MODULE 3

- **Programming and Interfacing of 8051**

Simple programming examples in assembly language: Addition, Subtraction, Multiplication and Division. Interfacing of LCD display, Keyboard, Stepper Motor, DAC and ADC with 8051.

Course Outcome

- After completion of the module the student will be able to
  - Interface the various peripheral devices to the microcontroller using assembly/ C programming

8051 Program to Add two 8- Bit numbers

Here we will add two 8-bit numbers using 8051 microcontroller. The register A (Accumulator) is used as one operand in the operations. There are seven registers R0 – R7 in different register banks. We can use any of them as the second operand.

a) Using internal data memory
(Here we will use the internal data memory of 8051 to store the data)

We are taking two numbers 5FH and D8H at location 20H and 21H, After adding them, the result will be stored at location 30H and 31H.

INPUT

Address	Value
20 H	5F H
21 H	D8 H
30 H	00 H
31 H	00 H

```
        MOV R2,#00H;        Clear register R2 to store carry
        MOV R0,#20H;        set source address 20H to R0
        MOV R1,#30H;        set destination address 30H to R1
        MOV A,@R0;          take the value from source to register A
        MOV R3,A;           Move the value from A to R3
        INC R0;             Point to the next location(21h)
        MOV A,@R0;          Take the value from source to register A
        ADD A,R3;           Add R3 with A and store to register A
        JNC  SAVE           If there is no carry after addition then jump to location
                            pointed by the label SAVE,
        INC R2;             else if there is a carry then Increment R2 to get carry
SAVE:   MOV @R1,A;          Store the result  to 30 H
        INC R1;             Increase R1 to point to the next address(31H)
        MOV A,R2;           Get carry to register A
        MOV @R1,A;          Store the carry to 31 H
```

HALT: SJMP HALT; Stop the program

So by adding 5FH + D8H, the result will be 137H. 37H will be stored at 30H and 01H will be stored at 31H

OUTPUT

Address	Value
20 H	5F H
21 H	D8 H
30 H	37 H
31 H	01 H

b) Using external data memory
(Here we will use the external data memory of 8051 to store the data)

We are taking two numbers 5FH and D8H at location 4200H and 4201H, After adding them, the result will be stored at location 4300H and 4301H.

INPUT

Address	Value
4200 H	5F H
4201 H	D8 H
4300 H	00 H
4301 H	00 H

```
MOV R2,#00H;          Clear register R2 to store carry
MOV DPTR,#4200H;      set source address 4200H to DPTR
MOVX A,@DPTR;         take the value from source to register A
MOV R3,A;             Move the value from A to R3
INC DPTR;             Point to the next location (4201h)
MOVX A,@DPTR;         take the value from source to register A
ADD A,R3;             Add R3 with A and store to register A
JNC  SAVE             If there is no carry after addition then jump to
                      location pointed by the label SAVE,
INC R2;           else if there is a carry then Increment R2 to get carry
SAVE:  INC DPH        Point to the destination location(4300h)
```

```
        MOVX @DPTR,A;          Store the result  to 4300 H
        INC DPTR;              Increase DPTR to point to the next address (4301H)
        MOV A,R2;              Get carry to register A
        MOVX @DPTR,A;          Store the carry to 4301 H
HALT:   SJMP HALT;             Stop the program
```

So by adding 5FH + D8H, the result will be 137H. Then 37H is stored at 4300H and 01H will be stored at 4301H

OUTPUT

Address	Value
4200 H	5F H
4201 H	D8 H
4300 H	37 H
4301 H	01 H

2) 8051 Program to subtract two 8- Bit numbers

Here we will see how to subtract two 8-bit numbers using this microcontroller. The register A (Accumulator) is used as one operand in the operations. There are seven registers R0 – R7 in different register banks. We can use any of them as the second operand.

We are taking two numbers 73H and BDH at location 20H and 21H. After subtracting, the result will be stored at location 30H and 31H.

