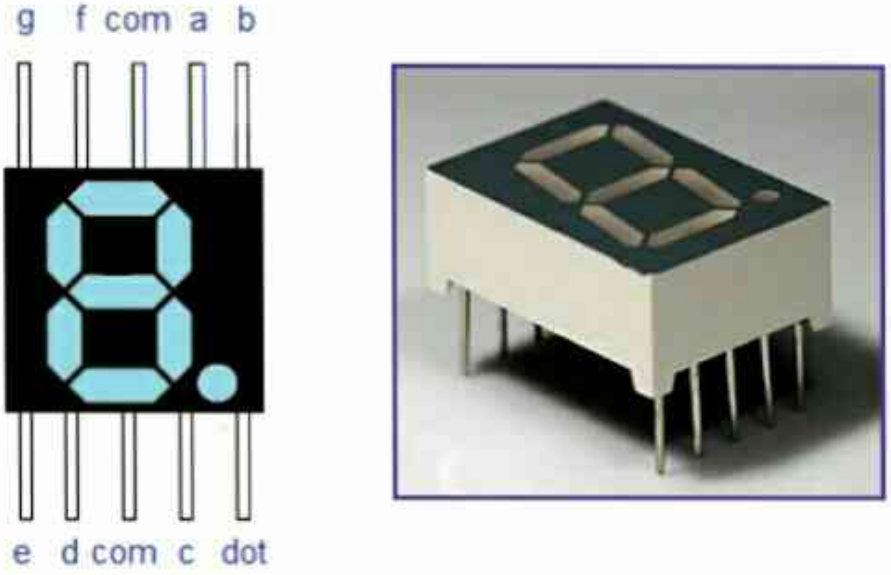
A Note about 7 segment LED display.

This article is about how to interface a seven segment LED display to an 8051 microcontroller. 7 segment LED display is very popular and it can display digits from 0 to 9 and quite a few characters like A, b, C, ., H, E, e, F, n, o,t,u,y, etc. Knowledge about how to interface a seven segment display to a micro controller is very essential in designing embedded systems. A seven segment display consists of seven LEDs arranged in the form of a squarish **'8'** slightly inclined to the right and a single LED as the dot character. Different characters can be displayed by selectively glowing the required LED segments. Seven segment displays are of two types, ***common cathode and common anode.*** In common cathode type , the cathode of all LEDs are tied together to a single terminal which is usually labeled as **'com'** and the anode of all LEDs are left alone as individual pins labeled as a, b, c, d, e, f, g & h (or dot) . In common anode type, the anode of all LEDs are tied together as a single terminal and cathodes are left alone as individual pins. The pin out scheme and picture of a typical 7 segment LED display is shown in the image below.



7 segment LED display

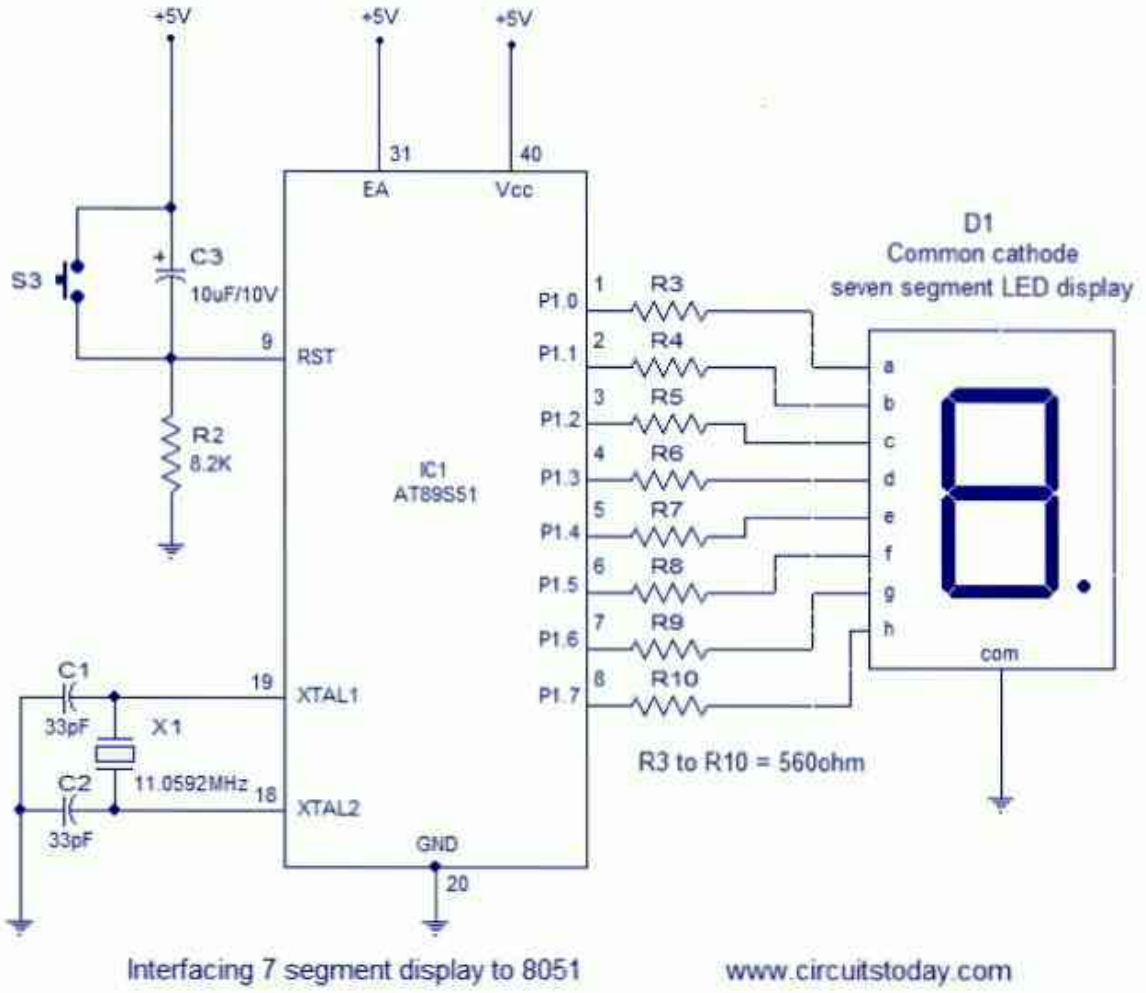
Digit drive pattern.

