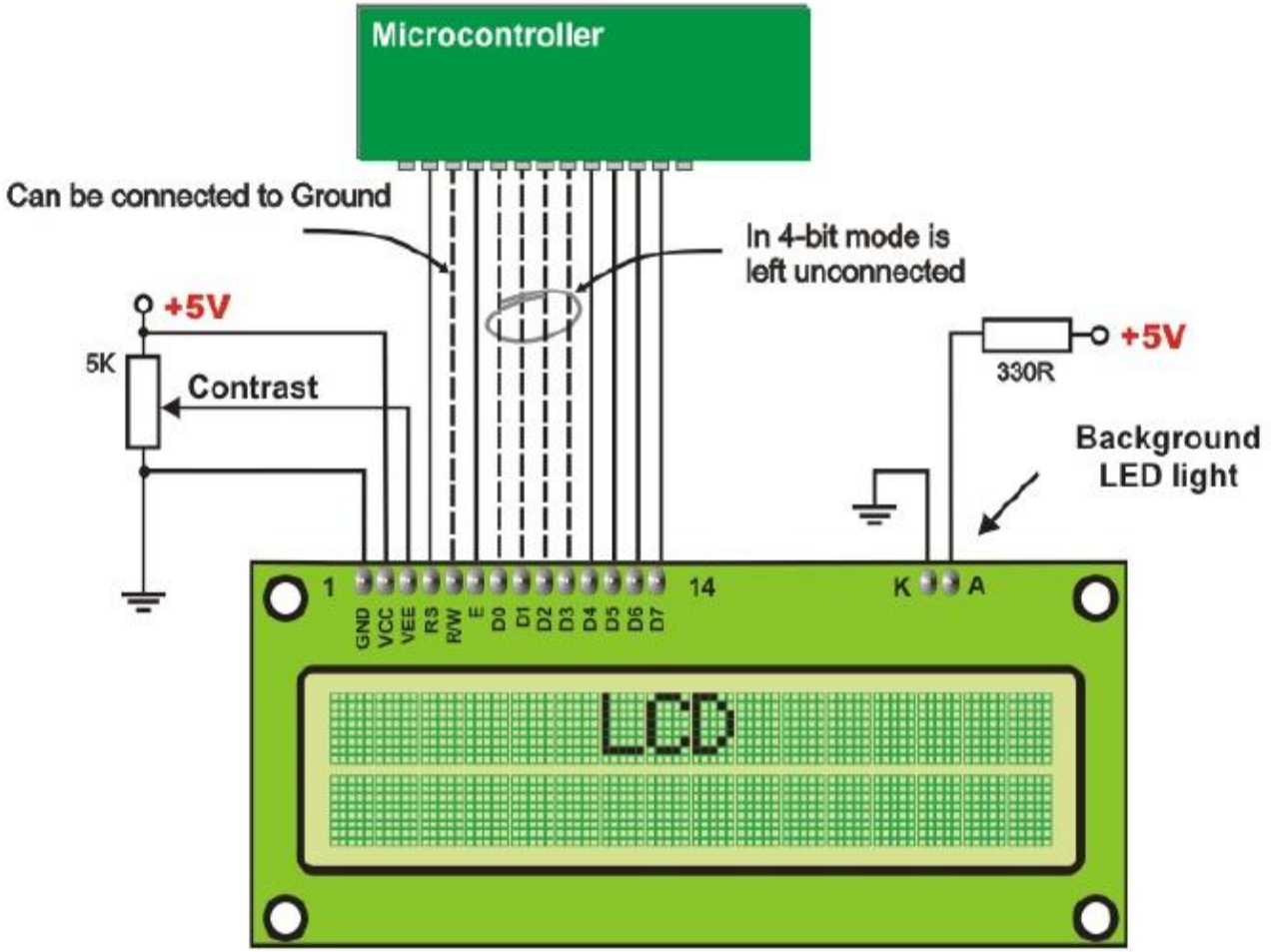
# UNIT IV: INTERFACING OF 8051 MICROCONTROLLER:

## LCD,
## STEPPER MOTOR, ADC
## AND DAC

# WHAT IS LCD?

- An LCD (Liquid crystal display) display is specifically manufactured to be used with microcontrollers, which means that it cannot be activated by standard IC circuits.
- It is used for displaying different messages on a miniature liquid crysal display.
- It displays all the letters of alphabet, Greek letters, punctuation marks, mathematical symbols etc. In addition, it is possible to display symbols made up by the user.
- Other useful features include automatic message shift (left and right), cursor appearance, LED backlight etc.

# LCD INTERFACING WITH 8051



# LCD PIN DESCRIPTION

**Pin Descriptions for LCD**

| Pin | Symbol | I/O | Descriptions |
|---|---|---|---|
| 1 | VSS | -- | Ground |
| 2 | VCC | -- | +5V power supply |
| 3 | VEE | -- | Power supply to control contrast |
| 4 | RS | I | RS=0 to select command register,<br>RS=1 to select data register |
| 5 | R/W | I | R/W=0 for write,<br>R/W=1 for read |
| 6 | E | I/O | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

used by the LCD to latch information presented to its data bus

- Send displayed information or instruction command codes to the LCD
- Read the contents of the LCD's internal registers

# LCD COMMANDS

**LCD Command Codes**

| Code (Hex) | Command to LCD Instruction Register |
|---|---|
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to 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 | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning to 1st line |
| C0 | Force cursor to beginning to 2nd line |
| 38 | 2 lines and 5x7 matrix |

# LCD INTERFACING C CODE

**Example 12-2**

Write an 8051 C program to send letters 'M', 'D', and 'E' to the LCD using the busy flag method.