INPUT

Address	Value
20 H	73 H
21 H	BD H
30 H	00 H
31 H	00 H

Program

```
        MOV R2,#00H;        Clear register R2 to store borrow
        MOV R0,#20H;        set source address 20H to R0
        MOV R1,#30H;        set destination address 30H to R1
        MOV A,@R0;          take the value from source to register A
        MOV R5,A;           Move the value from A to R5
        INC R0;             Point to the next location
        MOV A,@R0;          take the value from source to register A
        MOV R3,A;           store second byte
        MOV A,R5;           get back the first operand
        SUBB A,R3;          Subtract R3 from A
        JNC  SAVE           If there is no borrow after addition then jump to
                            location pointed by the label SAVE,
        INC R2;             else if there is a borrow then Increment R2 to get
                            borrow
SAVE:   MOV @R1,A;          Store the result at location 30H
        MOV A, R2;          Get borrow to register A
```

```
        INC R1;              Increase R1 to point to the next address (31H)
        MOV @R1,A;           Store the borrow at location 31H
HALT:   SJMP HALT ;          Stop the program
```

So by subtracting 73H –BDH, the result will be B6H. At location 30H, we will get the result B6H and at location 31H, we will get 01H. This indicates that the result is negative. The get the actual value from result B6H, we have to perform 2's complement operation. After performing 2's Complement, the result will be -4AH.

OUTPUT

Address	Value
20 H	73 H
21 H	BD H
30 H	B6 H
31 H	01 H

3) 8051 Program to Multiply two 8 Bit numbers

Now we will try to multiply two 8-bit numbers using this 8051 microcontroller. The register A and B will be used for multiplication. No other registers can be used for multiplication. The result of the multiplication may exceed the 8-bit size. So the higher order byte is stored at register B, and lower order byte will be in the Accumulator A after multiplication.

We are taking two numbers FFH and FFH at location 20H and 21H, After multiplying the result will be stored at location 30H and 31H.

INPUT

Address	Value
20 H	FF H
21 H	FF H
30 H	00 H
31 H	00 H

```
MOV R0, #20H;        set source address 20H to R0
MOV R1, #30H;        set destination address 30H to R1
MOV A, @R0;          take the first operand from source to register A
```

```
        MOV B,A          and copy to B
        INC R0;          Point to the next location
        MOV A,@R0;       take the second operand from source to register A
        MUL AB ;         Multiply A and B
        MOV @R1, A;      Store lower order byte to 30H
        INC R1;          Increase R1 to point to the next location
        MOV A,B          Copy the higher byte to A
        MOV @R1, A;      Store higher order byte to 31H
HALT:   SJMP HALT ;      Stop the program
```

OUTPUT

Address	Value
20 H	FF H
21 H	FF H
30 H	01 H (Lower Byte result)
31 H	FE H (Higher Byte result)

4) 8051 Program to Divide two 8 Bit numbers

Now we will see another arithmetic operation. The divide operation to divide two 8-bit numbers using 8051 microcontroller. The register A and B will be used in this operation. No other registers can be used for division. The result of the division has two parts: The quotient part and the remainder part. Register A will hold Quotient, and register B will hold Remainder.

We are taking two number 0EH and 03H at location 20H and 21H, After dividing, the result will be stored at location 30H (Quotient) and 31H (Remainder).

INPUT

Address	Value
20 H	0E H
21 H	03 H
30 H	00 H
31 H	00 H

Program

```
        MOV R0, #20H;          set source address 20H to R0
        MOV R1, #30H;          set destination address 30H to R1
        MOV A, @R0;            take the first operand from source to register A
        MOV R3,A               and copy it to R3
        INC R0;                Point to the next location
        MOV A, @R0;            take the second operand from source to register A
        MOV B,A                and copy to B
        MOV A,R3               Get the 1st number again in A from R3
        DIV AB ;               Divide A by B
        MOV @R1, A;            Store Quotient to 30H
        INC R1;                Increase R1 to point to the next location
        MOV A,B                Copy the remainder from B register to A
        MOV @R1, A;            Store Remainder to 31H
HALT:   SJMP HALT ;            Stop the program
```

