



SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)



Microcontrollers and Embedded Systems

Unit III: Interfacing External Peripherals

Symbiosis Institute of Technology, Nagpur

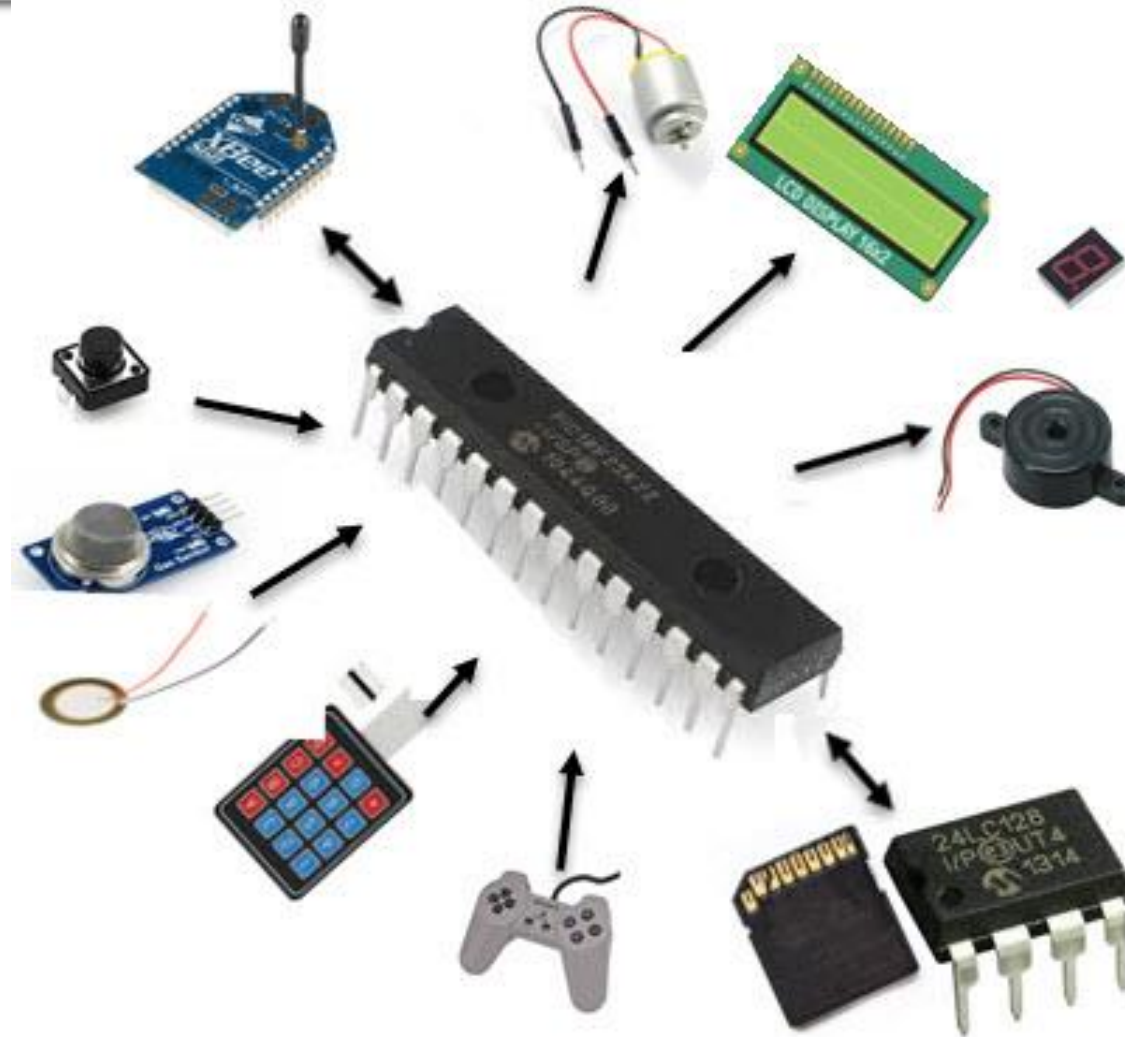
Symbiosis Institute of Technology, Nagpur Campus (SIT-N)



Interfacing External Peripherals:

Memory Interfacing RAM, and ROM, Interfacing and Assembly Language Programming of LED, Seven Segment Display, Liquid Crystal Display (LCD), DC Motor, and Stepper Motor.





8251 Interfacing Module



Memory Interfacing with 8051:

Overview

- The **8051 microcontroller** has limited internal memory:
 - **128 bytes of RAM** (for data storage and registers)
 - **4 KB of ROM** (for program storage)
- For larger programs and data storage, **external memory** (RAM/ROM) is interfaced using the **Address Bus, Data Bus, and Control Signals**.



Memory Classification

Internal Memory (On-chip)

1.RAM: 128 bytes, divided into:

- 1.32 general-purpose registers (R0-R7)**
- 2.Bit-addressable area (16 bytes, 128 bits)**
- 3.General-purpose memory (80 bytes)**

2.ROM: 4KB (stores program code, fixed at manufacture or via Flash memory)



Memory Interfacing with 8051:

External Memory (Off-chip)

- RAM (e.g., 62256 SRAM) for volatile data storage
- ROM (e.g., 27C512 EPROM) for storing program code permanently
- Uses **MOVX** instruction for accessing external memory



Addressing Modes for External Memory

➤ External Data Memory (RAM) Access

- MOVX A, @DPTR (Read from external RAM)
- MOVX @DPTR, A (Write to external RAM)

➤ External Program Memory (ROM) Access

- MOVC A, @A+DPTR (Fetch data from external ROM)



The **8051 microcontroller differentiates between RAM and ROM** using separate address spaces and distinct control signals for memory access.

1. Differentiation by Addressing Mode

- **ROM (Program Memory):** Accessed using the **PC (Program Counter)**, which fetches instructions from internal or external program memory.
- **RAM (Data Memory):** Accessed using **registers (R0, R1) or direct addressing**, managed by the **DPTR (Data Pointer) or indirect addressing modes**.



2. Control Signals for External Memory

- **PSEN (Program Store Enable, Pin 29):** Used to fetch code (ROM). Activated when executing instructions stored in external program memory.
- **RD (Read, Pin 17) & WR (Write, Pin 16):** Used for external RAM access. The microcontroller asserts RD/WR signals when reading or writing external data memory.



3. Internal vs External Memory Mapping

- **Internal ROM (0x0000 – 0x0FFF in typical 8051):** Accessed automatically for fetching instructions.
- **External ROM (Above 4KB, if enabled):** When using external ROM, the **EA (External Access)** pin is set low.
- **Internal RAM (0x00 – 0xFF):** Includes general-purpose RAM (0x00–0x7F) and SFRs (0x80–0xFF).
- **External RAM (Above 256 bytes):** Accessed via **MOVX** instruction.



System Design Example-

Design a system with:

- 1.8051 operating at 12 MHz
- 2.4KB EPROM using 2KB EPROM chips
- 3.8KB RAM using 4KB RAM chips



Memory Interfacing Components

Memory interfacing involves four types of lines:

- **Address Lines** – Selects memory location
- **Data Lines** – Transfers data between microcontroller & memory
- **Control Lines** – Signals for read/write operations
- **Chip Select** – Enables specific memory chips



4KB EPROM Interfacing

- . Uses **2 chips of 2KB EPROM**
- . **Address lines:** $2^{10} \times 2^1 = 2^{11} \rightarrow 11$ address lines required
- . **Data lines:** 8-bit (1 byte)
- . **Control Line:** **PSEN** (Program Store Enable) for reading ROM



8KB RAM Interfacing

- Uses **2 chips of 4KB RAM**
- **Address lines:** $2^{10} \times 2^2 = 2^{12} \rightarrow 12$ address lines required
- **Data lines:** 8-bit (1 byte)
- **Control Lines:** **RD** (Read) and **WR** (Write) for RAM access



System Design Example- Memory Mapping

8051 has 16 Address Lines.

Internal ROM Address : 0000H – 0FFFH, cannot be used for external address. Hence we have to use next available address.

Memory IC	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
EPROM 1 2KB	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH
EPROM 2 2KB	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1800H
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH
RAM 1 4KB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH
RAM 2 4KB	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Can be used for Chip Select

External ROM Address starts: 1000H

2KB EPROM has 11 Address Lines (A0-A10) so end ending address for 2KB EPROM1 is 17FFH.

Hence, starting address of EPROM2 is 1800H and ending address is 1FFFH



System Design Example- Memory Mapping

8051 has 16 Address Lines.

Internal RAM (0x00 – 0xFF): Includes general-purpose RAM (0x00–0x7F) and SFRs (0x80–0xFF)

Memory IC	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
EPROM 1 2KB	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH
EPROM 2 2KB	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1800H
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH
RAM 1 4KB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH
RAM 2 4KB	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Can be used for Chip Select

External ROM Address starts: 0000H.

We can have overlapping of the address in case of RAM, as we specify external read and write operation with RD and WR pins.