**Solution:**

```
#include <reg51.h>
sfr ldata = 0x90; //P1=LCD data pins
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);    //line 1, position 6
   lcdcmd('M');
   lcdcmd('D');
   lcdcmd('E');
}
.....
```

# C CODE CONTD

```
void lcdcmd(unsigned char value){
  lcdready();          //check the LCD busy flag
  ldata = value;       //put the value on the pins
  rs = 0;
  rw = 0;
  en = 1;              //strobe the enable pin
  MSDelay(1);
  en = 0;
  return;
}

void lcddata(unsigned char value){
  lcdready();          //check the LCD busy flag
  ldata = value;       //put the value on the pins
  rs = 1;
  rw = 0;
  en = 1;              //strobe the enable pin
  MSDelay(1);
  en = 0;
  return;
}
.....
```

# C CODECONTD

```
void lcdready(){
  busy = 1;          //make the busy pin at input
  rs = 0;
  rw = 1;
  while(busy==1){    //wait here for busy flag
    en = 0;          //strobe the enable pin
    MSDelay(1);
    en = 1;
}

void lcddata(unsigned int itime){
  unsigned int i, j;
  for(i=0;i<itime;i++)
    for(j=0;j<1275;j++);
}
```

# LCD ASSEMBLY CODE

To send any of the commands to the LCD, make pin RS=0. For data, make RS=1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. This is shown in the code below.

```
;calls a time delay before sending next data/command
;P1.0-P1.7 are connected to LCD data pins D0-D7
;P2.0 is connected to RS pin of LCD
;P2.1 is connected to R/W pin of LCD
;P2.2 is connected to E pin of LCD
        ORG   0H
        MOV   A,#38H   ;INIT. LCD 2 LINES, 5X7 MATRIX
        ACALL COMNWRT  ;call command subroutine
        ACALL DELAY    ;give LCD some time
        MOV   A,#0EH   ;display on, cursor on
        ACALL COMNWRT  ;call command subroutine
        ACALL DELAY    ;give LCD some time
        MOV   A,#01    ;clear LCD
        ACALL COMNWRT  ;call command subroutine
        ACALL DELAY    ;give LCD some time
        MOV   A,#06H   ;shift cursor right
        ACALL COMNWRT  ;call command subroutine
        ACALL DELAY    ;give LCD some time
        MOV   A,#84H   ;cursor at line 1, pos. 4
        ACALL COMNWRT  ;call command subroutine
        ACALL DELAY    ;give LCD some time
```

```
        MOV   A,#'N'     ;display letter N
        ACALL DATAWRT    ;call display subroutine
        ACALL DELAY      ;give LCD some time
        MOV   A,#'O'      ;display letter O
        ACALL DATAWRT    ;call display subroutine
AGAIN:  SJMP  AGAIN      ;stay here
COMNWRT:                 ;send command to LCD
        MOV   P1,A       ;copy reg A to port 1
        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      ;give LCD some time
        CLR   P2.2       ;E=0 for H-to-L pulse
        RET
DATAWRT:                 ;write data to LCD
        MOV   P1,A       ;copy reg A to port 1
        SETB  P2.0       ;RS=1 for data
        CLR   P2.1       ;R/W=0 for write
        SETB  P2.2       ;E=1 for high pulse
        ACALL DELAY      ;give LCD some time
        CLR   P2.2       ;E=0 for H-to-L pulse
        RET
DELAY:  MOV   R3,#50     ;50 or higher for fast CPUs
HERE2:  MOV   R4,#255    ;R4 = 255
HERE:   DJNZ  R4,HERE    ;stay until R4 becomes 0
        DJNZ  R3,HERE2
        RET
        END
```

# ASSEMBLY CODE contd

```
;Check busy flag before sending data, command to LCD
;p1=data pin
;P2.0 connected to RS pin
;P2.1 connected to R/W pin
;P2.2 connected to E pin
      ORG   0H
      MOV   A,#38H        ;init. LCD 2 lines ,5x7 matrix
      ACALL COMMAND       ;issue command
      MOV   A,#0EH        ;LCD on, cursor on
      ACALL COMMAND       ;issue command
      MOV   A,#01H        ;clear LCD command
      ACALL COMMAND       ;issue command
      MOV   A,#06H        ;shift cursor right
      ACALL COMMAND       ;issue command
      MOV   A,#86H        ;cursor: line 1, pos. 6
      ACALL COMMAND       ;command subroutine
      MOV   A,#'N'        ;display letter N
      ACALL DATA_DISPLAY
      MOV   A,#'O'        ;display letter O
      ACALL DATA_DISPLAY
HERE:SJMP   HERE          ;STAY HERE
```