OUTPUT

Address	Value
20 H	0E H
21 H	03 H
30 H	04 H (Quotient)
31 H	02 H (remainder)

➢ LCD INTERFACING

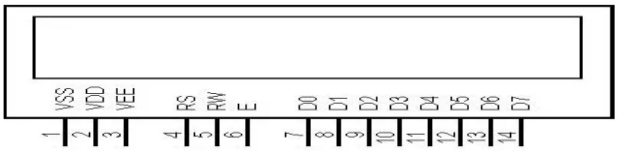
This section describes the operation modes of LCDs, then describes how to program and interface an LCD to an 8051 using Assembly and C.

LCD operation

LCD is finding widespread use replacing LEDs for the following reasons:

- ➢ The declining prices of LCD
- ➢ The ability to display numbers, characters, and graphics
- ➢ Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of
- ➢ the task of refreshing the LCD
- ➢ Ease of programming for characters and graphics.

LCD pin descriptions



The LCD discussed in this section has 14 pins. The function of each pin is given in Table

Table : Pin Descriptions for LCD

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	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

- ***V_{CC} / V_{DD} , V_{SS}, and V_{EE}***

While V_{cc} and V_{ss} provide +5V and ground, respectively, V_{EE} is used for controlling LCD contrast.

- ***RS - register select***

There are two very important registers inside the LCD. The RS pin is used for their selection as follows.

- If RS = 0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc.
- If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

- ***R/W-read/write***

R/W input allows the user to write information to the LCD or read information from it.

- R/W = 1 when reading;

- R/W = 0 when writing.
- ***E - enable***

The enable pin is used by the LCD to latch information presented to its data pins.

When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide.

- **DO-D7**

The 8-bit data pins, DO – D7, are used to send information to the LCD or read the contents of the LCD's internal registers.

- To display letters and numbers, we send ASCII codes for the letters A – Z, a – z, and numbers 0 – 9 to these pins while making RS = 1.
- There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor. Few commands are listed in table.

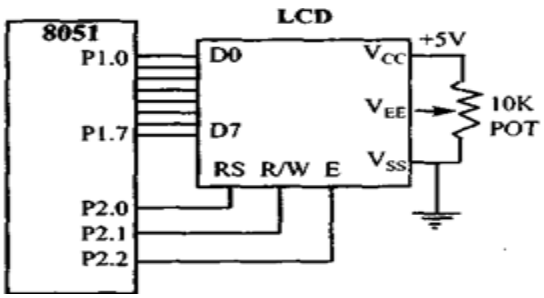
Table : LCD Command Codes

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 of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7 and can be read when R/W = 1 and RS = 0, as follows:

- if R/W= 1, RS = 0.
  - When D7 = 1 (busy flag = 1), the LCD is busy taking care of internal operations and will not accept any new information.
  - When D7 = 0, the LCD is ready to receive new information. Note: It is recommended to check the busy flag before writing any data to the LCD.

Circuit Diagram



- **Sending code or data to the LCD with checking busy flag**

  To send any of the commands from Table 2 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.

  The steps involved can be summarized as follows:

  - **Checking the busy flag from LCD before issuing a command or data to LCD**
    - To read the command register we make R/W = 1 and RS = 0, and a L-to-H pulse for the E pin will provide us the command register.
    - After reading the command register, if bit D7 (the busy flag) is high, the LCD is busy and no information (command or data) should be issued to it.
    - Only when D7 = 0 can we send data or commands to the LCD.

```
; 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
```

```
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. 7
ACALL COMMAND           ;issue command
MOV A,#"H"              ;display letter H
ACALL DATA_DISPLAY      ;issue data
```

```
        MOV A,#"I"                  ;display letter I
        ACALL   DATA_DISPLAY        ;issue data
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                 ; return to main program
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                 ; return to main program
READY:          SETB P1.7           ;make P1.7 input pin
                CLR P2.0            ;RS=0 access command reg
                SETB P2.1           ;R/W=1 read command reg ;
        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 here until busy flag=0
                RET
```