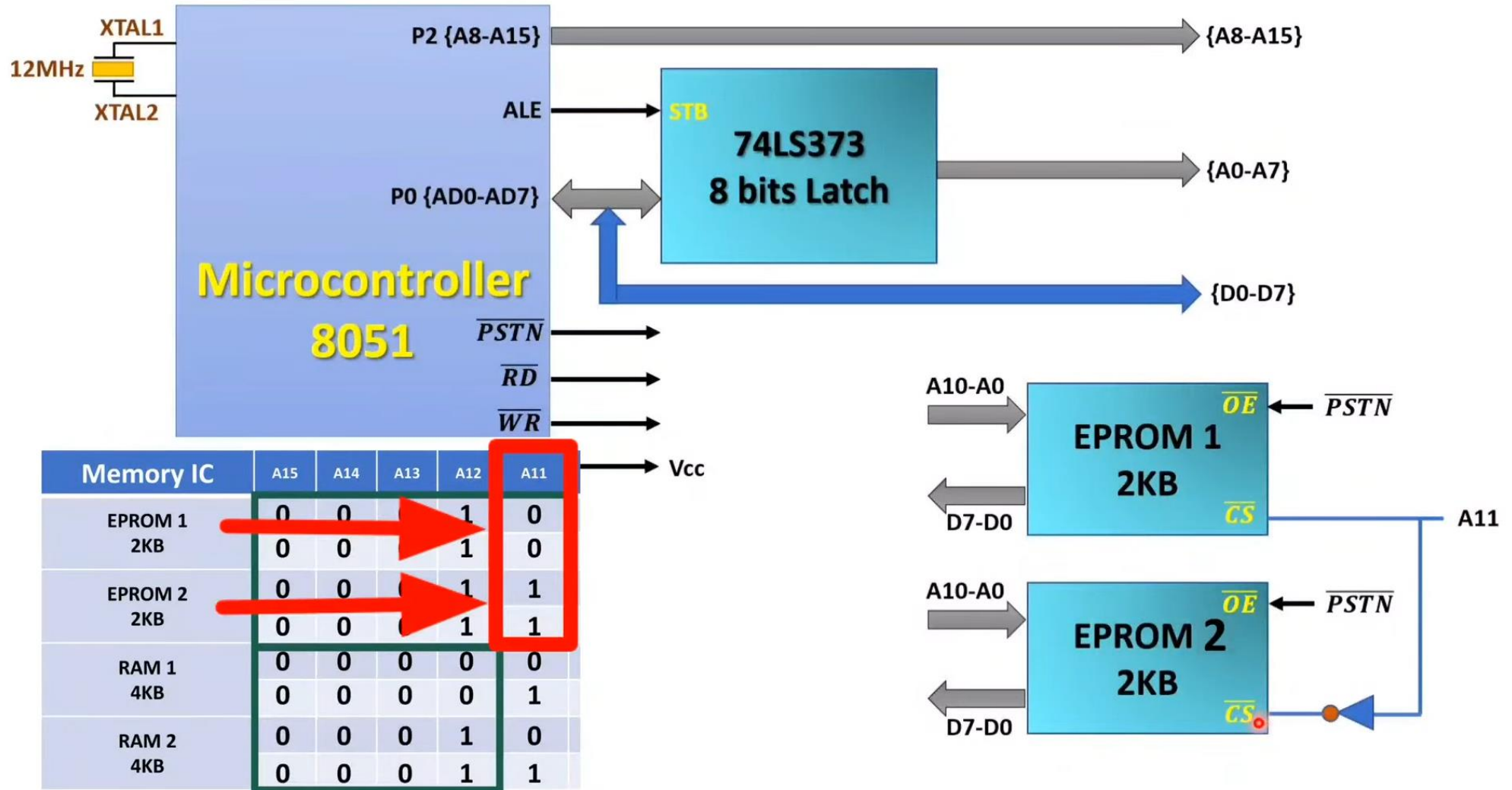
Key Takeaways

- **Memory addressing** is crucial for interfacing external ROM & RAM.
- **EPROM** needs **PSEN** for reading program memory.
- **RAM** needs **RD & WR** signals for data transfer.
- **Proper chip selection** is needed for accurate memory mapping.