Digit drive pattern of a seven segment LED display is simply the different logic combinations of its terminals **'a' to 'h'** in order to display different digits and characters. The common digit drive patterns (0 to 9) of a seven segment display are shown in the table below.

Digit	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Interfacing seven segment display to 8051.

Interfacing seven segment display to 8051.



Interfacing 7 segment display to 8051

The circuit diagram shown above is of an AT89S51 microcontroller based 0 to 9 counter which has a 7 segment LED display interfaced to it in order to display the count. This simple circuit illustrates two things. How to setup simple 0 to 9 up counter using 8051 and more importantly how to interface a seven segment LED display to 8051 in order to display a particular result. The common cathode seven segment display D1 is connected to the Port 1 of the microcontroller (AT89S51) as shown in the circuit diagram. R3 to R10 are current limiting resistors. S3 is the reset switch and R2,C3 forms a debouncing circuitry. C1, C2 and X1 are related to the clock circuit. The software part of the project has to do the following tasks.

- Form a 0 to 9 counter with a predetermined delay (around 1/2 second here).
- Convert the current count into digit drive pattern.
- Put the current digit drive pattern into a port for displaying.

All the above said tasks are accomplished by the program given below.

Program.

```
 ORG 000H //initial starting address
START: MOV A,#00001001B // initial value of accumulator
MOV B,A
MOV R0,#0AH //Register R0 initialized as counter which counts from 10 to 0
LABEL: MOV A,B
INC A
MOV B,A
MOVC A,@A+PC // adds the byte in A to the program counters address
MOV P1,A
ACALL DELAY // calls the delay of the timer
DEC R0//Counter R0 decremented by 1
MOV A,R0 // R0 moved to accumulator to check if it is zero in next instruction.
JZ START //Checks accumulator for zero and jumps to START.
Done to check if counting has been finished.
SJMP LABEL
DB 3FH // digit drive pattern for 0
DB 06H // digit drive pattern for 1
DB 5BH // digit drive pattern for 2
DB 4FH // digit drive pattern for 3
DB 66H // digit drive pattern for 4
DB 6DH // digit drive pattern for 5
DB 7DH // digit drive pattern for 6
DB 07H // digit drive pattern for 7
DB 7FH // digit drive pattern for 8
DB 6FH // digit drive pattern for 9
```

```
DELAY: MOV R4,#05H // subroutine for delay
WAIT1: MOV R3,#00H
WAIT2: MOV R2,#00H
WAIT3: DJNZ R2,WAIT3
DJNZ R3,WAIT2
DJNZ R4,WAIT1
RET
END
```

About the program.

Instruction MOVC A,@A+PC is the instruction that produces the required digit drive pattern for the display. Execution of this instruction will add the value in the accumulator A with the content of the program counter(address of the next instruction) and will move the data present in the resultant address to A. After this the program resumes from the line after MOVC A,@A+PC.

In the program, initial value in A is 00001001B. Execution of MOVC A,@A+PC will add oooo1001B to the content in PC (address of next instruction). The result will be the address of label DB 3FH (line15) and the data present in this address ie 3FH (digit drive pattern for 0) gets moved into the accumulator. Moving this pattern in the accumulator to Port 1 will display 0 which is the first count.

At the next count, value in A will advance to 00001010 and after the execution of MOVC A,@+PC ,the value in A will be 06H which is the digit drive pattern for 1 and this will display 1 which is the next count and this cycle gets repeated for subsequent counts.

The reason why accumulator is loaded with 00001001B (9 in decimal) initially is that the instructions from line 9 to line 15 consumes 9 bytes in total.

The lines 15 to 24 in the program which starts with label DB can be called as a **Look Up Table (LUT)**. label DB is known as Define Byte – which defines a byte. This table defines the digit drive patterns for 7 segment display as bytes (in hex format). MOVC operator fetches the byte from this table based on the result of adding PC and contents in the accumulator.