********** C Program**********

```
#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
        lcddata('H');
        lcddata('I');
```

```
}
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;
}
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 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 accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor. When a key is pressed, a row and a column make a contact. Otherwise, there is no connection between rows and columns. A 4x4 matrix connected to two ports. Rows are connected to an output port and the columns are connected to an input port.

Scanning and Identifying the Key:

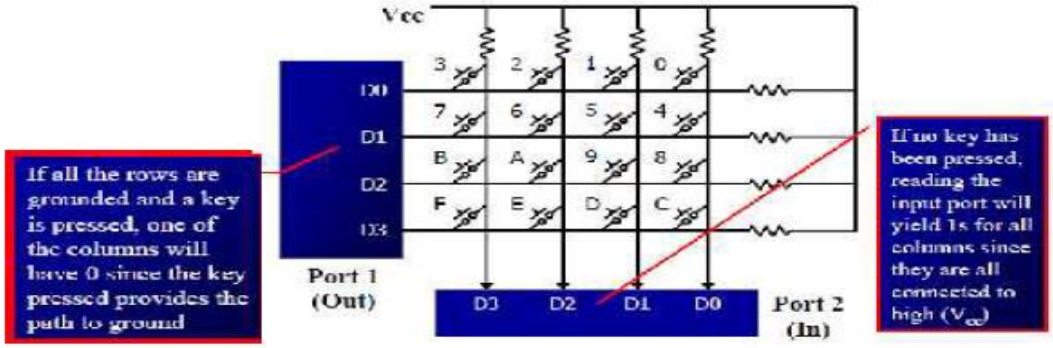


Fig: A 4X4 matrix keyboard

It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns
- If the data read from columns is D3 – D0 =1111, no key has been pressed and the process continues till key press is detected
- If one of the column bits has a zero, this means that a key press has occurred For example, if D3 – D0 = 1101, this means that a key in the D1 column has been pressed After detecting a key press, controller will go through the process of identifying the key.
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only. It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row

- It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified.
- After identification of the row in which the key has been pressed. Find out which column the pressed key belongs to.

Algorithm for detection and identification of key activation goes through following stages:

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high

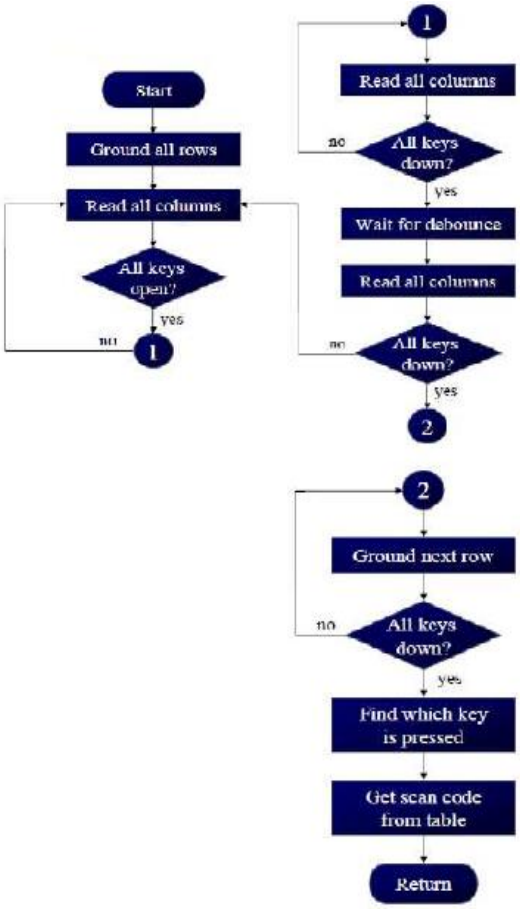
 - When all columns are found to be high, the program waits for a short amount of time before it goes to next stage of waiting for a key to be pressed

2. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it

 - Remember that the output latches connected to rows still have their initial zeros making them grounded

- After the key press detection, it waits 20 ms for the bounce and then scans the columns again

 (a) It ensures that the first key press detection was not an erroneous one due a spike noise



 (b) The key press. If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press

3. To detect which row key press belongs to, it grounds one row at a time, reading the columns each time

 -If it finds that all columns are high, this means that the key press cannot belong to that row. Therefore, it grounds the next row and continues until it finds the row key press belongs to.

- Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row

4. To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and

checks to see if it is low

- Upon finding the zero, it pulls out the ASCII code for that key from the look-up table
- otherwise, it increments the pointer to point to the next element of the look-up table.

➢ Stepper Motor Interfacing

Stepper Motor

A stepper motor is a device that translates electrical pulses into mechanical movement in steps of fixed step angle. Stepper motor is brushless DC motor, which can be rotated in small angles, these angles are called steps. Generally stepper motor use 200 steps to complete 360 degree rotation, means it rotates 1.8 degree per step.

- The stepper motor rotates in steps in response to the applied signals.