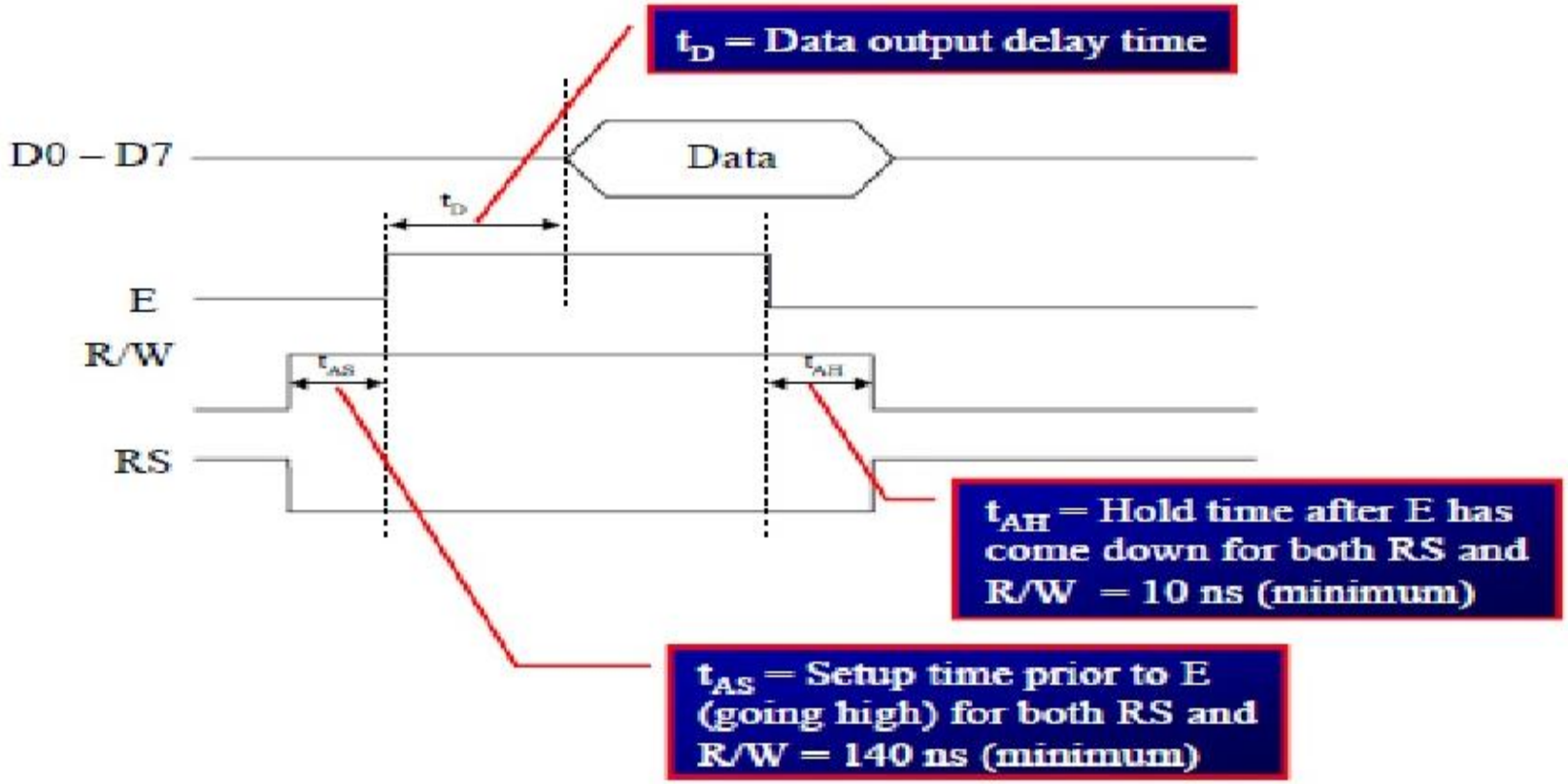
```
COMMAND:
      ACALL READY          ;is LCD ready?
      MOV   P1,A           ;issue command code
      CLR   P2.0           ;RS=0 for command
      CLR   P2.1           ;R/W=0 to write to LCD
      SETB  P2.2           ;E=1 for H-to-L pulse
      CLR   P2.2           ;E=0,latch in
      RET
DATA_DISPLAY:
      ACALL READY          ;is 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
      CLR   P2.2           ;E=0 latch in
      RET
READY:
      SETB  P1.7           ;make P1.7 input port
      CLR   P2.0           ;RS=0 access command reg
      SETB  P2.1           ;R/W=1 read command reg
;read command reg and check busy flag
BACK: SETB  P2.2           ;E=1 for H-to-L pulse
      CLR   P2.2           ;E=0 H-to-L pulse
      JB    P1.7,BACK      ;stay until busy flag=0
      RET
      END
```

To read the command register, we make R/W=1, RS=0, and a H-to-L pulse for the E pin.

If bit 7 (busy flag) is high, the LCD is busy and no information should be issued to it.