Register B is used as a temporary storage of the initial value of the accumulator and the subsequent increments made to accumulator to fetch each digit drive pattern one by one from the look up table(LUT).

Note:- In line 6, Accumulator is incremented by 1 each time (each loop iteration) to select the next digit drive pattern. Since MOVC operator uses the value in A to fetch the digit drive pattern from LUT, value in ACC has to be incremented/manipulated accordingly. The digit drive patterns are arranged consecutively in LUT.

Register R0 is used as a counter which counts from 10 down to 0. This ensures that digits from o to 9 are continuously displayed in the 7 segment LED. You may note lines 4, 11, 12, and 13 in the above program. Line 4 initializes R0 to 10 (OAh). When the program counter reaches line 11 for the first time, 7 segment LED has already displayed 0. So we can reduce one count and that is why we have written DEC Ro. We need to continuously check if R0 has reached full count (that is 0). In order to do that lines 12 and 13 are used. We move R0 to accumulator and then use the Jump if Zero (JZ) instruction to check if accumulator has reached zero. If Acc=0, then we makes the program to jump to START (initial state) and hence we restart the 7 segment LED to display from 0 to 9 again. If Acc not equal to zero, we continue the program to display the next digit (check line 14).

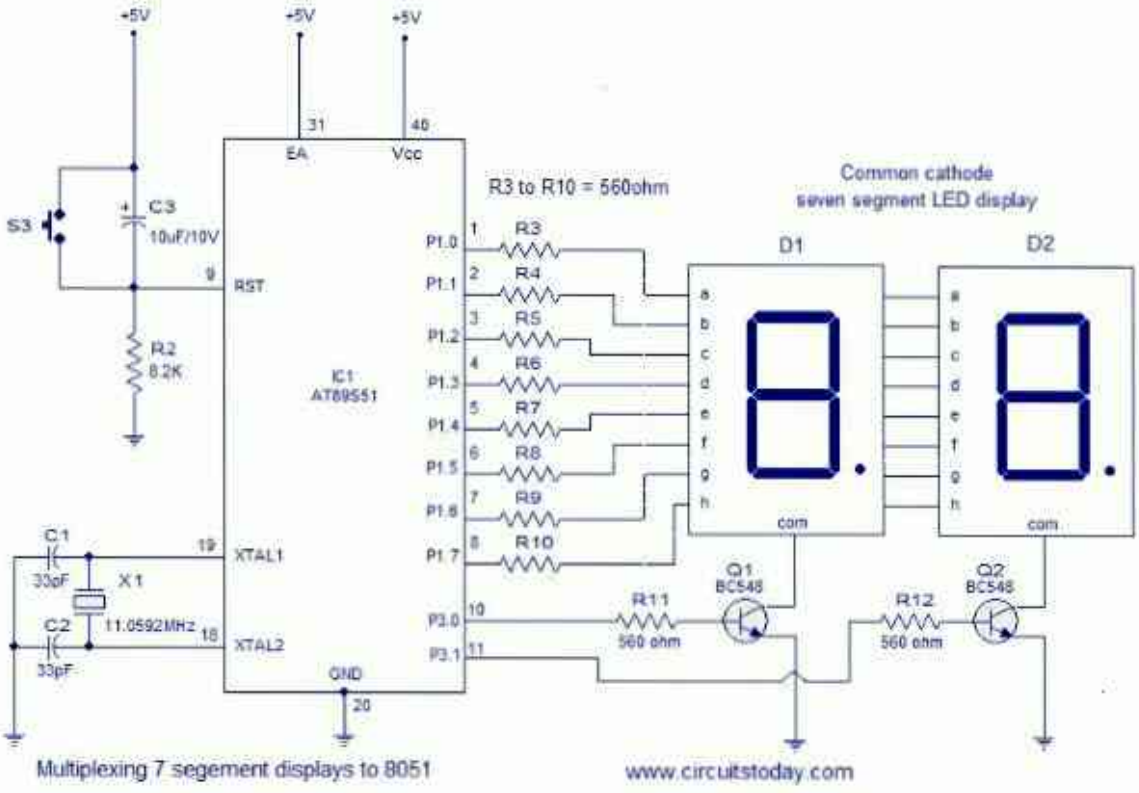
Multiplexing 7 segment display to 8051.

Suppose you need a three digit display connected to the 8051. Each 7 segment display have 8 pins and so a total amount of 24 pins are to the connected to the microcontroller and there will be only 8 pins left with the microcontroller for other input output applications. Also the maximum number of displays that can be connected to the 8051 is limited to 4 because 8051 has only 4 ports. More over three 3 displays will be ON always and this consumes a considerable amount of power. All these problems associated with the straight forward method can be solved by multiplexing .

In multiplexing all displays are connected in parallel to one port and only one display is allowed to turn ON at a time, for a short period. This cycle is repeated for at a fast rate and due to the persistence of vision of human eye, all digits seems to glow. The main advantages of this method are

- Fewer number of port pins are required .
- Consumes less power.
- More number of display units can be interfaced (maximum 24).