- It is mainly used for position control.
- It is used in disk drives, dot matrix printers, plotters and robotics and process control circuits.

Stepper motor is used in many devices which need precise rotational movement like robots, antennas, hard drives etc. We can rotate stepper motor to any particular angle by giving it proper instructions.

ADVANTAGES OF STEPPER MOTOR

Stepper motors are used because of their:

- Low cost
- High reliability (because of no contact brushes)
- Good performance in starting, stopping and reversing the direction
- High torque at low speed

Construction

Stepper motor is made up of stator and rotor. The construction of the motor is as shown in figure below. It has a permanent magnet rotor called the shaft which is surrounded by stator. The stator represent four electromagnets over which the electric coil is wound. Whenever the coils wounded on the stator are energised by applying current, electromagnetic field is created, which results in the rotation of rotor (permanent magnet). Coils should be energised in a particular sequence to make the rotor rotate.

One end of the coil are connected commonly either to ground or +5V. The other end is provided with a fixed sequence such that the motor rotates in a particular direction. Stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a precise position. Direction of the rotation is dictated by the stator poles. Stator poles are determined by the current sent through the wire coils.

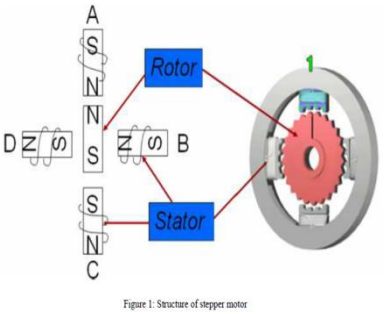
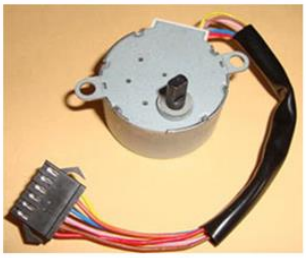
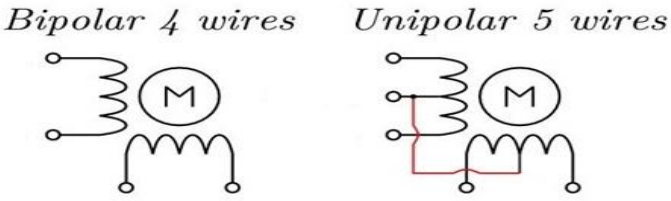


Figure 1: Structure of stepper motor



Stepper motors are basically two types: Unipolar and Bipolar.

Commonly used stepper motors have four stator windings that are paired with a center – tapped common. Such motors are called as four-phase or **Unipolar stepper motor.** Unipolar stepper motor generally has five or six wires, in which four wires are one end of four stator coils, and other end of the all four coils is tied together which represents fifth wire, this is called common wire (common point). Unipolar stepper motor is very common and popular because of its ease of use.



In **Bipolar stepper motor** there is just four wires coming out from two sets of coils, means there are no common wire.

STEP MODES:

Unipolar stepper motors can be used in three modes (based on the "sequence" of energizing the coil)

- Wave Drive mode
- Full Drive mode
- Half Drive mode

1) Wave drive mode:

- In this mode one coil is energised at a time.
- All four coil are energised one after another.
- Torque which is generated will be less than full drive.
- Power consumption is reduced in it.
- This type of drive is chosen when the power consumption fact is more important than torque.

Following is the table for producing this mode using microcontroller, which means we need to give Logic 1 to the coils in the sequential manner

Clockwise

Step #	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Counter-clockwise

2) Full Drive mode:

- Two electromagnets are energized at single time.
- Torque which is generated will be greater than wave drive.
- Power consumption is higher in it.
- This type of drive is chosen when the torque is more important than power consumption.