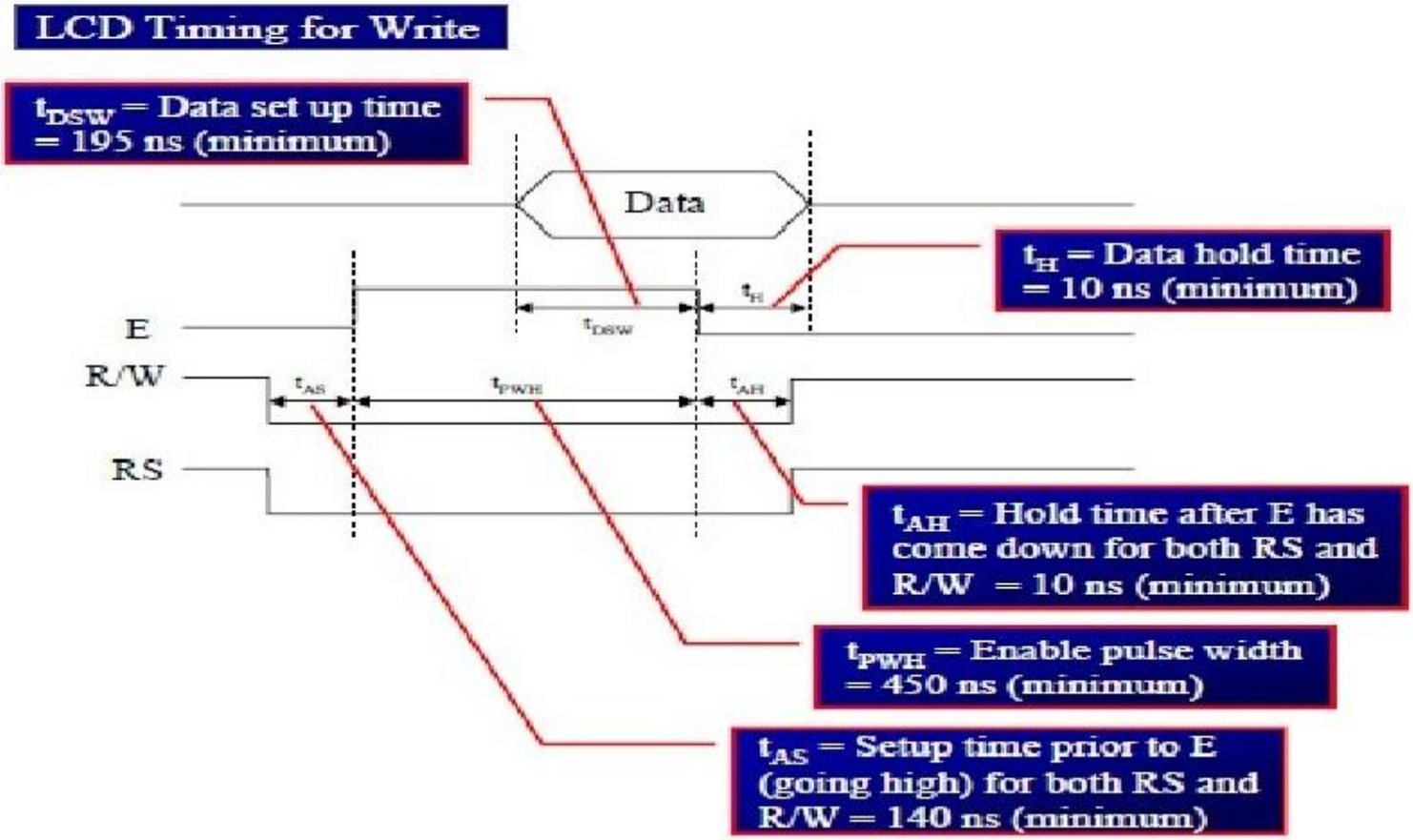
# LCD READ OPERATION

**LCD Timing for Read**




**Note : Read requires an L-to-H pulse for the E pin**

# LCD WRITE OPERATION



# STEPPER MOTOR

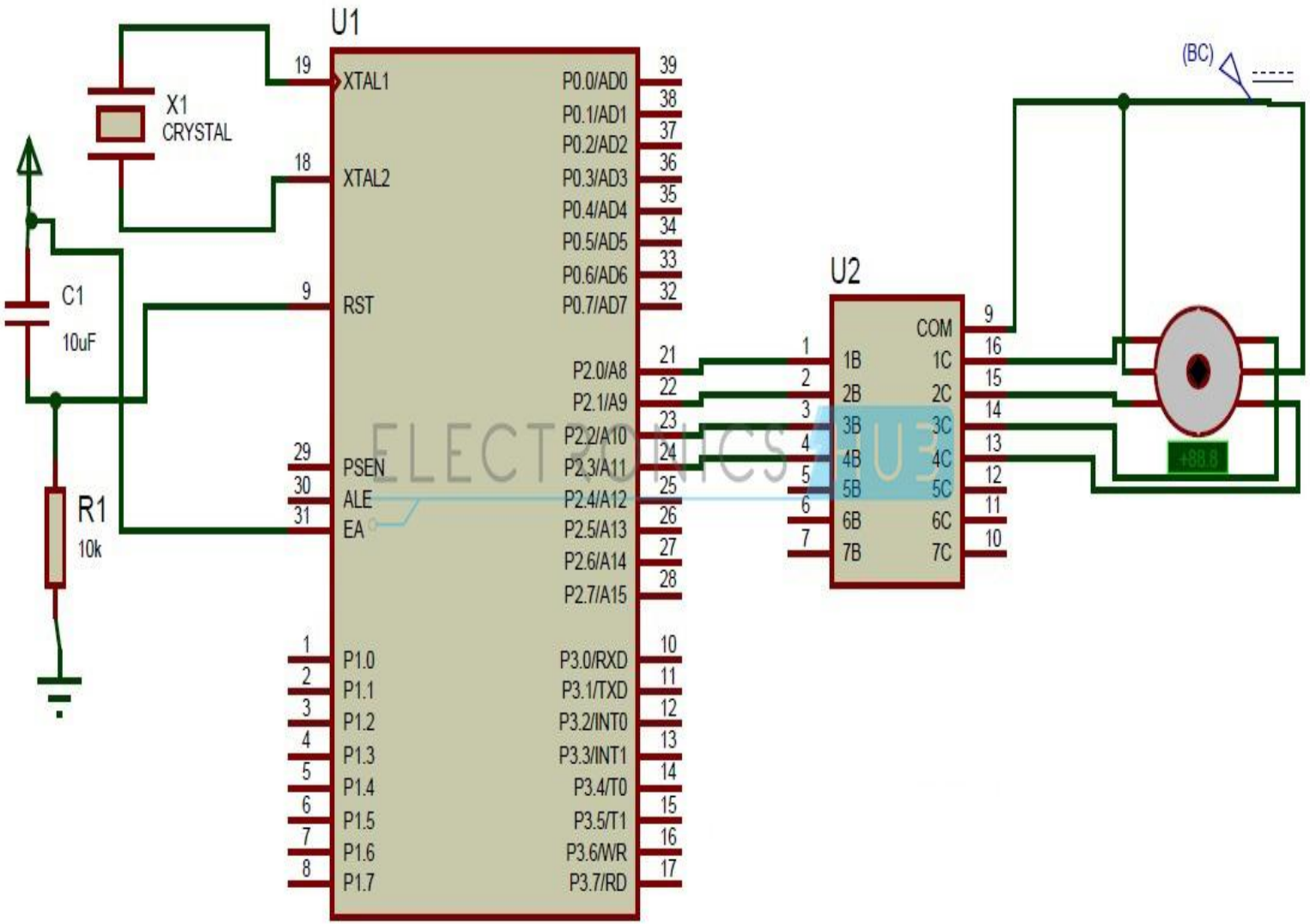
- A *stepper motor* is an electromechanical device which converts electrical pulses into discrete mechanical movements or steps.