The circuit diagram for multiplexing 2 seven segment displays to the 8051 is shown below.



Multiplexing 7 segement display to 8051

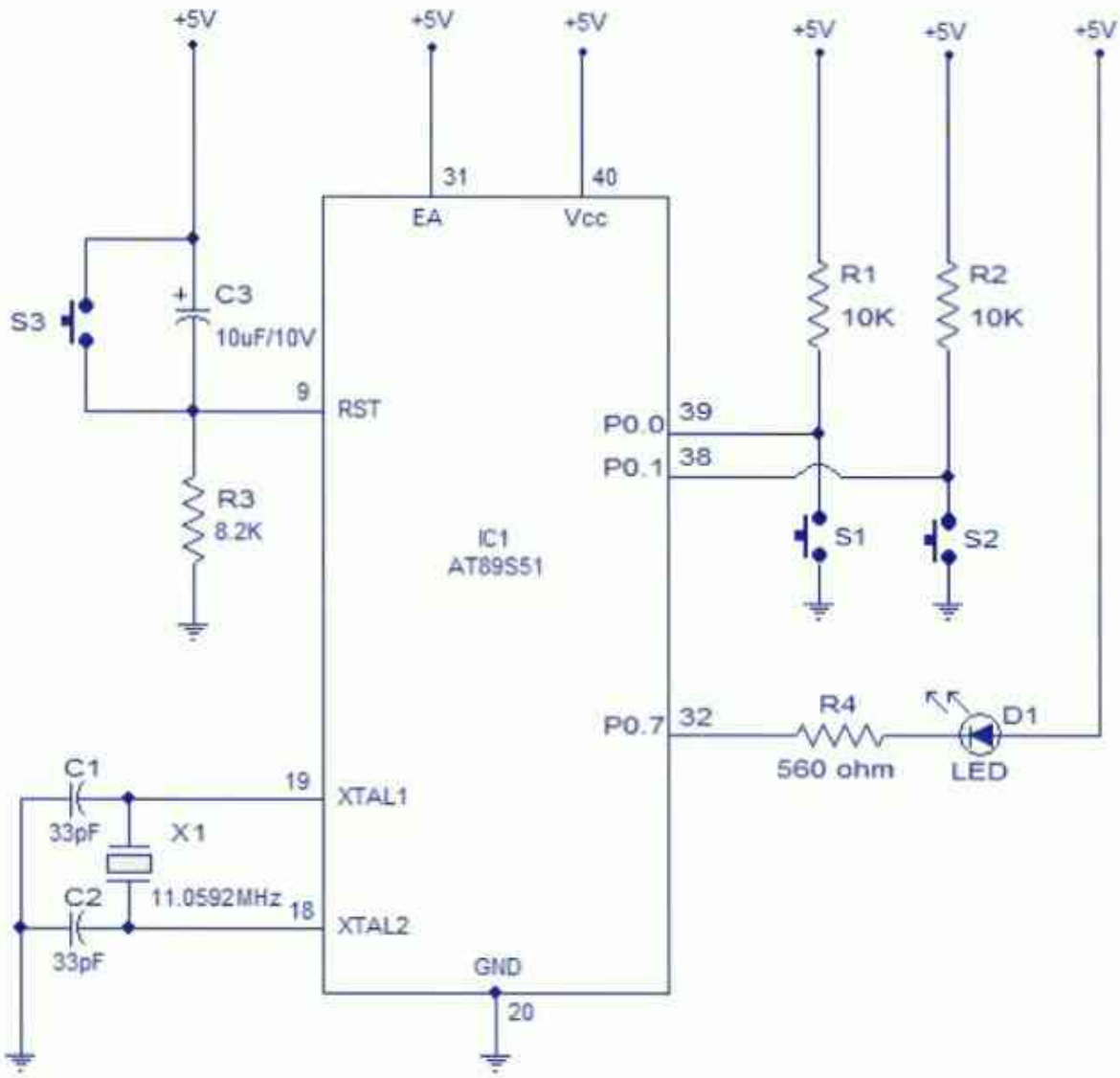
When assembled and powered on, the circuit will display the number '16' and let us see how it is done. Initially the first display is activated by making P3.0 high and then digit drive pattern for "1" is loaded to the Port 1. This will make the first display to show "1". In the mean time P3.1 will be low and so do the second display will be OFF. This condition is maintained for around 1ms and then P3.0 is made low. Now both displays will be OFF. Then the second display is activated by making P3.1 high and then the digit drive pattern for "6" is loaded to the port 1. This will make the second display to show "6". In the mean time P3.0 will be low and so the second display will be OFF. This condition is maintained for another 1ms and then port 3.1 is made low. This cycle is repeated and due to the persistence of vision you will feel it as "16".

Interfacing LED and push button switch to 8051

This article is all about how to interface push button switches to an 8051 microcontroller. Push button switches are widely used in embedded system projects and the knowledge about interfacing them to 8051 is very essential in designing such projects. A typical push button switch has two active terminals that are normally open and these two terminals get internally shorted when the push button is depressed. Images of a typical pushbutton switch is shown below.

Pushbutton switch

Circuit diagram.