Clockwise

Step #	A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Counter-clockwise

We need to give Logic 1 to two coils at the same time, then to the next two coils and so on.

3) Half Drive:

- One and two electromagnets are alternatively energized in this mode.
- It is a combination of both, wave drive and full drive.
- It has low torque but the angular resolution is doubled.
- This type of drive is chosen when we want to increase the angular resolution of the motor.

Clk

Step #	A	B	C	D
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

Anti-Clk

Since each drive has its own advantages and disadvantages, thus we choose the drive according to the application we want and power consumption.

In this mode one and two coils are energised alternatively, means firstly one coil is energised then two coils are energised then again one coil is energised then again two, and so on.

Step Angle: The important parameter of a stepper motor is the step angle. It is the minimum degree of rotation associated with a single step.

Steps per revolution: This is the total number of steps needed to rotate one complete rotation or 360 degrees.

In a four phase motor if there are 200 steps in one complete rotation then the step angle is $360/200 = 1.8^0$.

Interfacing Stepper Motor with 8051 Microcontroller

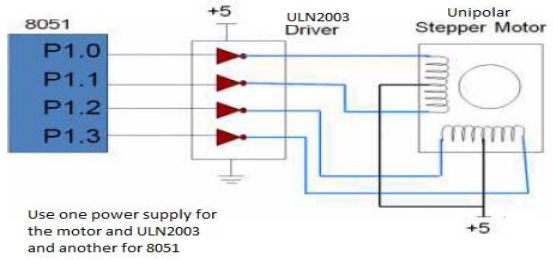
Even a small stepper motor require a current of 400 mA for its operation. But the ports of the microcontroller cannot source this much amount of current. If such a motor is directly connected to the microprocessor/microcontroller ports, the motor may draw large current from the ports and damage it.

So a suitable driver circuit is used with the microprocessor/microcontroller to operate the motor. 8051 doesn't provide enough current to drive the coils of the motor, so we need to use a **current driver IC that is ULN2003A**.

- **Motor Driver Circuit (ULN2003)**

Stepper motor driver circuits are available readily in the form of ICs. ULN2003 is one such driver IC which is a High-Voltage High-Current Darlington transistor array and can give a current of 500mA.This current is sufficient to drive a small stepper motor. Internally, it has protection diodes used to protect the motor from damage due to back emf and large eddy currents. So, this ULN2003 is used as a driver to interface the stepper motor to the microcontroller.

Interfacing with 8051 is very easy we just need to give the 0 and 1 to the four wires of stepper motor according to the above tables depending on which mode we want to run the stepper motor. And rest two wires should be connected to a proper 12v supply (depending on the stepper motor). Here we have used the unipolar stepper motor. We have connected four ends of the coils to the first four pins of port 1 of 8051 through the ULN2003.



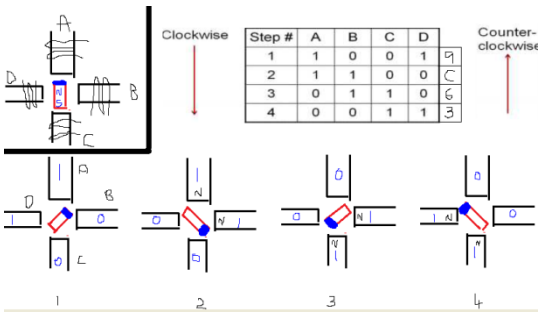


Figure: 8051 connection to stepper motor

So to rotate the stepper motor we have to apply the excitation pulse. For this the controller should send a hexadecimal code through one of its ports. The hex code mainly depends on the construction of the stepper motor. So, all the stepper motors do not have the same Hex code for their rotation.

Program

To rotate the stepper motor clockwise / anticlockwise continuously with full step sequence.