- This motor divides a full rotation of 360 degrees into a number of equal steps.

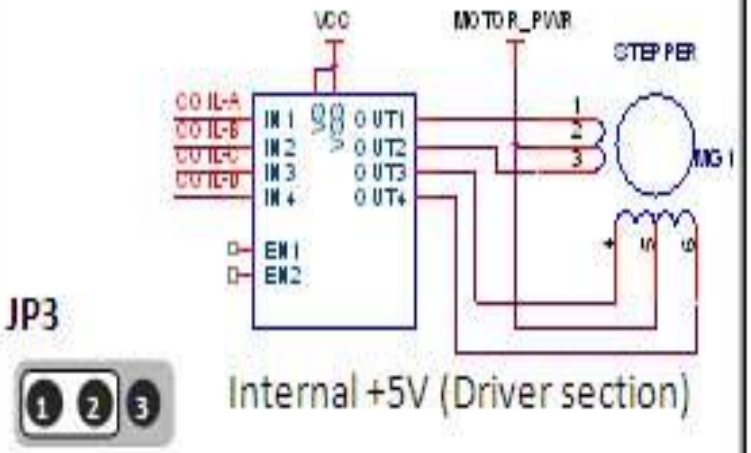
# INTERFACING STEPPER MOTOR WITH 8051

- We now want to control a stepper motor in 8051 . It works by turning ON & OFF a four I/O port lines generating at a particular frequency.
- The 8051 has four numbers of I/O port lines, connected with I/O Port lines (P0.0 – P0.3) to rotate the stepper motor.
- ULN2003 is a high voltage and high current Darlington array IC.
- ULN2003 is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.

# INTERFACING DIAGRAM

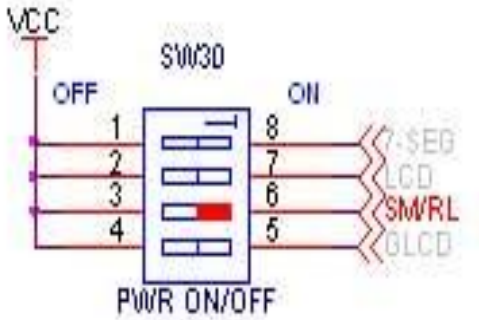


# Pin Assignment with 8051

| | Stepper Motor(5V) | 8051 Lines | Stepper Motor PWR Select |
|---|---|---|---|
| STEPPER MOTOR | COIL-A | P0.0 |  |
| | COIL-B | P0.1 | |
| | COIL-C | P0.2 | |
| | COIL-D | P0.3 | |



Make switch **SW30** to ***SM/RL*** label marking position.



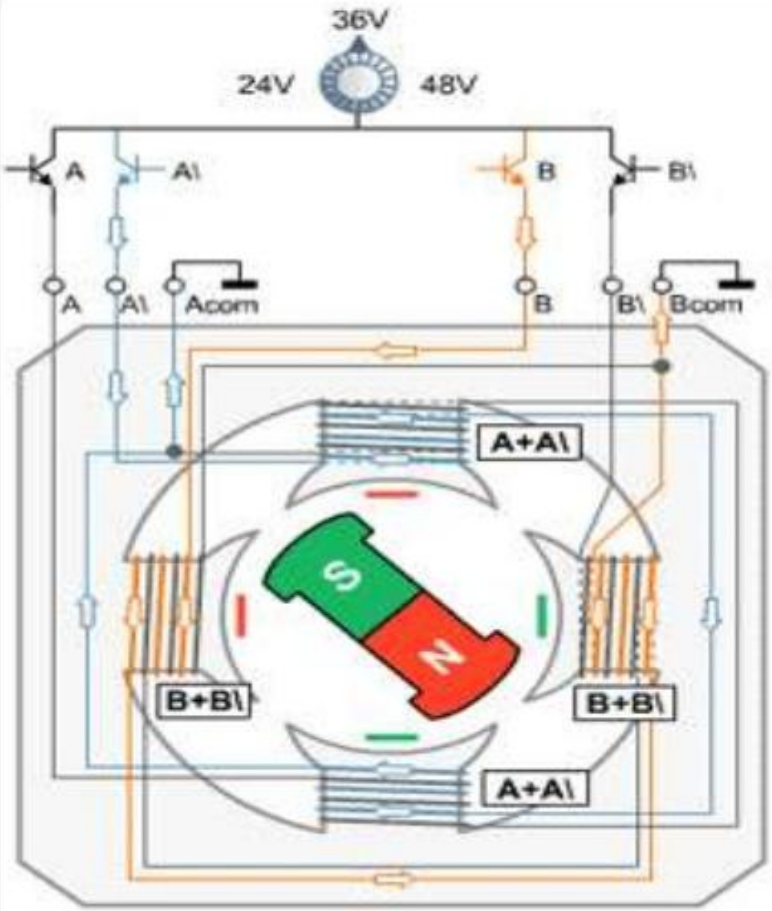
# Full step and half step sequence

- **In the full step sequence**, two coils are energized at the same time and motor shaft rotates. The order in which coils has to be energized is given in the table below.
- **In Half mode step sequence**, motor step angle reduces to half the angle in full mode. So the angualar resolution is also increased i.e. it becomes double the angular resolution in full mode. Also in half mode sequence the number of steps gets doubled as that of full mode. Half mode is usually preffered over full mode. Table below shows the pattern of energizing the coils.