Interfacng 8051 and pushbutton

The circuit diagram for interfacing push button switch to 8051 is shown above. AT89S51 is the microcontroller used here. The circuit is so designed that when push button S1 is depressed the LED D1 goes ON and remains ON until push button switch S2 is depressed and this cycle can be repeated. Resistor R3, capacitor C3 and push button S3 forms the reset circuitry for the microcontroller. Capacitor C1, C2 and crystal X1 belongs to the clock circuitry. R1 and R2 are pull up resistors for the push buttons. R4 is the current limiting resistor for LED.

Program.

```
MOV P0,#83H // Initializing push button switches and initializing LED in OFF state.
READSW: MOV A,P0 // Moving the port value to Accumulator.
RRC A // Checking the vale of Port 0 to know if switch 1 is ON or not
JC NXT // If switch 1 is OFF then jump to NXT to check if switch 2 is ON
CLR P0.7 // Turn ON LED because Switch 1 is ON
SJMP READSW // Read switch status again.
NXT: RRC A // Checking the value of Port 0 to know if switch 2 is ON or not
JC READSW // Jumping to READSW to check status of switch 1 again (provided switch 2 is OFF)
SETB P0.7 // Turning OFF LED because Switch 2 is ON
SJMP READSW // Jumping to READSW to read status of switch 1 again.
END
```

The Logic

The first instruction – **MOV P0 #83H** – is to turn LED off (Hex 83 in binary = 10000011) and to initialize switches 1 and 2. Switch 1 is connected to port 0.0 and switch 2 is connected to port 0.1. Also note that LED is connected to port 0.7.

Note:- Po.0 = 1 means switch 1 is OFF and Po.1 = 1 means switch 2 is OFF. P0.0 = o means switch 1 is ON and p0.1 = o means switch 2 is ON. LED turns ON when P0.7 = 0 and turns OFF when P0.7 = 1

The program has two labels – READSW and NXT. It's all about reading switch values – that is P0.0 and P0.1. We are using RRC instruction to read switch values. The values of port 0 is moved to accumulator. Since port 0 and 1 are used to interface switches 1 and 2, we can get the values of both port bits in LSB"s 0 and 1 of accumulator by using MOV A,P0 instruction. RRC – means – rotate right through carry. You can learn more about this instruction here – **8051 programming tutorial 1 .** What RRC do is simple – it will move value of port 0.0 to the carry bit. Now we can check the carry bit using instruction JC – which means "jump if carry is set" . If carry is SET – then it means port0.0 =1 and this means switch 1 is OFF. If switch 1 is OFF then we have to check status of switch 2 and that is why we jump to label NXT.

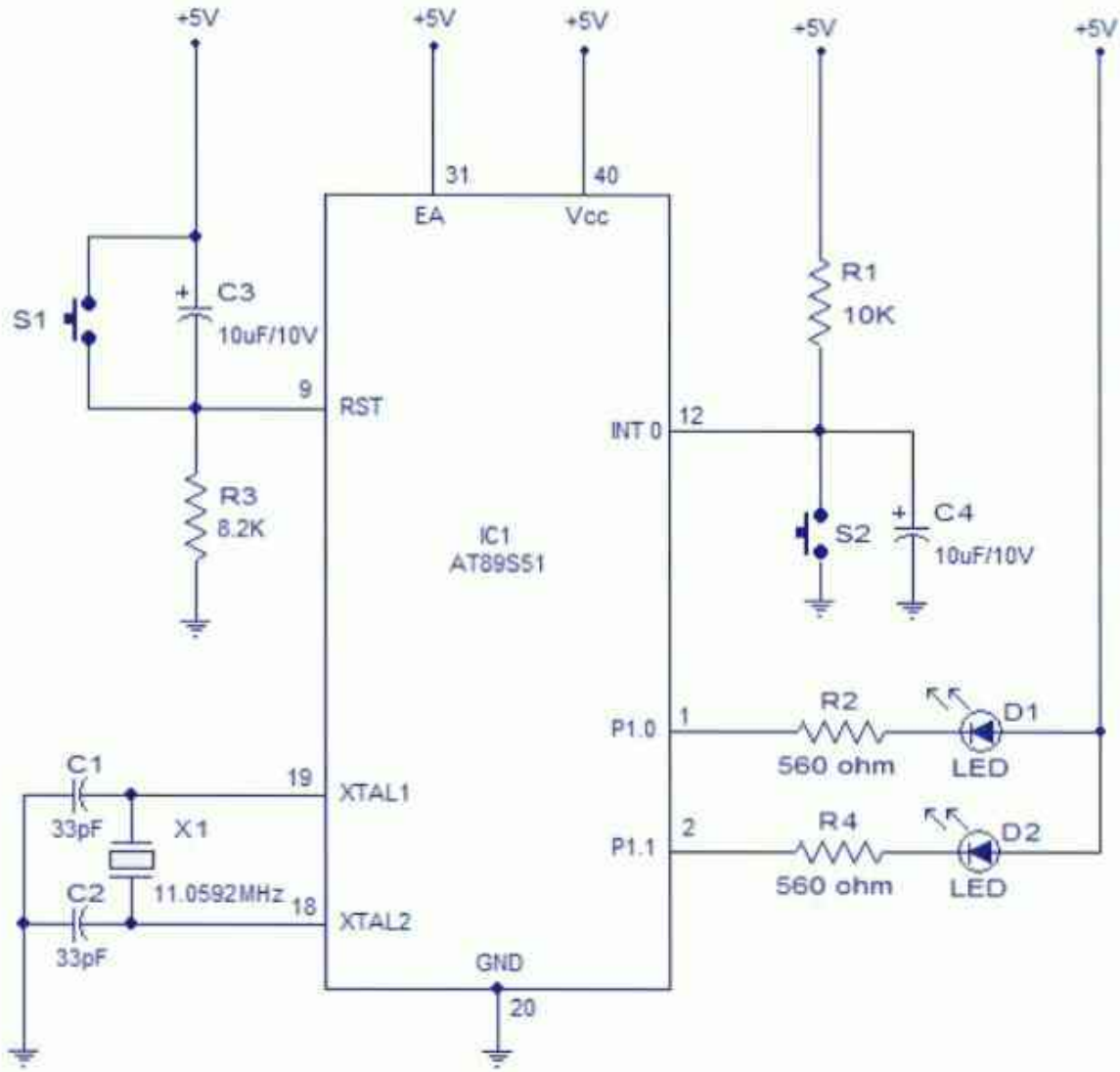
In the mean time if switch 1 is pressed – then value of port 0.0 will be equal to zero. This will get moved to accumulator and hence an RRC will result in carry bit = o. If carry bit = 0 then result of executing JC instruction is negative and it will not jump. The next instruction will get executed – that is CLR P0.7. This clears port 0.7 to zero and hence LED will turn ON. Once turned On- LED will be kept On until switch 2 is pressed.