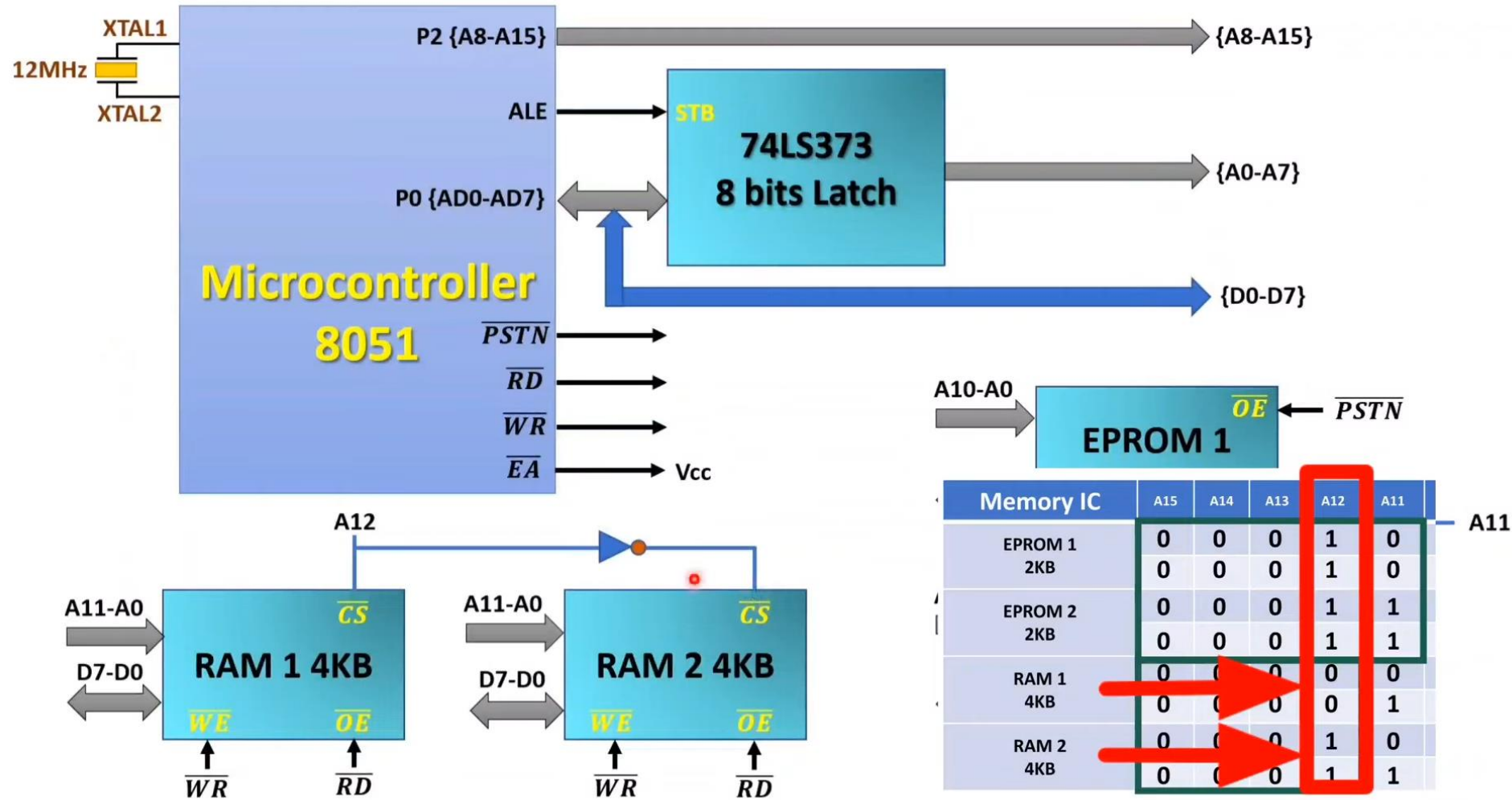
Circuit of Memory Interfacing in 8051

Interfacing with EPROM



Circuit of Memory Interfacing in 8051

Interfacing with RAM



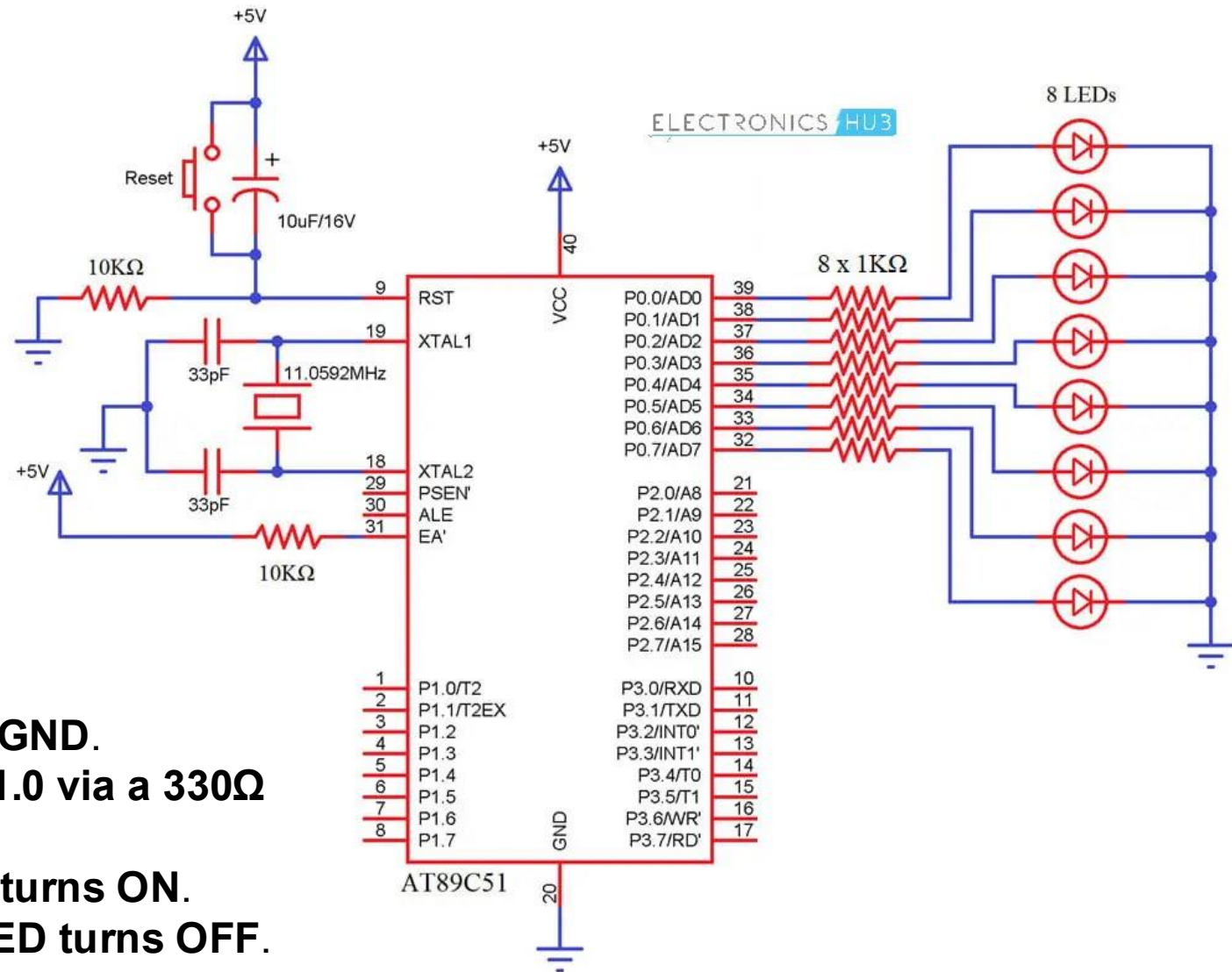
Circuit Diagram: LED Interfacing with 8051 Components:

- 8051 microcontroller
- LED (Light Emitting Diode)
- Resistor (330Ω or $1K\Omega$)
- Power Supply (5V)



Interfacing of LED with 8051

Circuit Diagram: LED Interfacing with 8051 Components.



- The **cathode** of the **LED** is connected to **GND**.
- The **anode** of the **LED** is connected to **P1.0** via a **330Ω resistor**.
- When **P0 = 0**, current flows, and the **LED turns ON**.
- When **P0 = 1**, no current flows, and the **LED turns OFF**.

Interfacing of LED with 8051

Controlling LED Using Assembly Language: Basic LED Blinking Program (8051 Assembly)

ORG 0000H

START:

MOV P1, #00H ; Turn ON all LEDs (Active Low)

CALL DELAY

MOV P0, #0FFH ; Turn OFF all LEDs

CALL DELAY

SJMP START ; Repeat forever

DELAY:

MOV R7, #255

DELAY_LOOP:

DJNZ R7, DELAY_LOOP

RET

END



Interfacing of LED with 8051

Controlling LED Using Assembly Language: Basic LED Blinking Program (8051 Assembly)

ORG 0000H

START:

```
MOV P1, #00H ; Turn ON all LEDs (Active Low)
CALL DELAY
MOV P0, #0FFH ; Turn OFF all LEDs
CALL DELAY
SJMP START ; Repeat forever
```

DELAY:

```
MOV R7, #255
```

DELAY_LOOP:

```
DJNZ R7, DELAY_LOOP
RET
```

END

Explanation:

1. MOV P1, #0FFH → Initialize Port 1 as **all HIGH** (LEDs OFF).
2. CLR P1.0 → Clear **P1.0**, making it LOW, which **turns ON the LED**.
3. CALL DELAY → Wait for some time (delay subroutine).
4. SETB P1.0 → Set **P1.0 HIGH**, which **turns OFF the LED**.
5. CALL DELAY → Wait again.
6. SJMP AGAIN → Repeat the blinking process indefinitely.



Interfacing of LED with 8051

Controlling LED Using Assembly Language: Basic LED Blinking Program (8051 Assembly)

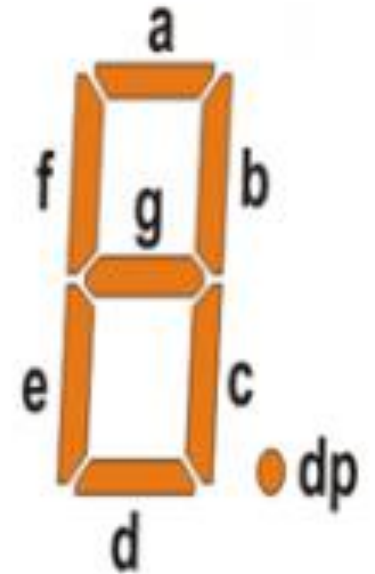
Key Takeaways

- **LEDs are interfaced with 8051 using output ports (P1, P2, etc.).**
- **Active LOW configuration** means LEDs turn ON when port output is LOW.
- **Resistors (330Ω or $1K\Omega$) protect LEDs from excessive current.**
- **Assembly code uses CLR/SETB instructions to control LED states.**
- **Multiple LEDs can be controlled using 8-bit port manipulation.**



Interfacing of 7 Segment Display with 8051

The name 7 segments imply there are 7 LED segments arranged as shown in figure 1. After LEDs, these are the easiest interfaces to a microcontroller. There is also a decimal point or dp. It is used when decimal digits like 5.1 etc are displayed.

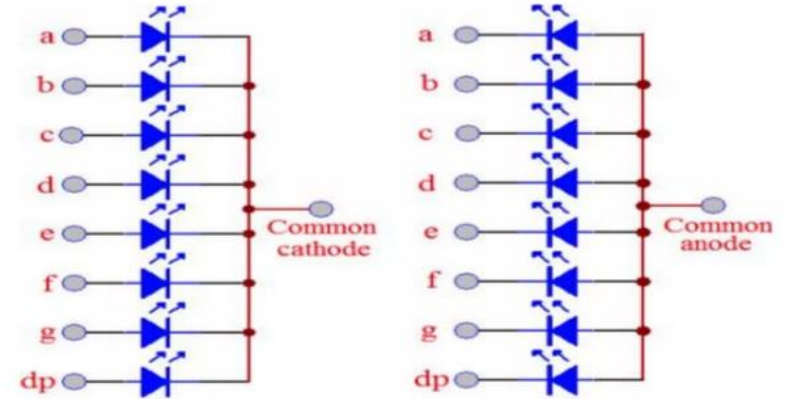


Interfacing of 7 Segment Display with 8051

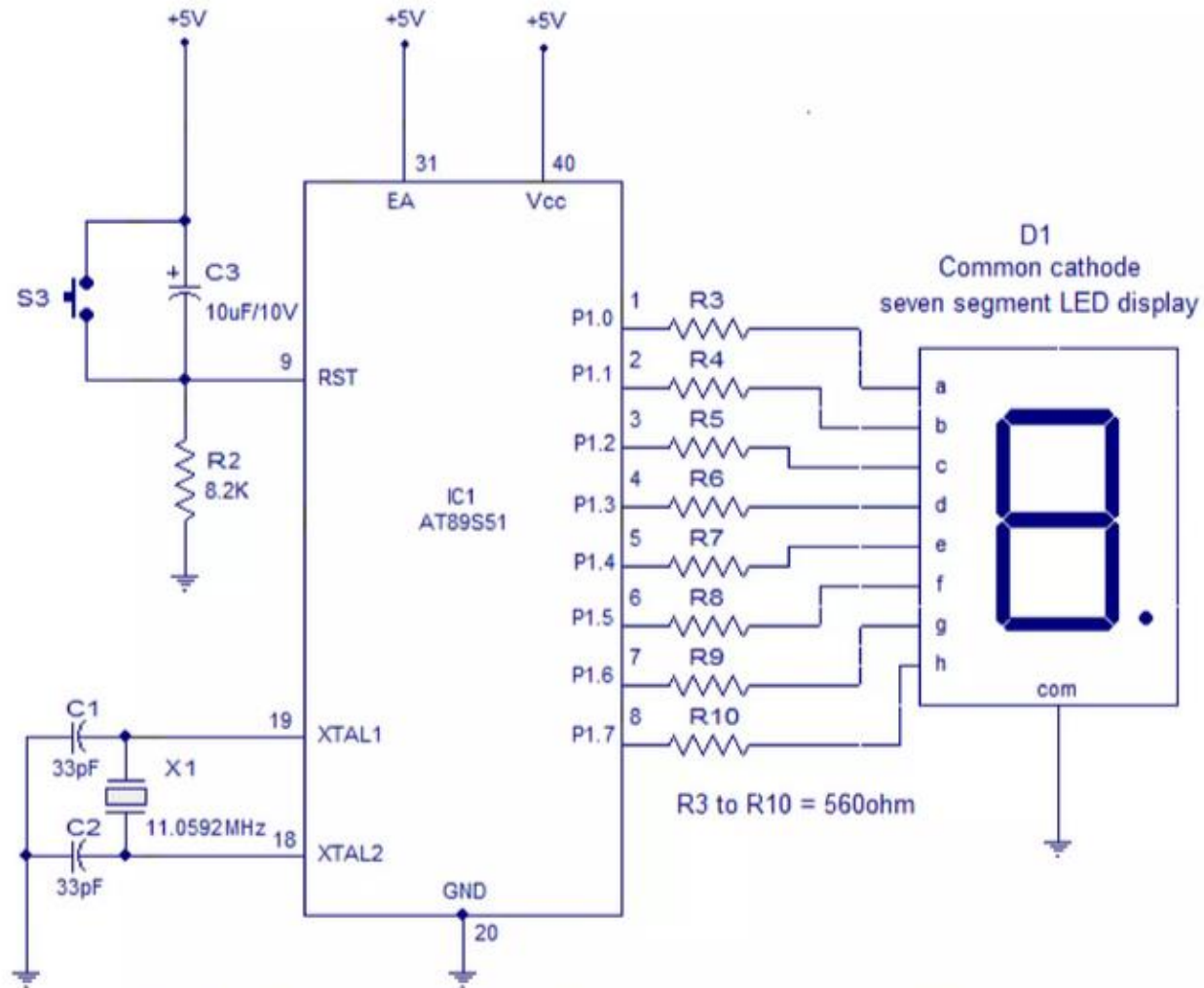
There are two main constructions of the seven-segment display modules

Common Cathode: In this type of segments all the cathode terminals are made common and tied to GND. Thus the segments a to g needs a logic High signal(5v) in order to glow.