```
        MOV A, #66H          ; load step sequence
BACK:   MOV P1,A             ; issue the sequence to motor
        RR A                 ; rotate right for clockwise rotation
        ACALL   DELAY        ;wait
        SJMP    BACK         ;keep going
DELAY:  MOV R1, #100
  UP1:  MOV R2, #50
  UP:   DJNZ R2,UP
        DJNZ R1,UP1
        RET
```

Note: * For the motor to rotate in anticlockwise use instruction RL A instead of RR A

**** Change the value of DELAY to set the speed of rotation***

********** C Program**********

```
#include <reg51.h>
void main ( )
{
        while (1)
```

```
        {
           P1=0x66;
           MSDELAY (200);
           P1=0x33;
          MSDELAY (200);
          P1=0x99;
          MSDELAY (200);
          P1=0xCC;
          MSDELAY (200);
        }
}
void MSDELAY (unsigned char value)
{
unsigned int x,y;
for(x=0;x<1275;x++)
for(y=0;y<value;y++);
}
```

➢ Digital-to-Analog (DAC) converter:

The DAC is a device widely used to convert digital pulses to analog signals. In this section we will discuss the basics of interfacing a DAC to 8051. The two methods of creating a DAC is binary weighted and R/2R ladder.

The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2n, where n is the number of data bit inputs.

DAC0808:

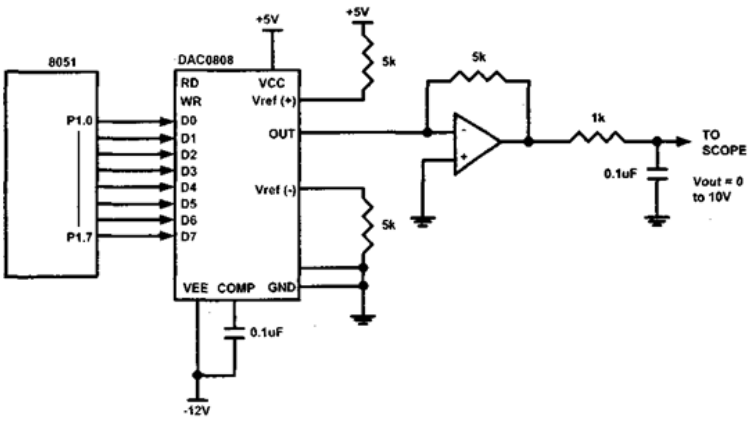
The digital inputs are converter to current Iout, and by connecting a resistor to the Iout pin, we can convert the result to voltage. The total current Iout is a function of the binary numbers at the D0-D7 inputs of the DAC0808 and the reference current Iref , and is as follows:

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

Usually reference current is 2mA.

Interfacing DAC 0808 with 8051

Ideally we connect the output pin to a resistor, convert this current to voltage, and monitor the output on the scope. But this can cause inaccuracy; hence an opamp is used to convert the output current to voltage. The 8051 connection to DAC0808 is as shown in the figure



Example : Write an Assembly Language Program (ALP) to generate a triangular waveform.

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

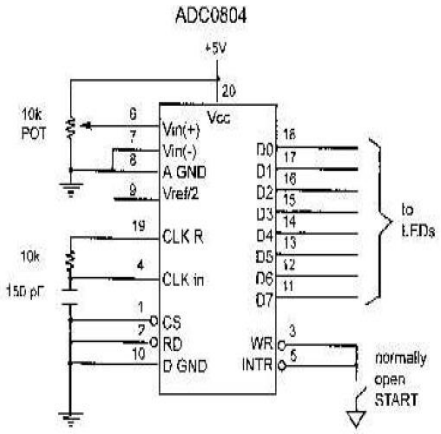
➢ Analog-to-digital converter (ADC) interfacing:

ADCs (analog-to-digital converters) are among the most widely used devices for data acquisition. A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current) signals using a device called a transducer, or sensor. We need an analog-to-digital converter to translate the analog signals to digital numbers, so microcontroller can read them.

ADC804 Chip:

ADC804 IC is an analog-to-digital converter. It works with +5 volts and has a resolution of 8 bits. Conversion time is another major factor in judging an ADC. Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number. In ADC804 conversion time varies depending on the clocking signals applied to CLK R and CLK IN pins, but it cannot be faster than 110μs.

0804- **Pin Description of ADC804:**



➢ **CLK IN and CLK R:**

CLK IN is an input pin connected to an external clock source. To use the internal clock generator (also called self-clocking), CLK IN and CLK R pins are connected to a capacitor and a resistor and the clock frequency is determined by: $f = \frac{1}{1.1RC}$

Typical values are R = 10K ohms and C =150pF. We get f = 606 kHz and the conversion time is 110μs.

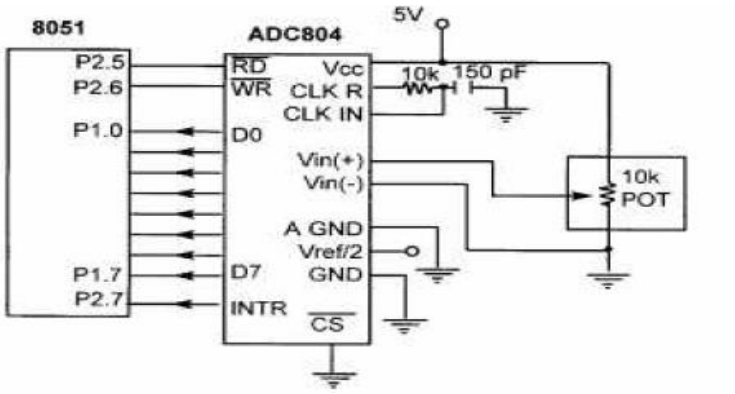
➢ **Vref/2 :** It is used for the reference voltage. If this pin is open (not connected), the analog input voltage is in the range of 0 to 5 volts (the same as the Vcc pin). If the analog input range needs to be 0 to 4 volts, Vref/2 is connected to 2 volts. Step size is the smallest change can be discerned by an ADC.

➢ **D0-D7:** The digital data output pins. These are tri-state buffered. The converted data is accessed only when CS =0 and RD is forced low. To calculate the output voltage, use the following formula $D_{out} = \frac{V_{in}}{step\ size}$

- Dout = digital data output (in decimal),
- Vin = analog voltage, and
- step size (resolution) is the smallest change

- **Analog ground and digital ground:** Analog ground is connected to the ground of the analog Vin and digital ground is connected to the ground of the Vcc pin. To isolate the analog Vin signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.
- **Vin(+) & Vin(-):** Differential analog inputs where Vin = Vin (+) – Vin (-). Vin (-) is connected to ground and Vin (+) is used as the analog input to be converted.
- **RD:** Is "output enable" a high-to-low RD pulse is used to get the 8-bit converted data out of ADC804.
- **INTR:** It is "end of conversion" When the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up.
- **WR:** It is "start conversion" When WR makes a low-to-high transition, ADC804 starts converting the analog input value of Vin to an 8- bit digital number.
- **CS:** It is an active low input used to activate ADC804.

The 8051 connection to ADC0804 is as shown in the figure below



The following steps must be followed for data conversion by the ADC804 chip:

1. Make CS= 0 and send a L-to-H pulse to pin WR to start conversion.
2. Monitor the INTR pin, if high keep polling but if low, conversion is complete, go to next step.
3. Make CS= 0 and send a H-to-L pulse to pin RD to get the data out