The status of switch 2 is checked in NXT label. When NXT is executed, we are using RRC for the second time consecutively. This means, the carry bit now holds the value of P0.1 – which is status of switch 2. If carry bit = 1 then switch 2 is OFF. This means LED should not be turned OFF. If carry bit = 0 then LED should be turned OFF (The instruction SETB P0.7 turns LED OFF)

Toggling 2 LED with a pushbutton using interrupt.

This circuit demonstrates how to toggle two LEDs with a single push button using the external interrupts. Interrupt is an asynchronous signal (either hardware or software) which indicates the processor to make a change in current execution. When the processor receives a valid interrupt signal it saves the current state and then goes to execute a set of predefined steps called interrupt service routine (ISR). After executing ISR, the processor goes back to the point where it deviated and continues from there. To learn more about interrupts check this link. External interrupt handling in 8051.

Circuit diagram.



Toggling LED using 8051 with interrupt

In the circuit shown above D1, D2 (the LEDs to be toggled) are connected to P1.0 and P1.1 respectively. R2 and R4 limits the current through the LEDs. The push button switch S2 is connected to the INT0 pin where R1 is a pull up resistor and C4 is the debouncing capacitor. C3, R3 and S3 forms the reset circuitry. Capacitors C2, C2 and crystal X1 are related to the clock circuitry. When powered ON LED D1 will be OFF and and LED D2 will be ON. Whenever push button switch S2 is pressed it creates an interrupt and the software makes the status of P1.o and P1.1 to toggle which gets reflected in the LEDs.

Program.

```
 ORG 000H // starting address
SJMP LABEL //jumps to the LABEL
ORG 003H // starting address for the ISR(INT0)
ACALL ISR // calls the ISR (interrupt service routine)
RETI // returns from the interrupt
LABEL: MOV A,#10000000B // sets the initial stage of the LEDs
(D1 OFF & D2 ON)
MAIN: // main function that sets the interrupt parameters
SETB IP.0 // sets highest priority for the interrupt INT0
SETB TCON.0 // interrupt generated by a falling edge signal at
INT0 (pin12)
SETB IE.0 // enables the external interrupt
SETB IE.7 // enables the global interrupt control
SJMP MAIN // jumps back to the MAIN subroutine
ISR: // interrupt service routine
CPL A // complements the current value in accumulator A
MOV P1,A // moves the current accumulator value to port 1
RET // jumps to RETI
END
```