Common Anode: In this type of segments all the anodes terminals are made common and tied to VCC(5v). Thus the segments a to g needs a logic LOW signal(GND) in order to glow.



Interfacing of 7 Segment Display with 8051



Interfacing 7 segment display to 8051

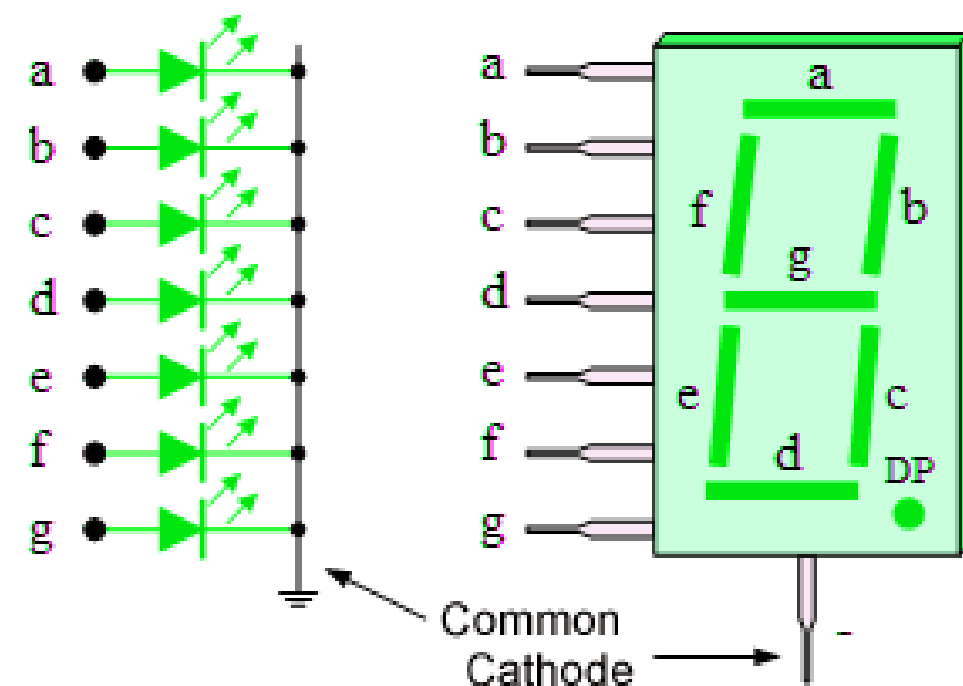
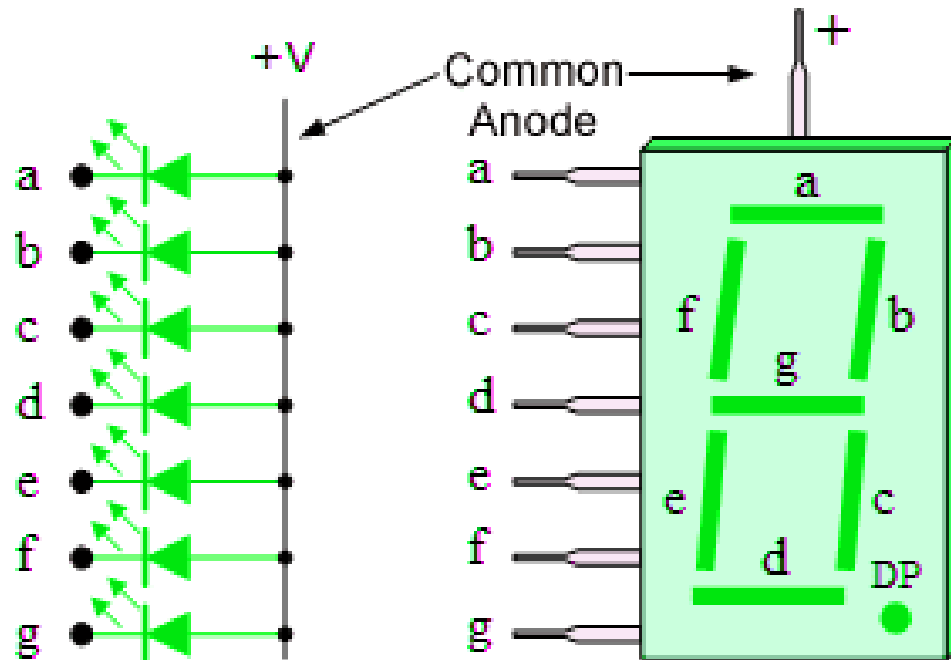
www.circuitstoday.com



Interfacing of 7 Segment Display with 8051

The data (bitmap) has to be sent to the 7 segment display from the controller through the data bus. So we have to have a pre determined set of bitmaps for the numbers that we would be using.

In any kind of application, it's good practice to have a Look-up-Table for the data to be displayed on the 7 segment. The data to be displayed has to be framed separately for common cathode and common anode type displays.



Interfacing of 7 Segment Display with 8051

Common Anode Configuration:

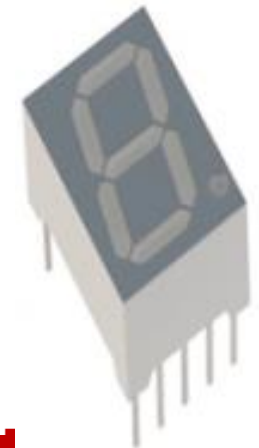
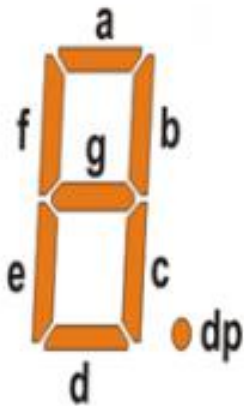
Number	g f e d c b a	Hex code
0	1000000	C0
1	1111001	F9
2	0100100	A4
3	0110000	B0
4	0011001	99
5	0010010	92
6	0000010	82
7	1111000	F8
8	0000000	80
9	0010000	90



Interfacing of 7 Segment Display with 8051

Common Cathode Configuration:

Number	g f e d c b a	Hex Code
0	0111111	3F
1	0000110	06
2	1011011	5B
3	1001111	4F
4	1100110	66
5	1101101	6D
	1111101	7D
	0000111	07
	1111111	7F
	1001111	4F



Interfacing of 7 Segment Display with 8051

Common Anode Configuration:

ORG 0000H

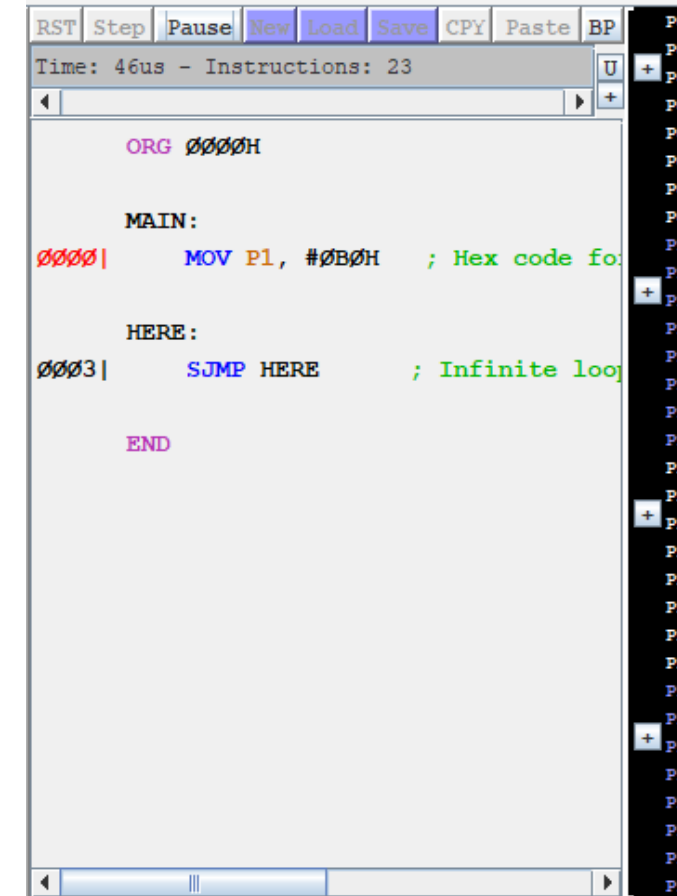
MAIN:

MOV P1, #0B0H ; Hex code for '3' in Common Anode 7-segment display

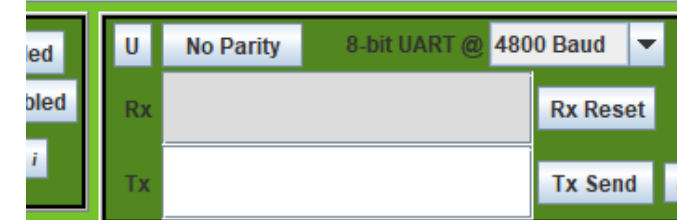
HERE:

SJMP HERE ; Infinite loop to keep displaying '3'

END



```
RST Step Pause New Load Save CPY Paste BP
Time: 46us - Instructions: 23
ORG 0000H
MAIN:
0000| MOV P1, #0B0H ; Hex code for
HERE:
0003| SJMP HERE ; Infinite loop
END
```



Interfacing of 7 Segment Display with 8051

Common Anode Configuration: *Try this for all four segments.*

; This program multiplexes the number 1234
; on the four 7-segment displays.
; Note: a logic 0 lights a display segment.

start:

```
SETB P3.3  
SETB P3.4  
MOV P1, #11111001B  
CALL delay
```

```
CLR P3.3  
MOV P1, #10100100B  
CALL delay
```

```
CLR P3.4  
SETB P3.3  
MOV P1, #10110000B  
CALL delay
```

```
CLR P3.3  
MOV P1, #10011001B  
CALL delay  
JMP start
```

```
; a crude delay  
delay:  
MOV R0, #200  
DJNZ R0, $  
RET
```



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 crystal display.

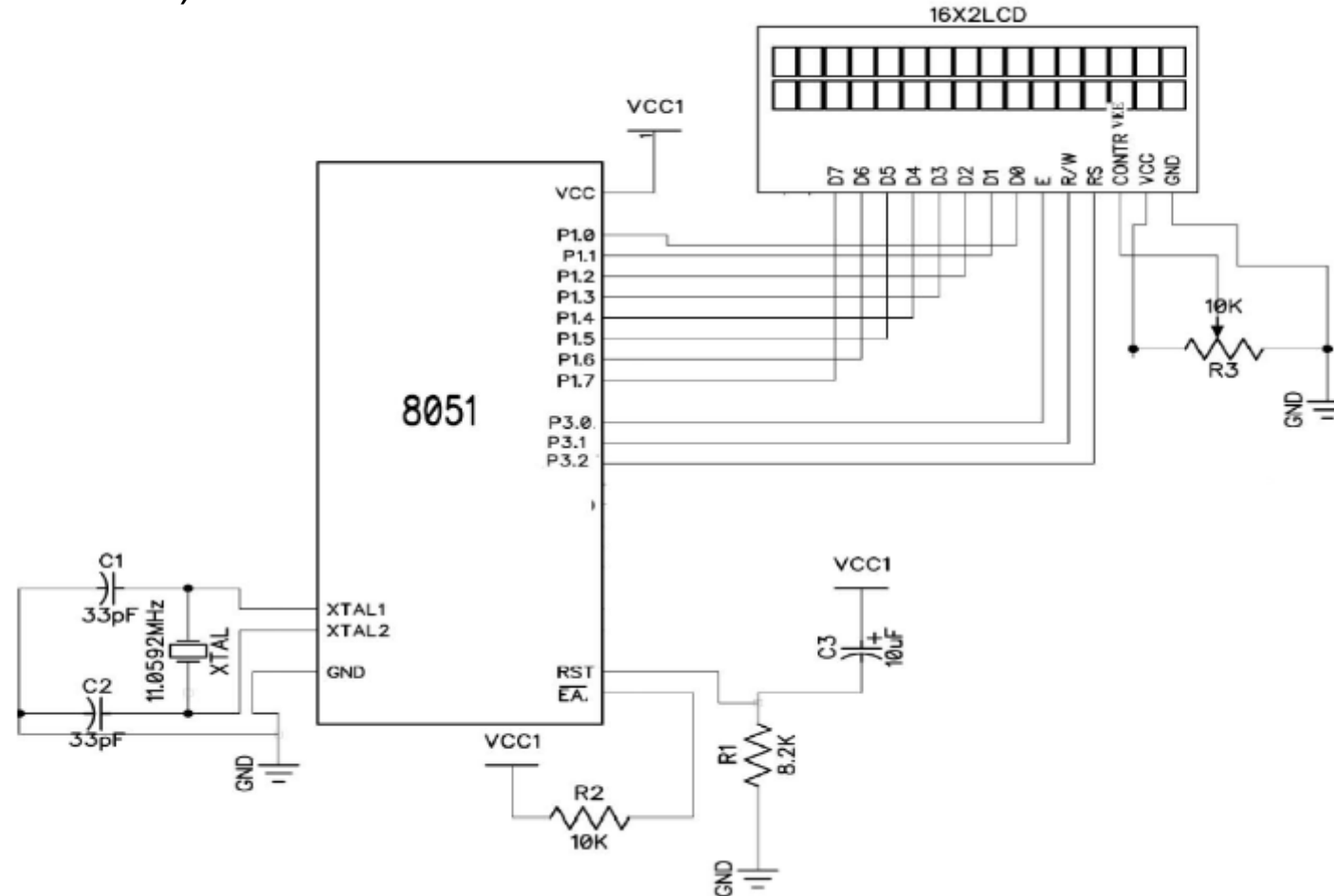
It displays all the letters of alphabet, Greek letters, punctuation marks, mathematical symbols etc. In addition, it is possible to display symbols made up by the user.

Other useful features include automatic message shift (left and right), cursor appearance, LED backlight etc



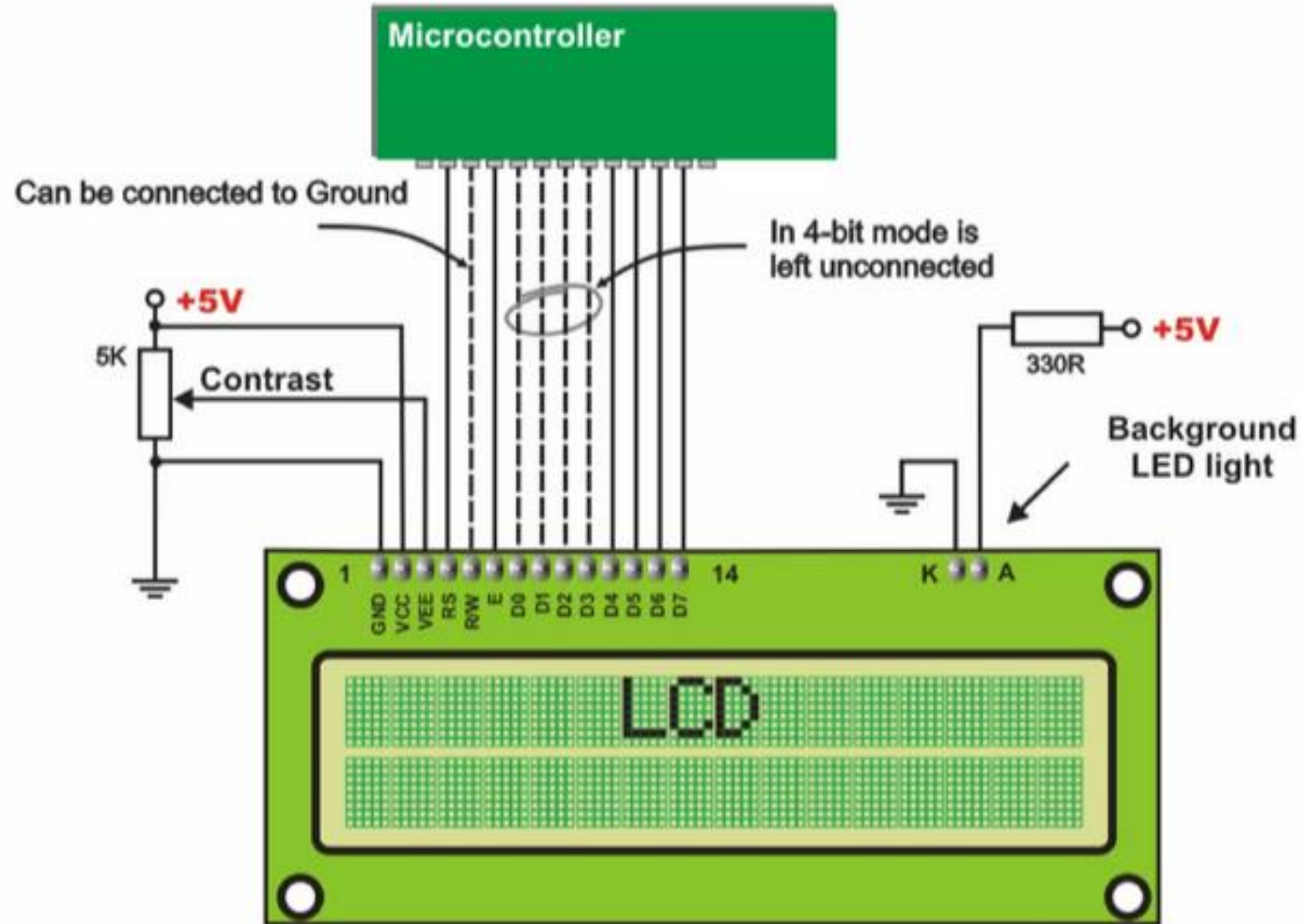
Interfacing of LCD Display with 8051

LCD Interfacing (8-bit Mode)