# Full step sequence

| Full Mode Sequence | | | | |
|---|---|---|---|---|
| Step | A | B | A\ | B\ |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 1 | 0 | 0 | 1 |

# Full step sequence




**Lead Unipolar Driver**

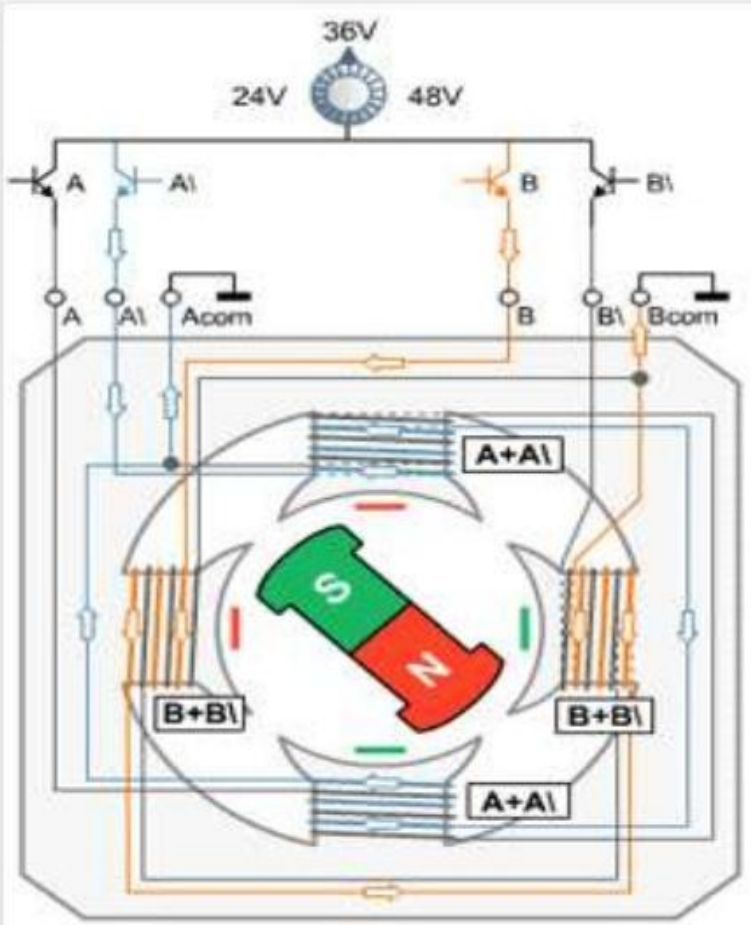
Unipolar control is the most simple and cost-effective way to drive a stepper motor, but results in approximately 30% less torque in comparison to the nowadays widely used bipolar drivers. Since the cost advantage is very small today due to cheap integrated circuits, bipolar drivers are now used in most new applications.

| Stepmode | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F | 0 | | 1 | | 2 | | 3 | |
| H | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| B | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| A\ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| B\ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| dez | 12 | 4 | 6 | 2 | 3 | 1 | 9 | 8 |

# Half step sequence

| Half Mode Sequence | | | | |
|---|---|---|---|---|
| Step | A | B | A\ | B\ |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 | 0 |

# Half step sequence




**Lead Unipolar Driver**

Unipolar control is the most simple and cost-effective way to drive a stepper motor, but results in approximately 30% less torque in comparison to the nowadays widely used bipolar drivers. Since the cost advantage is very small today due to cheap integrated circuits, bipolar drivers are now used in most new applications.

| Stepmode | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F | 0 | | 1 | | 2 | | 3 | |
| H | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| B | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| A\ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| B\ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| dez | 12 | 4 | 6 | 2 | 3 | 1 | 9 | 8 |

# C Program to control stepper motor using 8051

```
#include <reg51.h>                    //Define 8051 registers
#include<stdio.h>
void DelayMs(unsigned int);           //Delay function\

//--------------------------------

//    Main Program

//--------------------------------
void Clockwise (void)
{
      unsigned int i;
      for (i=0;i<30;i++)
      {
            P0 = 0x01;DelayMs(5);          //Delay 20msec
            P0 = 0x02;DelayMs(5);
            P0 = 0x04;DelayMs(5);
            P0 = 0x08;DelayMs(5);
      }
}
```

# C Program to Control Stepper Motor using 8051 Contd….

```
void AntiClockwise (void)
{
    unsigned int i;
    for (i=0;i<30;i++)
    {
        P0 = 0x08;DelayMs(5);          //Delay 20msec
        P0 = 0x04;DelayMs(5);
        P0 = 0x02;DelayMs(5);
        P0 = 0x01;DelayMs(5);
    }
}
void main (void)
{
    P0 = 0;                          //Initialize Port0

    while(1)                         //Loop Forever
    {
        Clockwise ();
        DelayMs (100);
        P0   =    0;
        AntiClockwise ();
        DelayMs (100);
        P0   =    0;
    }
}
```

# STEPPER MOTOR ASSEMBLY CODE

```
org 0H

 stepper equ P1

 main:
        mov stepper, #0CH
        acall delay
        mov stepper, #06H
        acall delay
        mov stepper, #03H
        acall delay
        mov stepper, #09H
        acall delay
        sjmp main
```

# ASSEMBLY CODE CONTD

```
delay:
        mov r7,#4
    wait2:
        mov r6,#0FFH
    wait1:
        mov r5,#0FFH
    wait:
        djnz r5,wait
        djnz r6,wait1
        djnz r7,wait2
        ret
        end
```

# Interfacing ADC to 8051

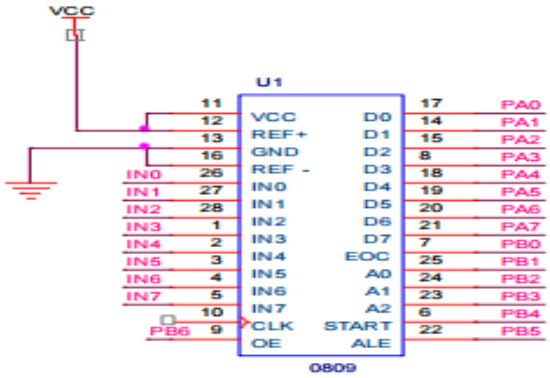
•8051 microcontroller it doesn’t have an On chip ADC to accept the digital input, it will not accept analog input.

