



**K. D. K. College of Engineering ,  
Great Nag Road, Nandanvan, Nagpur**

**Department of Electronics & Telecommunication Engineering  
Microcontroller & Application  
(BEETC401P)**

**DEPARTMENT VISION**

Endeavoring in developing technically competent, confident and socially responsible Electronics Engineers

**DEPARTMENT MISSION**

- M1- Focus on teaching-learning process to spread in-depth knowledge of principles and its applications pertaining to Electronics Engineering and interdisciplinary areas.
- M2- To inculcate creative thinking through innovative and group work exercises which enhances the practical and project skills, entrepreneurship, employability and research capabilities.
- M3- Provide ethical and value based education by promoting activities addressing the societal needs.

**Subject Teacher :            Prof. Vijay V. Chakole  
Assistant Professor.**

## PROGRAMME EDUCATION OBJECTIVES

Graduates of Electronics Engineering shall:

- PEO1 -** Exhibit their knowledge and skills in analyzing and solving the real time engineering problems.
- PEO2 -** Be successful in careers, higher studies, research or entrepreneurial assignments.
- PEO3 -** Have exposure to emerging technologies, have opportunities to work as team member on multidisciplinary projects with effective communication skills and leadership qualities.

## Program Outcome (POs):

Electronics Engineering Graduates will be able to:

1. *Engineering knowledge :*  
Apply the knowledge of mathematics, science, Electronics Engineering fundamentals, and an Electronics engineering specialization to the solution of complex engineering problems.
2. *Problem analysis :*  
Identify, formulate, review research literature, and analyze complex Electronics engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. *Design/development of solutions :*  
Design solutions for complex Electronics engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. *Conduct investigations of complex problems :*  
Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. *Modern tool usage:*  
Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. *The engineer and society :*  
Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. *Environment and sustainability:*  
Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. *Ethics :*  
Apply ethical principles and commit to professional ethics and responsibilities and norms

of the engineering practice.

**9. Individual and team work :**

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**10. Communication :**

Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**11. Project management and finance :**

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**12. Life-long learning :**

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**PROGRAMME SPECIFIC OUTCOMES**

**PSO1** - Have ability to design and deploy systems in Electronics, Communication, Embedded Systems, Automation, VLSI, Signal Processing, Robotics and IoT.

**PSO2** - Have aptitudes towards design, development and research of innovative products/systems in electronics solving real time, societal problems.

## Microcontroller and Application Laboratory

As per Choice based Credit System (CBCS) scheme of R. T. M. Nagpur University

Semester : IV (Electronics & Telecommunication Engineering)

Laboratory Code : BEETC401P

No of LectureHours / week : 02 Hours Laboratory

College Assessment : 25

University Assessment : 25

### Course Objectives:

*The course objectives are:*

1. To perform a practical based on different microcontroller based systems.
2. To study assembly language programming skills.
3. Interface different peripherals with microcontrollers for its practical use.

### Course Outcomes:

*After the completion of practicals, the students will*

1. Demonstrate the concept of Assembly languages and higher level language programming.
2. Interface various peripherals with 8051, Atmega 32, MSP 430 and Arduino.
3. Simulate the programs on different software platforms.

### CO-PO MAPPING :

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO1												
CO2												
CO3												
CO (Avg)												

### Assessment of Internal Marks:

1. Record / Journal : 08 Marks
2. Lab Performance : 08 Marks
3. Written Test / performance : 05 Marks
4. Viva : 04 Marks

### List of Experiments :

Sr. No	Aim of the Practical
1	To study the Architecture and Pin Diagram of Microcontroller 8051
2	Write and Execute the Arithmetic Operation using 8051 Microcontroller.
3	Write and execute ALP for 8051 to convert HEX numbers into its equivalent ASCII code.
4	Write and execute ALP for 8051 to find the square of a number.
5	Write and execute ALP for 8051 to find the square root of a number.
6	Write and execute ALP for 8051 to sort n numbers.
7	Write and execute ALP for 8051 to for toggling the LED connected to one of the port pins of 8051.
8	Write and execute ALP for 8051 to display the decimal numbers in 7 Segment display.
9	Write and execute ALP for 8051 to interface stepper motor with 8051.
10	Write and execute ALP for 8051 to generate a square wave using 8051.
11	Write and execute ALP for 8051 to display a message in LCD display.
12	<b>To Study of AVR.</b>
13	Design an MCU AVR Atmega32 interfacing with LCD and displaying string and Modify this program to interface LM 34 for displaying temperature in Degree Centigrade and Fahrenheit on LCD display.
14	Write a program to interface an LED to the port 2 of MSP 430 microcontroller. Use both conditions of active low and active high in program.
15	Interface MSP 430 microcontroller with a matrix keyboard and display different characters on LCD
16	Design a traffic light controller using Arduino in timer mode for four roads. Use 3 LEDs, Red, Green and Yellow in each direction.
17	<b>To study of Arduino.</b>
18	Subject Practical Based Activity