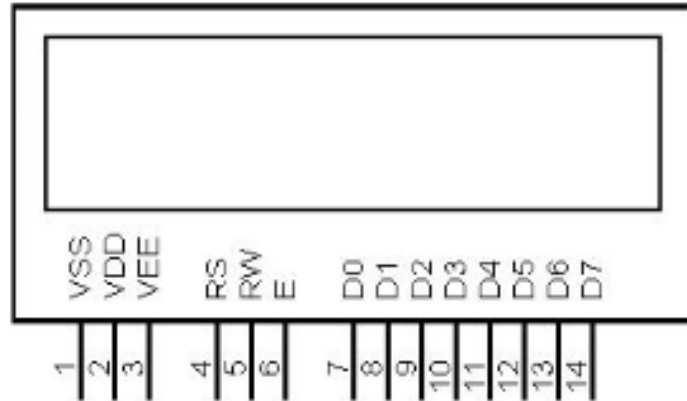
Interfacing of LCD Display with 8051

LCD Interfacing (4-bit Mode)



Interfacing of LCD Display with 8051

Interfacing of LCD with 8051



Pin No:	Name	Function
1	VSS	This pin must be connected to the ground
2	VCC	Positive supply voltage pin (5V DC)
3	VEE	Contrast adjustment
4	RS	Register selection RS=1 selects data register RS=0 Selects Command Register
5	R/W	Read or write To read from reg R/W=1 to write on reg. R/W=0
6	E	Enable High to low pulse enables command or data reg.
7-14	D0-D7	LCD Data Lines
15	LED+	Back light LED+
16	LED-	Back light LED-

Table 3.2 LCD Pins



Interfacing of LCD Display with 8051

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



8-bit Mode

How It Works?

- The **microcontroller sends 8-bit data in one cycle.**
- Requires **8 data pins (DB0–DB7)**, plus **RS, RW, E** → **Total 11 GPIO pins**
- Faster communication (one operation per command or character)

Timing of an 8-bit Command

- 1.Set RS (0 = Command, 1 = Data)
- 2.Set RW (0 = Write, 1 = Read)
- 3.Put 8-bit command on DB0–DB7
- 4.Generate Enable (E) pulse (HIGH → LOW)



Advantages of 8-bit Mode

- **Faster execution** (one transfer per character).
- **Simpler implementation** (no need to split data into nibbles)

Disadvantages of 8-bit Mode

- **Requires more GPIO pins** (11 pins needed).
- **Not always feasible for microcontrollers with fewer I/O pins**



Interfacing of LCD Display with 8051

Example: Writing 'A' (ASCII 0x41) in 8-bit Mode

Step	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
1	1	0	0	1	0	0	0	0	0	1	1
2	1	0	0	1	0	0	0	0	0	1	0



4-bit Mode

How It Works

- Uses **only 4 data lines** (DB4–DB7), reducing the number of required GPIO pins.
- Data is **split into two nibbles** (upper 4 bits first, lower 4 bits second).
- Requires **RS, RW, E, and DB4–DB7** → Total **7 GPIO pins**

Timing of a 4-bit Command

1. Set RS (0 = Command, 1 = Data)
2. Set RW (0 = Write, 1 = Read)
3. Send HIGH nibble (DB4–DB7)
4. Generate Enable (E) pulse
5. Send LOW nibble (DB4–DB7)
6. Generate Enable (E) pulse



Initialization of LCD in 4-bit Mode

Since LCD **defaults to 8-bit mode** at power-on, we need to initialize it to **4-bit mode**:

1. Send **Function Set** command twice (using only DB4–DB7).
2. Configure **display settings** (cursor, blinking, etc.).
3. Set **entry mode** (increment cursor, no display shift).

Example Function Set Command (0x28 = 4-bit mode, 2-line, 5x8 font)

- First send **high nibble (0x2)**
- Then send **low nibble (0x8)**



Interfacing of LCD Display with 8051

Example: Writing 'A' (ASCII 0x41) in 4-bit Mode

Step	RS	RW	DB7	DB6	DB5	DB4	E
1	1	0	0	1	0	0	1
2	1	0	0	1	0	0	0
3	1	0	0	0	0	1	1
4	1	0	0	0	0	1	0



Advantages of 4-bit Mode

- **Saves GPIO pins** (only 7 needed instead of 11)
- **Allows using microcontrollers with fewer I/O ports**

Disadvantages of 4-bit Mode

- **Slower than 8-bit mode** (each command needs two operations)
- **More complex implementation** (splitting data into nibbles)



Interfacing of LCD Display with 8051

Comparison Table

Feature	8-bit Mode	4-bit Mode
Data Lines	DB0–DB7	DB4–DB7
GPIO Pins	11	7
Speed	Faster (1 write)	Slower (2 writes)
Ease of Use	Easier	More complex
Suitability	When many GPIOs are available	When GPIOs are limited



Interfacing of LCD Display with 8051 (Sample Program)

Explanation of the Assembly Code for 8051 with HD44780 LCD (4-bit Mode):

This program initializes the LCD in **4-bit mode** and sends the characters "ABC" stored in RAM to the display. Below is a breakdown of the code sections:

1. Store Data in RAM

```
MOV 30H, #'A' ; Store 'A' at RAM location 30H
MOV 31H, #'B' ; Store 'B' at RAM location 31H
MOV 32H, #'C' ; Store 'C' at RAM location 32H
MOV 33H, #0   ; End of data marker (NULL character)
```



2. LCD Initialization (Function Set, Entry Mode, Display ON/OFF)

(a) Function Set - Switch LCD to 4-bit Mode

CLR P1.3 ; RS = 0 (Sending Command)

CLR P1.7 ; D7 = 0

CLR P1.6 ; D6 = 0

SETB P1.5 ; D5 = 1

CLR P1.4 ; D4 = 0

- *The LCD defaults to 8-bit mode at power-on.*
- *We force it into 4-bit mode by sending two high nibbles (0x20).*
- The **Function Set** command is represented as follow: **0 0 1 D L N F 0 0**

SETB P1.2 ; E = 1 (Enable pulse)

CLR P1.2 ; E = 0 (Negative Edge Trigger)

CALL delay ; Wait for busy flag to clear

SETB P1.2 ; E = 1

CLR P1.2 ; E = 0



Interfacing of LCD Display with 8051 (Sample Program)

2. LCD Initialization (Function Set, Entry Mode, Display ON/OFF)

(a) Function Set - Switch LCD to 4-bit Mode

SETB P1.7 ; D7 = 1

CLR P1.5 ; D5 = 0

SETB P1.2 ; E = 1

CLR P1.2 ; E = 0

Then, low nibble (0x28) is sent to confirm 4-bit mode, 2-line display, 5x8 font.

Binary (DL N F)	Hex Code	Mode
0011 0000	0x30	8-bit, 1-line, 5x8 font
0011 0100	0x34	8-bit, 1-line, 5x10 font
0011 1000	0x38	8-bit, 2-line, 5x8 font
0010 0000	0x20	4-bit, 1-line, 5x8 font
0010 0100	0x24	4-bit, 1-line, 5x10 font
0010 1000	0x28	4-bit, 2-line, 5x8 font



2. LCD Initialization (Function Set, Entry Mode, Display ON/OFF) (b) Entry Mode Set (Cursor Movement)

```
CLR P1.7 ; D7 = 0  
CLR P1.6 ; D6 = 0  
CLR P1.5 ; D5 = 0  
CLR P1.4 ; D4 = 0 (High Nibble: `0x00`)
```

```
SETB P1.2 ; E = 1  
CLR P1.2 ; E = 0
```

```
SETB P1.6 ; D6 = 1  
SETB P1.5 ; D5 = 1  
SETB P1.4 ; D4 = 1 (Low Nibble: `0x06`)
```

```
SETB P1.2 ; E = 1  
CLR P1.2 ; E = 0
```

***0x06 Entry Mode Set:**
Cursor moves **right** after
each character, no display
shift.*



3. Send Data (Characters "ABC") to LCD

```
SETB P1.3 ; RS = 1 (Sending Data)
MOV R1, #30H ; Point to first character in RAM
```

```
loop:
MOV A, @R1 ; Load character from RAM
JZ finish ; If NULL (0), stop sending data
CALL sendCharacter ; Send to LCD
INC R1 ; Move to next character
JMP loop ; Repeat
```

- *MOV A, @R1* → Load character from RAM
- *JZ finish* → If NULL (0), stop
- *CALL sendCharacter* → Send character



4. Function: Send a Character to LCD

```
sendCharacter:
MOV C, ACC.7
MOV P1.7, C
MOV C, ACC.6
MOV P1.6, C
MOV C, ACC.5
MOV P1.5, C
MOV C, ACC.4
MOV P1.4, C ; High nibble set
```

```
SETB P1.2
CLR P1.2 ; Send High Nibble
```

```
MOV C, ACC.3
MOV P1.7, C
MOV C, ACC.2
MOV P1.6, C
MOV C, ACC.1
MOV P1.5, C
MOV C, ACC.0
MOV P1.4, C ; Low nibble set
```

```
SETB P1.2
CLR P1.2 ; Send Low Nibble
```

```
CALL delay ; Wait for busy flag
RET
```

•Splits ASCII character into **High Nibble (D7-D4)** and **Low Nibble (D3-D0)**
•Sends **High Nibble first**, then **Low Nibble**



5. Delay Function

delay:

```
MOV R0, #50  
DJNZ R0, $  
RET
```

*A simple delay loop to wait for
LCD busy flag to clear*

Final Execution Order:

1. Initialize LCD in 4-bit mode
2. Set Entry Mode (Cursor moves right, no shift)
3. Turn on Display (Cursor & Blinking ON)
4. Send characters 'A', 'B', 'C' to LCD



Interfacing of DC Motor with 8051

DC motor converts electrical energy in the form of Direct Current into mechanical energy.