4x4 Matrix Keypad Interfacing with 8051 Microcontroller

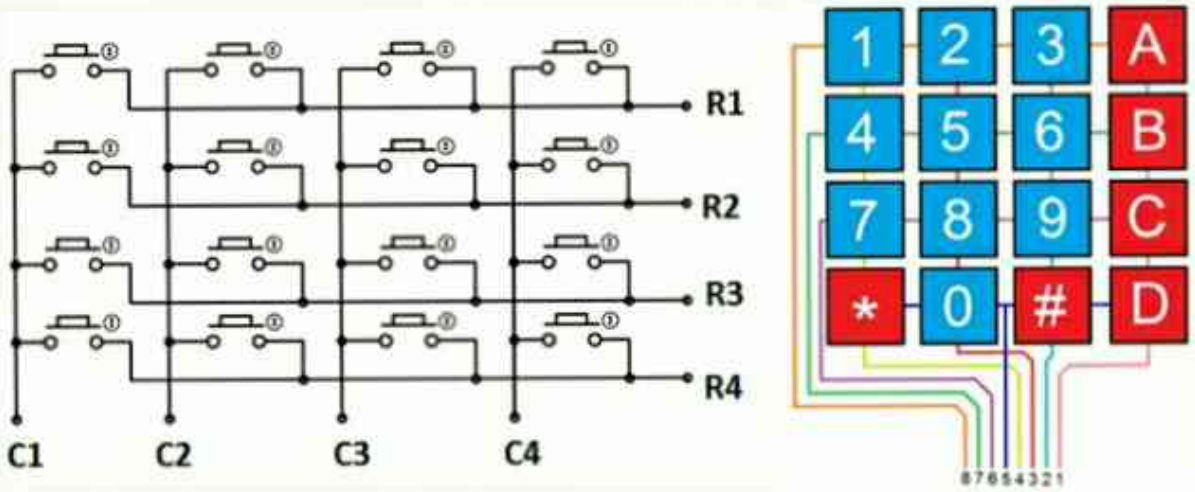
By Jayant | Jul 08, 2015

4x4 Matrix Keypad Interfacing with 8051 Microcontroller

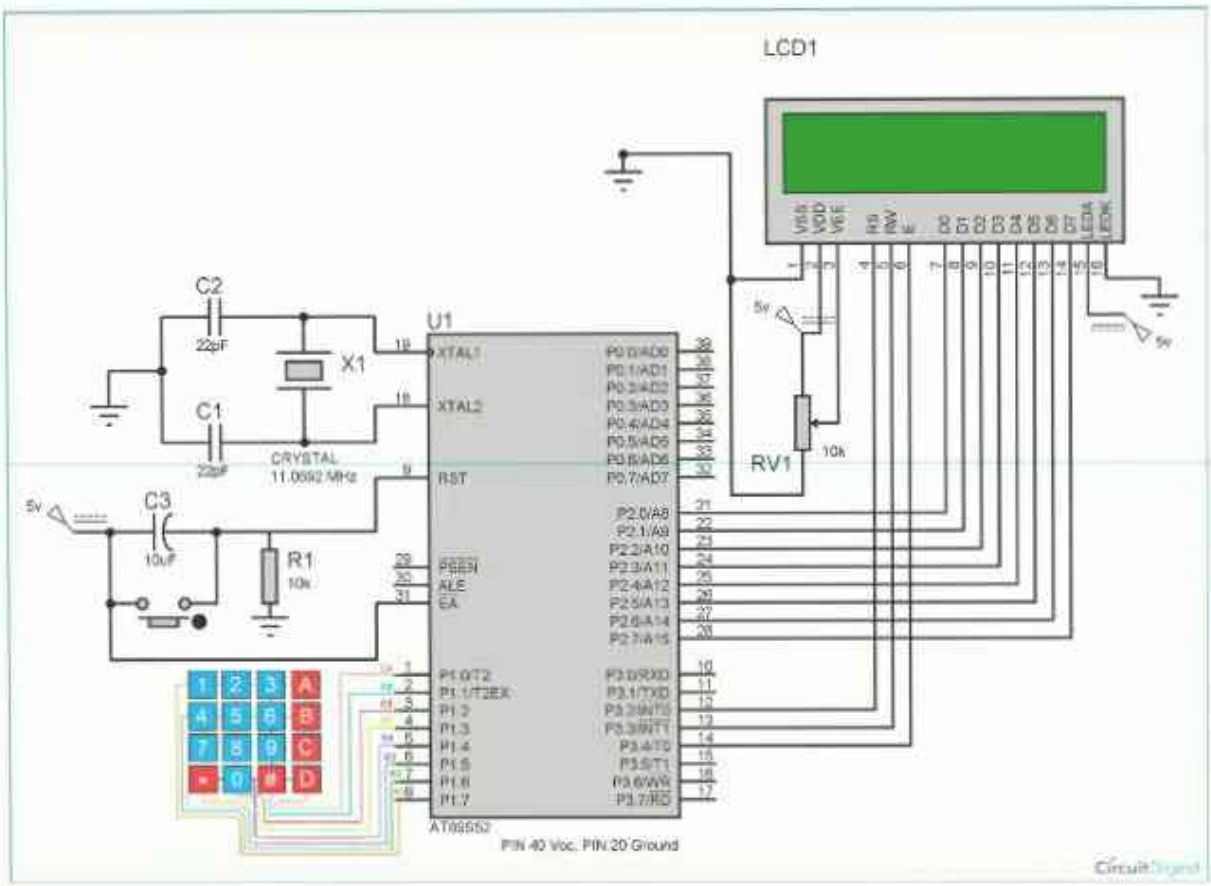
Keypads are widely used input devices being used in various electronics and embedded projects. They are used to take inputs in the form of numbers and albhabets, and feed the same into system for further processing. In this tutorial we are going to **interface a 4x4 matrix keypad with 8051 microcontroller**.