• so we need a ADC to process the analog signal.

• For interfacing ADC 0809 we require 8 data lines.

• So ADC 0809 is an 8 bit ADC has 8 channels works on successive approximation conversion technique.

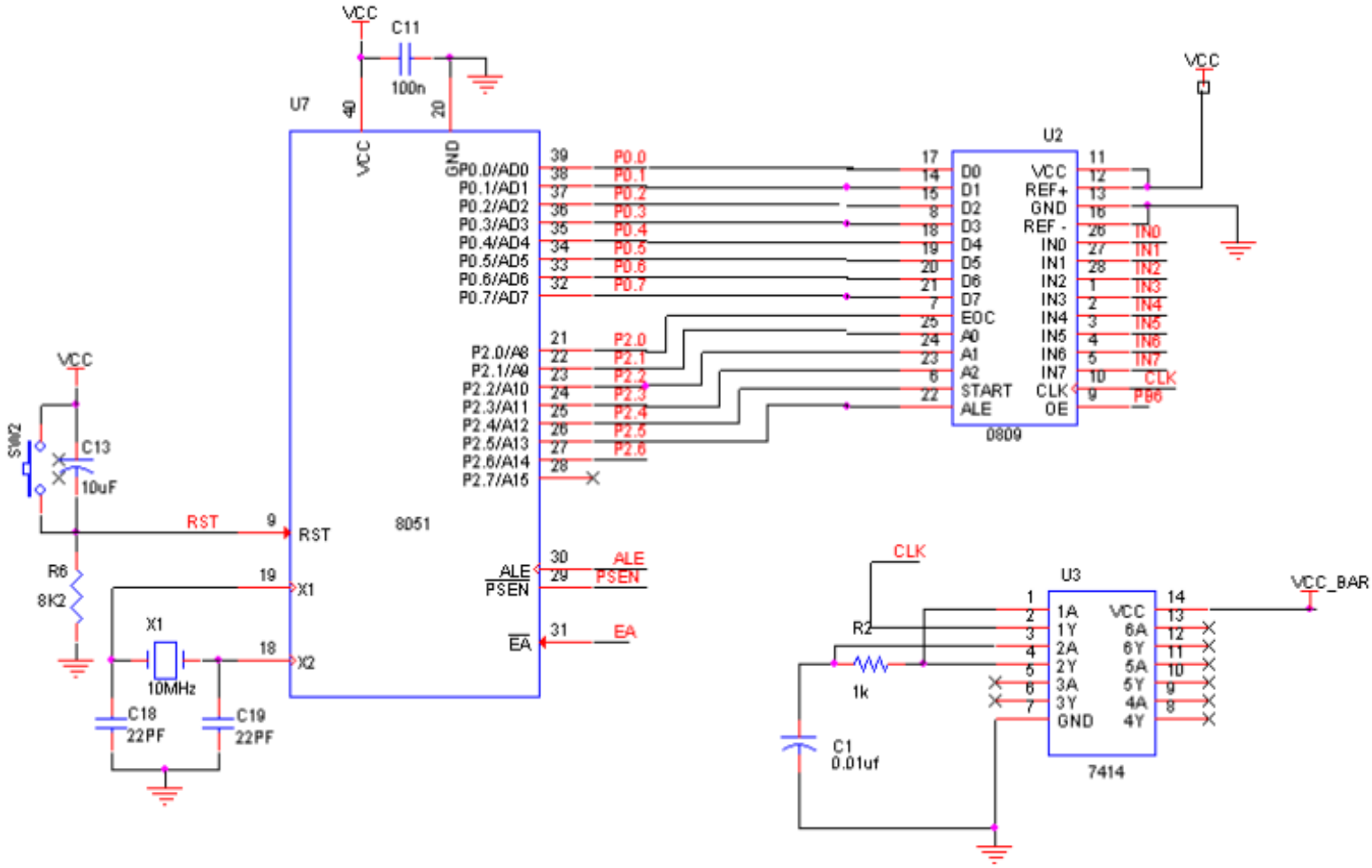
# PIN ASSIGNMENT WITH 8051

| | 20PIN CONNECTOR | ADC 0809 | ADC 0809 |
|---|---|---|---|
| Data lines | PA.0 | D0 |  |
| | PA.1 | D1 | |
| | PA.2 | D2 | |
| | PA.3 | D3 | |
| | PA.4 | D4 | |
| | PA.5 | D5 | |
| | PA.6 | D6 | |
| | PA.7 | D7 | |
| Control Lines | PB.0 | EOC | |
| | PB.1 | A0 | |
| | PB.2 | A1 | |
| | PB.3 | A2 | |
| | PB.4 | Start | |
| | PB.5 | ALE | |
| | PB.6 | OE | |
| | PB.7 | NC | |
| PWR | 17,19 | Vcc | Supply form 8051/8086/80805 trainer Kit |
| | 18,20 | Gnd | |



PA0 – PA7 is connected to 8 no's of LED

# CIRCUIT DIAGRAM TO INTERFACE ADC 0809 WITH 8051



# 8051 Assembly Code

```
ale equ P3.4
oe equ P3.7
start equ P3.5
eoc equ P3.6
sel_a equ P3.1
sel_b equ P3.2
sel_c equ P3.3
adc_data equ P1
org 0H
;Data port to input
mov adc_data, #0FFH
;EOC as Input
setb eoc
;rest of output signals
clr ale
clr oe
clr start
main_loop:
;Select Analog Channel 1
setb sel_a
clr sel_b
clr sel_c
;Latch channel select
setb ale
```

# 8051 Assembly Code Contd…

```
;Start conversion
setb start
clr ale
clr start
;Wait for end of conversion
jb eoc, $ ; $ means jump to same location
jnb eoc, $
;Assert read signal
setb oe
 ; Read Data
mov A, adc_data
clr oe
 ;ADC data is now in accumulator
;Start over for next conversion
 sjmp main_loop
 end
```

# DAC Interfacing with 8051

• Microcontroller are used in wide variety of applications like for measuring and control of physical quantity like temperature, pressure, speed, distance, etc.