- In the case of the motor, the mechanical energy produced is in the form of a rotational movement of the motor shaft.
- The direction of rotation of the shaft of the motor can be reversed by reversing the direction of Direct Current through the motor.
- The motor can be rotated at a certain speed by applying a fixed voltage to it. If the voltage varies, the speed of the motor varies.
- Thus, the DC motor speed can be controlled by applying varying DC voltage; whereas the direction of rotation of the motor can be changed by reversing the direction of current through it.
- For applying varying voltage, we can make use of the PWM technique.
- For reversing the current, we can make use of an H-Bridge circuit or motor driver ICs that employ the H-Bridge technique or any other mechanisms.



Interfacing of DC Motor with 8051

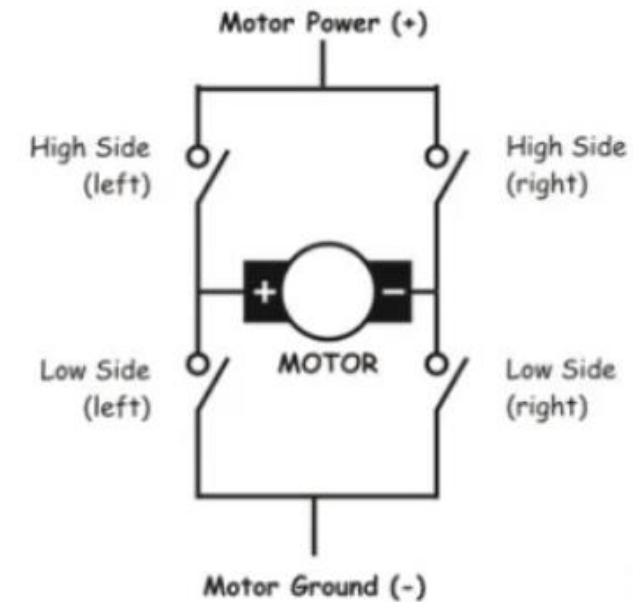
H-Bridge Circuit

The name “H-Bridge” comes from the shape of the switching circuit, which controls the motion of the motor. It is also known as “Full Bridge”. This circuit has four switching elements. We can make H-bridges with the help of transistors, MOSFETs, etc. It will be cheap, but they will increase the size of the design and the circuit board, which is mostly not preferable, so we can use a small 16-pin IC for this purpose.

Working of H-bridge Circuit

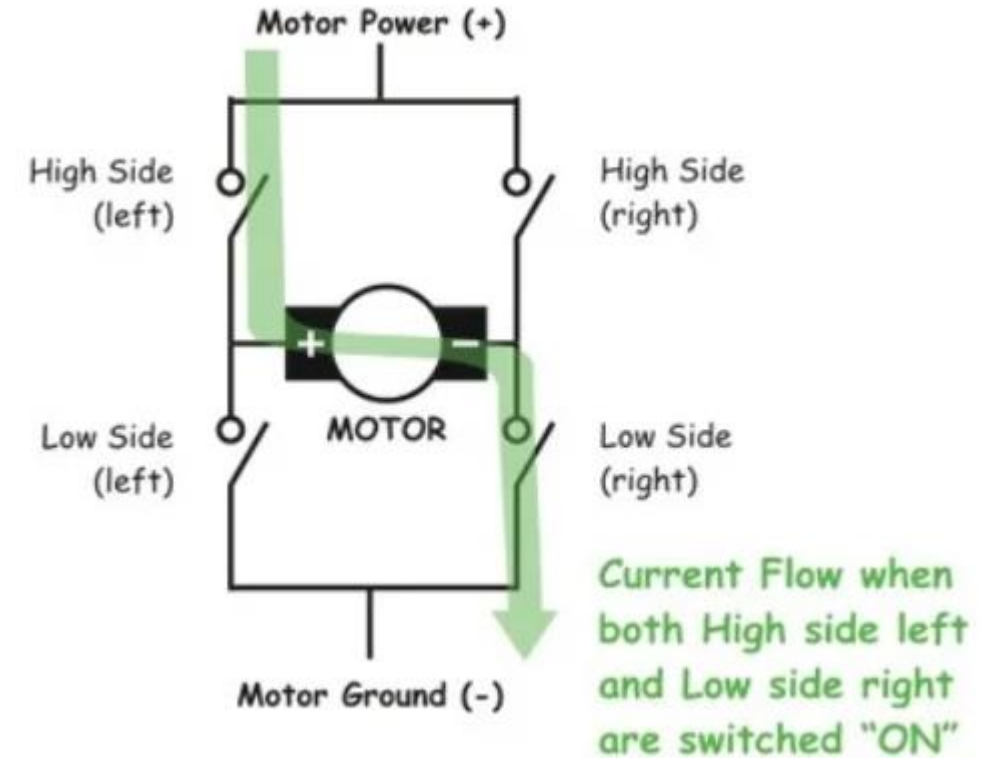
As we can see in the figure, there are four switching elements named:

- High side left
- High side right
- Low side right
- Low side left



Interfacing of DC Motor with 8051

When these switches are turned on in pairs, the motor changes its direction accordingly. If we switch on “**High side left**” and “**Low side right**”, then the motor will rotate in a forward direction as current from the power supply flows through the motor coil and goes to ground through the switch on the low side (right). This is shown in the figure below. Similarly, when we switch on the low side (left) and the high side (right), the current flows in the opposite direction and the motor rotates in the opposite direction. This is the basic working of H-Bridge. We can also make a small truth table according to the switching of H-Bridge explained above.

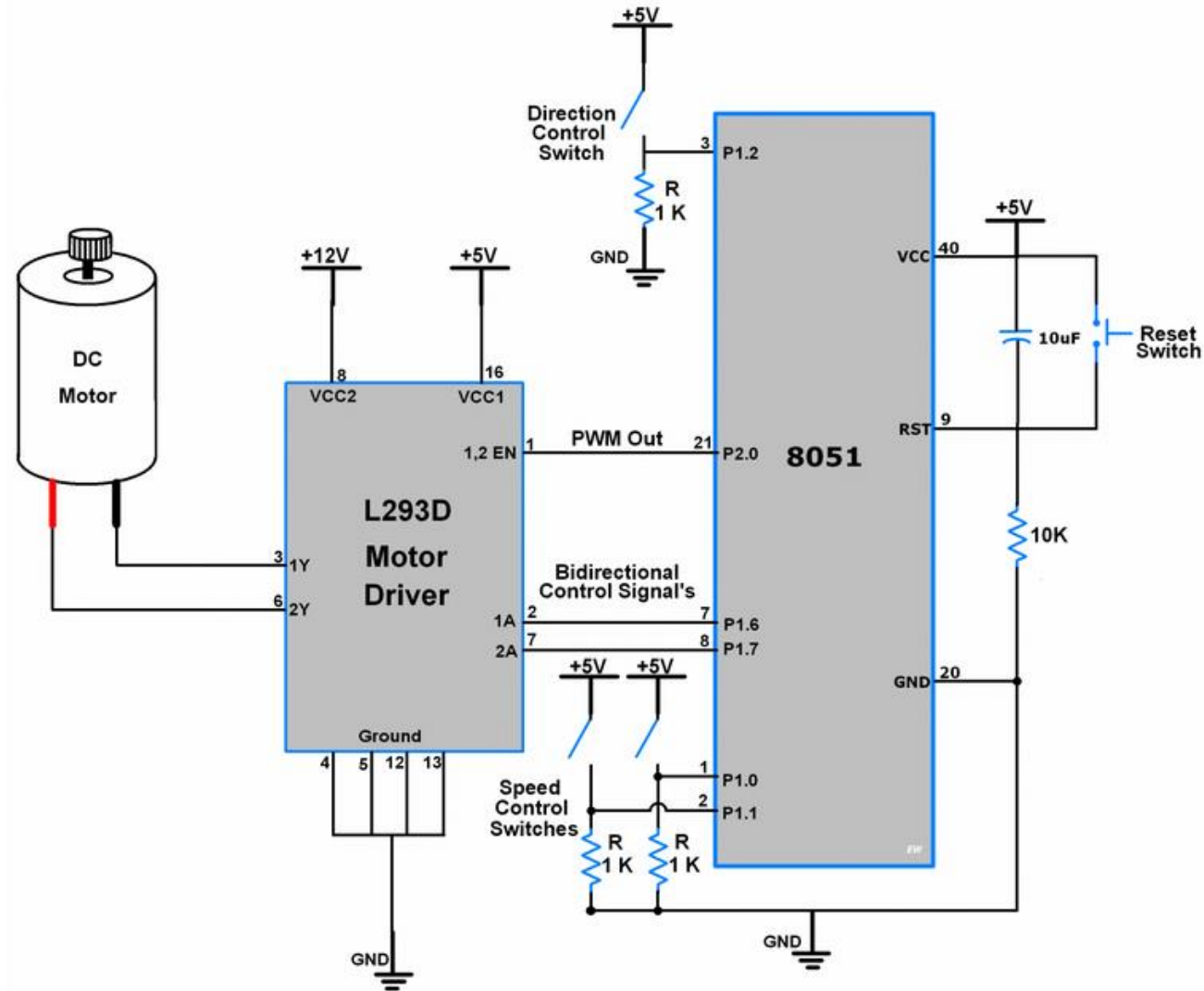


Interfacing of DC Motor with 8051

High Left	High Right	Low Left	Low Right	Description
On	Off	Off	On	Motor runs clockwise
Off	On	On	Off	Motor runs anti-clockwise
On	On	Off	Off	Motor stops or decelerates
Off	Off	On	On	Motor stops or decelerates



Interfacing of DC Motor with 8051



8051 DC Motor Connection using L293D Motor Driver

Interfacing of DC Motor with 8051

Let's interface the DC motor with the 8051 microcontroller and control the DC motor speed by using the Speed Increment Switch and Speed Decrement Switch connected to the Microcontroller port and direction by using Direction Switch.