4X4 Matrix Keypad

Before we interface the keypad with microcontroller, first we need to understand how it works. Matrix keypad consists of set of Push buttons, which are interconnected. Like in our case we are using 4X4 matrix keypad, in which there are 4 push buttons in each of four rows. And the terminals of the push buttons are connected according to diagram. In first row, one terminal of all the 4 push buttons are connected together and another terminal of 4 push buttons are representing each of 4 columns, same goes for each row. So we are getting 8 terminals to connect with a microcontroller.



Interfacing keypad with 8051 microcontroller (AT89S52)



First we need to interface a LCD module to display the data which will be feed through KEYPAD, so please go through "LCD Interfacing with 8051 Microcontroller" article before interfacing KEYPAD.

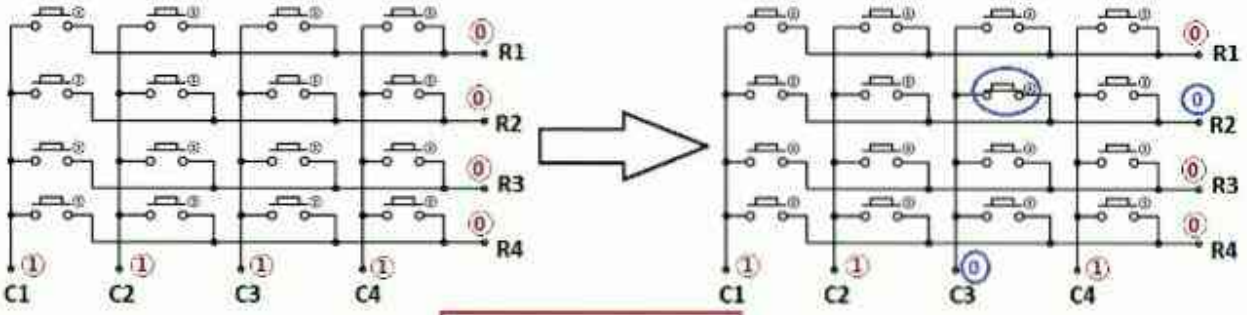
First we need to interface a LCD module to display the data which will be feed through KEYPAD, so please go through "LCD Interfacing with 8051 Microcontroller" article before interfacing KEYPAD.

As shown in above circuit diagram, to interface Keypad, we need to connect 8 terminals of the keypad to any port (8 pins) of the microcontroller. Like we have connected keypad terminals to Port 1 of 8051. Whenever any button is pressed we need to get the location of the button, means the corresponding ROW an COLUMN no. Once we get the location of the button, we can print the character accordingly.

Now the question is how to get the location of the pressed button? I am going to explain this in below steps and also want you to look at the code:

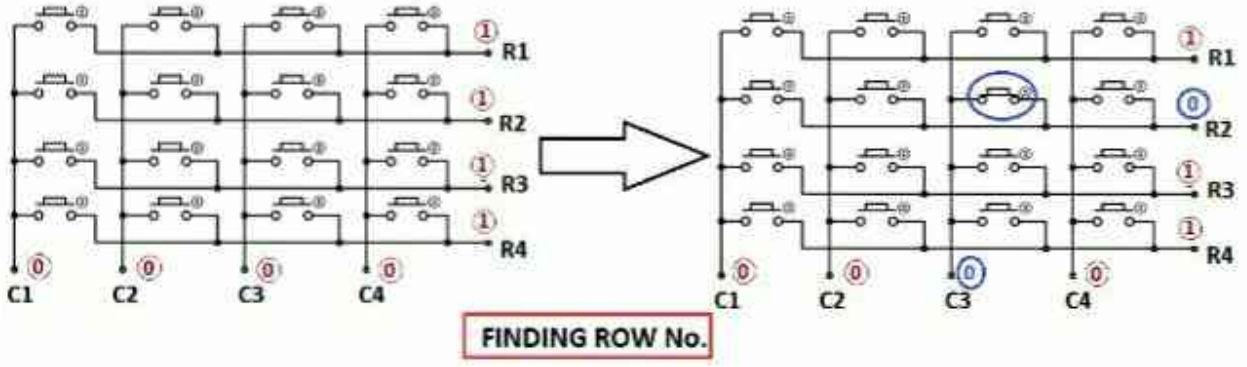
1. First we have made all the Rows to Logic level 0 and all the columns to Logic level 1.

2. Whenever we press a button, column and row corresponding to that button gets shorted and makes the corresponding column to logic level 0. Because that column becomes connected (shorted) to the row, which is at Logic level 0. So we get the column no. See main() function.



3. Now we need to find the Row no., so we have created four functions corresponding to each column. Like if any button of column one is pressed, we call function row_finder1(), to find the row no.

4. In row_finder1() function, we reversed the logic levels, means now all the Rows are 1 and columns are 0. Now Row of the pressed button should be 0 because it has become connected (shorted) to the column whose button is pressed, and all the columns are at 0 logic. So we have scanned all rows for 0.



5. So whenever we find the Row at logic 0, means that is the row of pressed button. So now we have column no (got in step 2) and row no., and we can print no. of that button using lcd_data function.

Same procedure follows for every button press, and we are using while(1), to continuously check, whether button is pressed or not.