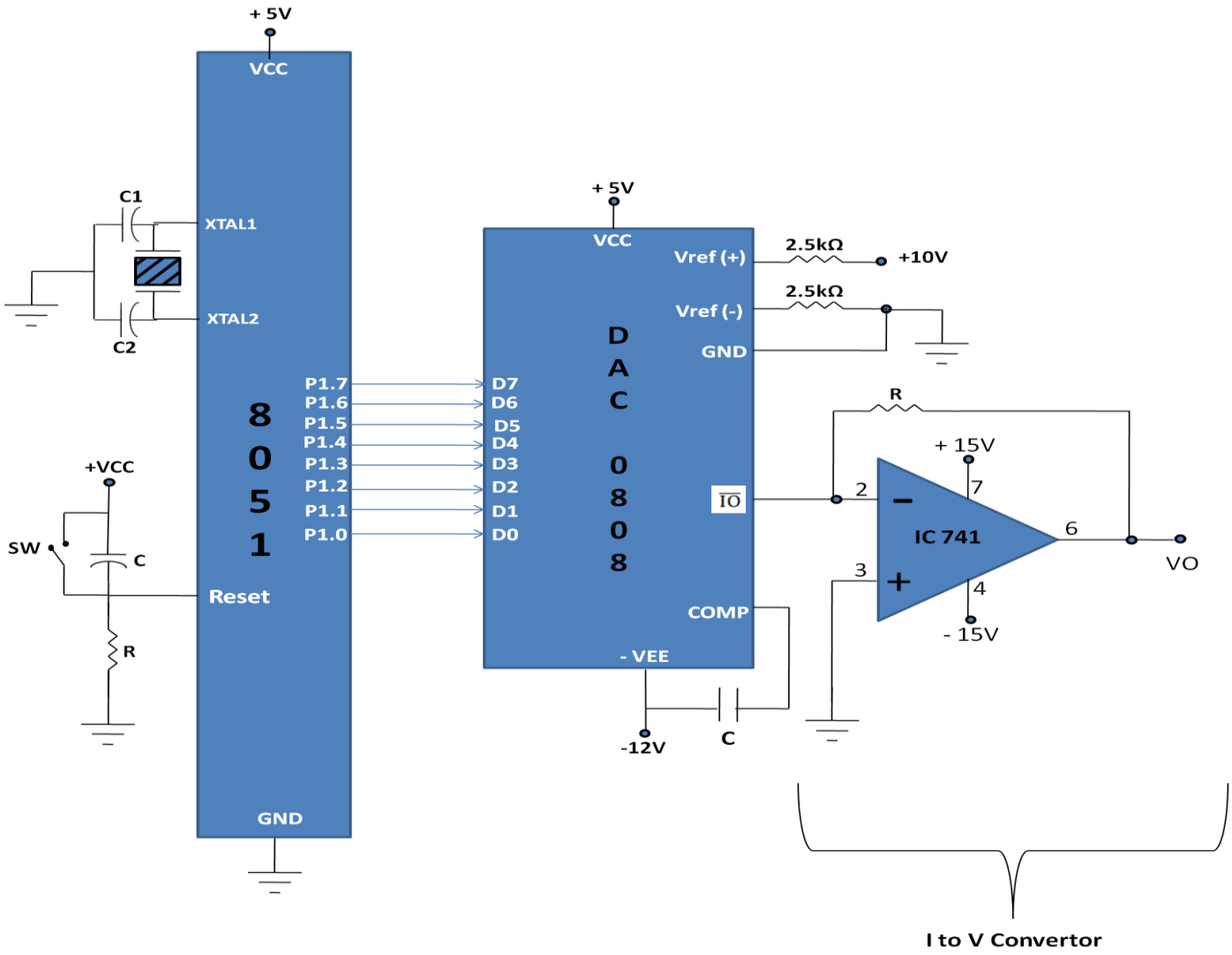
• In these systems microcontroller generates output which is in digital form but the controlling system requires analog signal as they don't accept digital data thus making it necessary to use DAC which converts digital data into equivalent analog voltage.

• The IC DAC0808 converts digital data into equivalent analog Current. Hence we require an I to V converter to convert this current into equivalent voltage.

• According to theory of DAC Equivalent analog output is given as:

$$V0=Vref\left[\frac{D0}{2}+\frac{D1}{4}+\frac{D2}{8}+\frac{D3}{16}+\frac{D4}{32}+\frac{D5}{64}+\frac{D6}{128}+\frac{D7}{256}\right]$$

# Interfacing Diagram



To generate triangular waveform using DAC.

**Program**

```
        MOV  DPTR, #FFC8
START   MOV  A, #00
LOOP1   MOV  @DPTR,A
        INC  A
        JNZ  LOOP1
        MOV  A, #FF
LOOP2   MOVX @DPTR,A
        DEC  A
        JNZ  LOOP2
        LJMP START
```

# To generate square wave using DAC.

***PROGRAM***

```
START   MOV   DPTR,#FFC8
        MOV A,#00
        MOVX  @DPTR,A
        LCALL  DELAY
        MOV    A,#FF
        MOVX  @DPTR,A
        LCALL  DELAY
        LJMP   START
DELAY   MOV    R1,  #05
LOOP    MOV    R2,  #FF
HERE    DJNZ   R2,  HERE
        DJNZ   R1, LOOP
        RET
```