We are going to use the L293D motor driver IC to control the DC motor movement in both directions. It has an in-built H-bridge motor drive.

- As shown in the above figure we have connected two toggle switches on the P1.0 and P1.1 pin of the AT89S52 microcontroller to change the speed of the DC motor by 10%.
- One toggle switch at pin P1.2 controls the motor's rotating direction.
- P1.6 and P1.7 pins are used as output direction control pins. It provides control to motor1 input pins of the L293D motor driver which rotate the motor clockwise and anticlockwise by changing their terminal polarity.
- And Speed of the DC Motor is varied through PWM Out pin P2.0.
- Here we are using the timer of AT89S52 to generate PWM.



Interfacing of Stepper Motor with 8051

A Stepper Motor is a brushless, synchronous motor which divides a full rotation into a number of steps. DC motor which rotates continuously when a fixed DC voltage is applied to it, while a step motor rotates in discrete step angles. The number of steps required to complete one complete rotations known as steps per revolution. If stepper motor has 12, 24, 72, 144, 180 and 200 resulting stepping angles are 30, 15, 5, 2.5, 2, and 1.8 degrees per step ($\text{step angle} = 360/\text{steps}$).



Interfacing of Stepper Motor with 8051

Working (Stepper Motor)

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor and electromagnets on the stationary portion that surrounds the motor, called the stator. Fig.3.12 illustrates one step rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner step by step upto starting position. In this example step angle is 90 degree and it will requires 4 steps to complete one rotation. This is full stepping method.



Interfacing of Stepper Motor with 8051

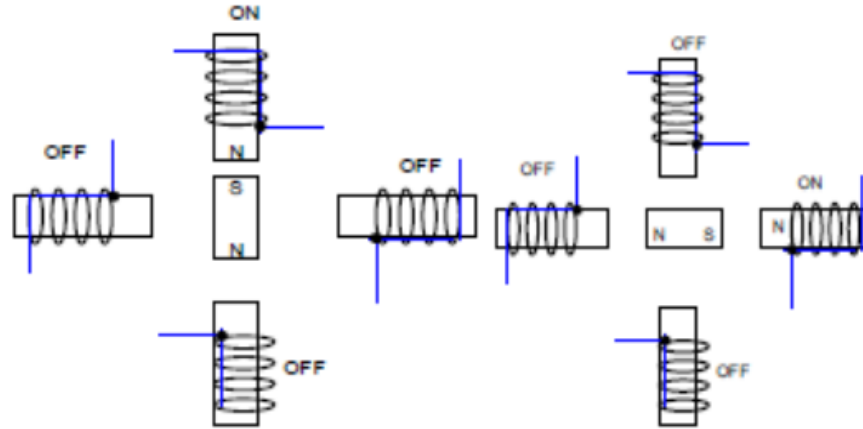


Fig 3.12 Full Stepping Method

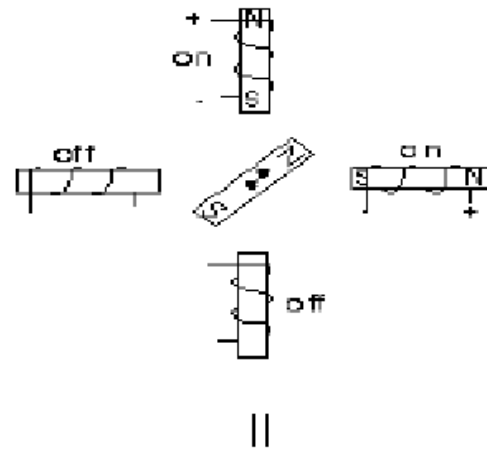
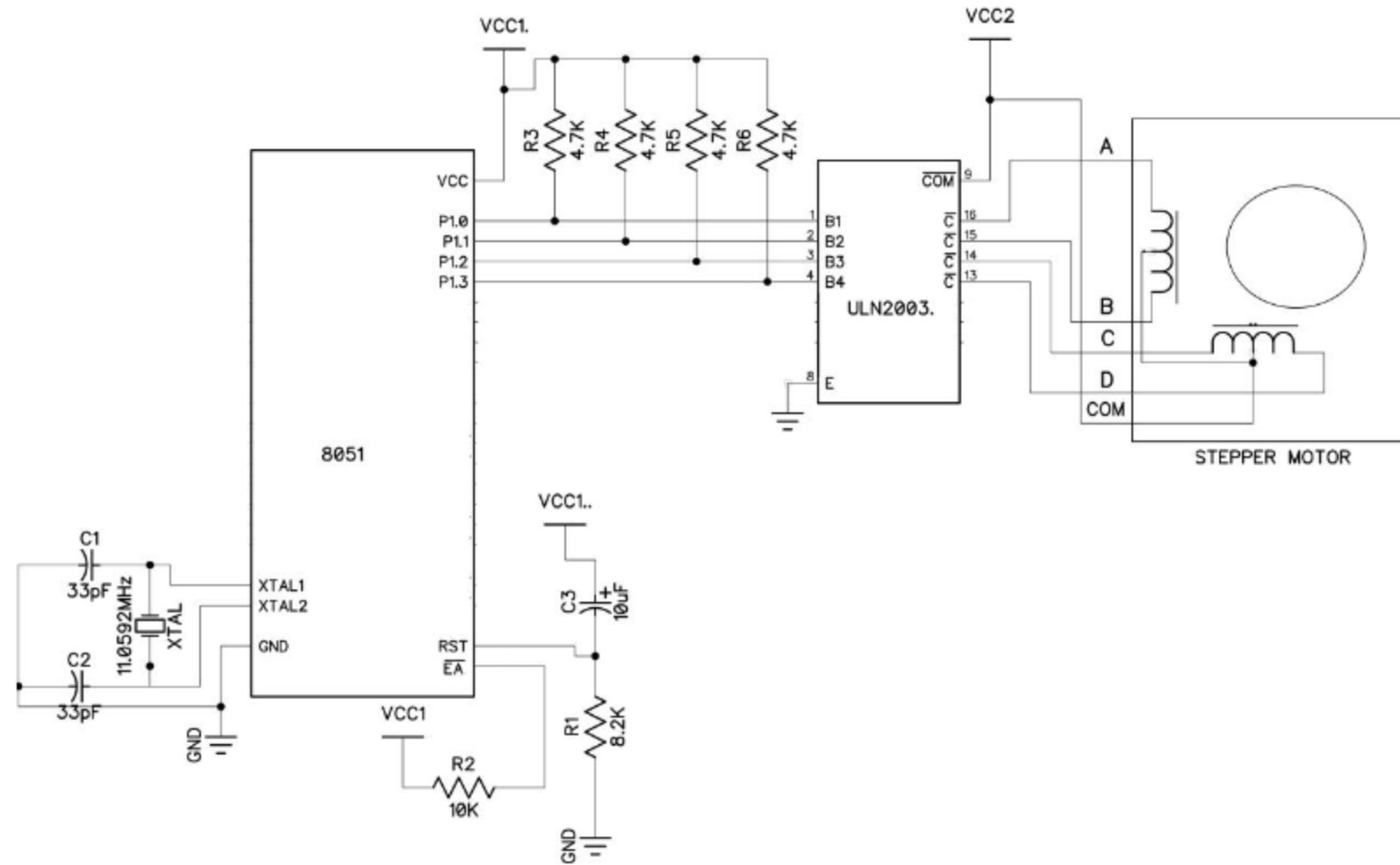


Fig. 3.13 Half stepping



Interfacing of Stepper Motor with 8051



Interfacing of Stepper Motor with 8051

ORG 0000H ; Start of program

MAIN:

MOV P1, #0H ; Clear Port 1 (Stepper Motor Control)

ROTATE:

MOV P1, #09H ; Step 1: 1001 (P1.3=1, P1.2=0, P1.1=0, P1.0=1)
ACALL DELAY

MOV P1, #0AH ; Step 2: 1010 (P1.3=0, P1.2=1, P1.1=0, P1.0=1)
ACALL DELAY

MOV P1, #06H ; Step 3: 0110 (P1.3=0, P1.2=1, P1.1=1, P1.0=0)
ACALL DELAY

MOV P1, #05H ; Step 4: 0101 (P1.3=1, P1.2=0, P1.1=1, P1.0=0)
ACALL DELAY

SJMP ROTATE ; Repeat indefinitely

; Delay Subroutine (Approximate for EdSim51)

DELAY:

MOV R3, #255
L1: MOV R2, #255
L2: DJNZ R2, L2
DJNZ R3, L1
RET

END