Note : Perform minimum 8 Practical + one Mini Project

### **Experiment No 1**

**Aim :** To study the Architecture and Pin Diagram of Microcontroller 8051

## Experiment No 1

**Aim :** To study the Architecture and Pin Diagram of Microcontroller 8051

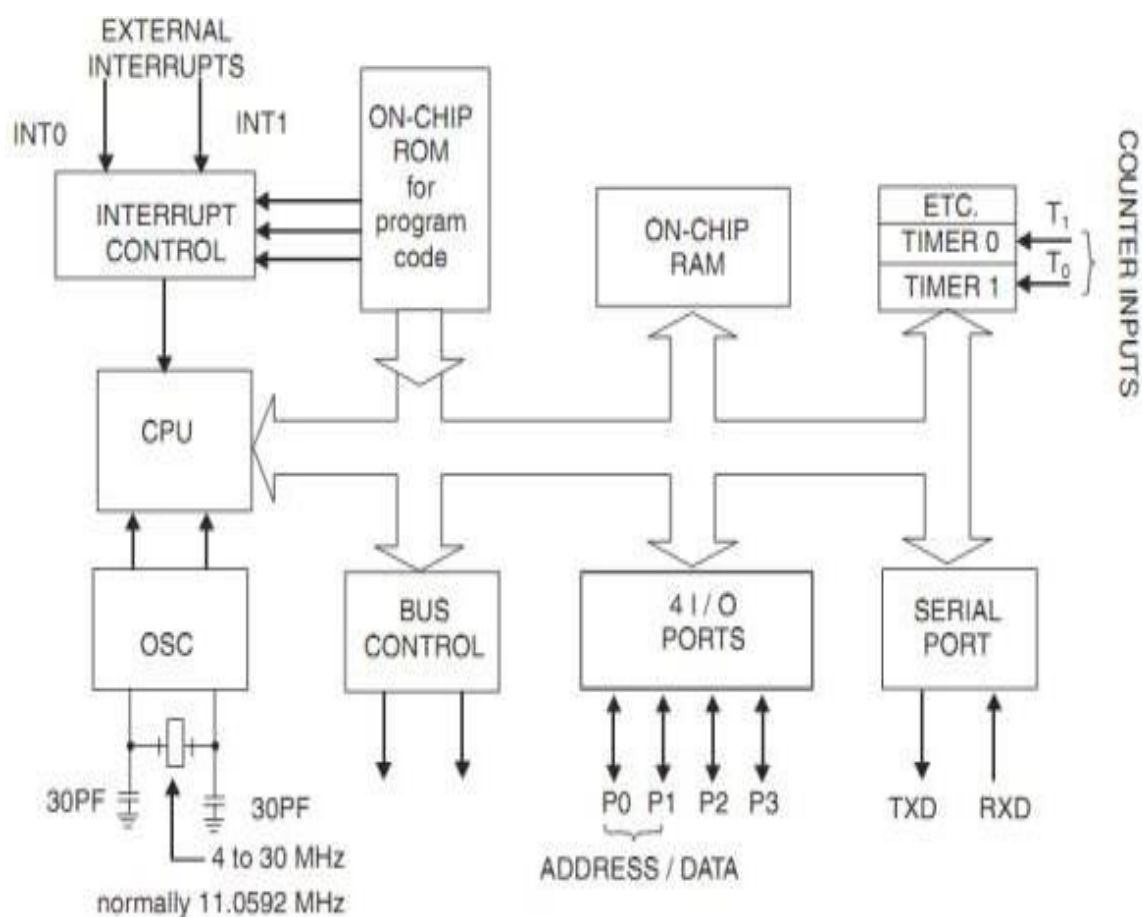
### Unit 1

#### Apparatus required:

8051 microcontroller kit

(0-5V) DC Power Supply

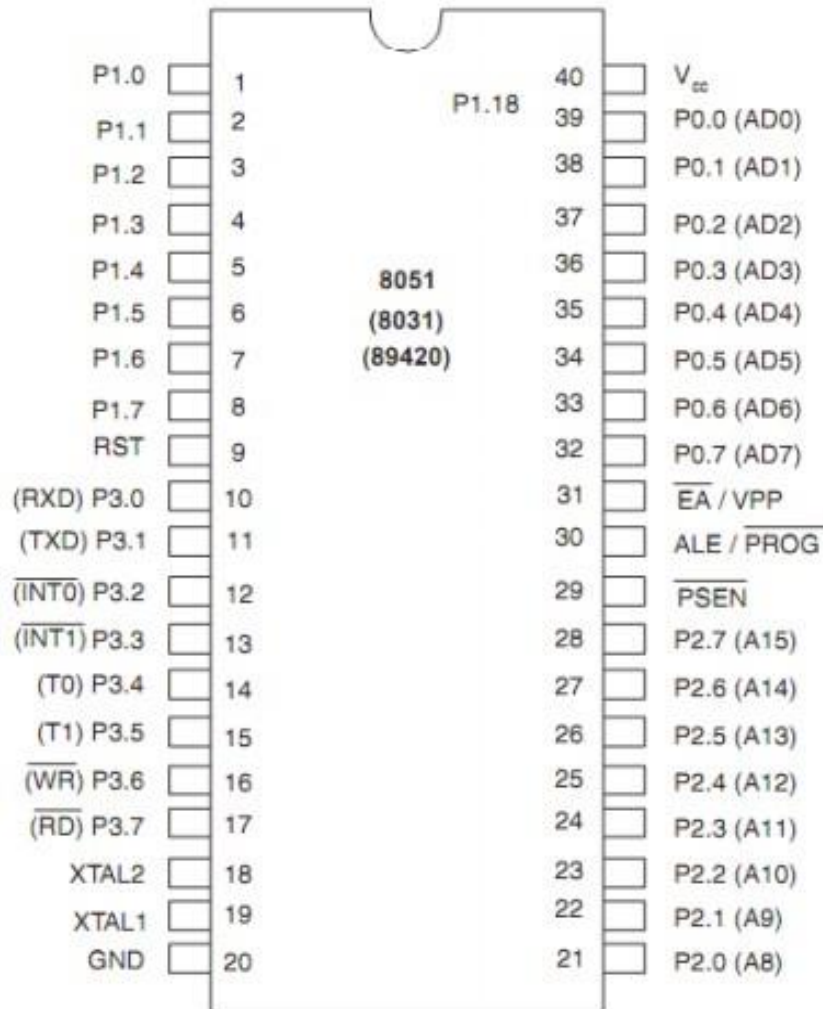
#### Architecture of 8051 Microcontroller



Architecture of 8051 microcontroller has following features

- 4 Kb of ROM is not much at all.
- 128Kb of RAM (including SFRs) satisfies the user's basic needs.
- 4 ports having in total of 32 input/output lines are in most cases sufficient to make all necessary connections to peripheral environment.

#### Pin Diagram of Microcontroller 8051:



### Pin out Description

**Pins 1-8:** Port 1 each of these pins can be configured as an input or an output.

**Pin 9:** RS A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.

**Pins10-17:** Port 3 Similar to port 1, each of these pins can serve as general input or output.

Besides, all of them have alternative functions:

**Pin 10:** RXD Serial asynchronous communication input or Serial synchronous communication output.

**Pin 11:** TXD Serial asynchronous communication output or Serial synchronous communication clock output.

**Pin 12:** INT0 Interrupt 0 inputs.

**Pin 13:** INT1 Interrupt 1 input.

**Pin 14:** T0 Counter 0 clock input.

**Pin 15:** T1 Counter 1 clock input.



**Pin 16:** WR Write to external (additional) RAM.

**Pin 17:** RD Read from external RAM.

**Pin 18, 19:** X2, X1 Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramics resonators can also be used for frequency stability. Later versions of microcontrollers operate at a frequency of 0 Hz up to over 50 Hz.

**Pin 20:** GND Ground.

**Pin 21-28:** Port 2 If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.

**Pin 29:** PSEN If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

**Pin 30:** ALE Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external register (usually 74HCT373 or 74HCT375 add-on chip) memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.

**Pin 31:** EA By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

**Pin 32-39:** Port 0 Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

**Pin 40:** VCC +5V power supply.

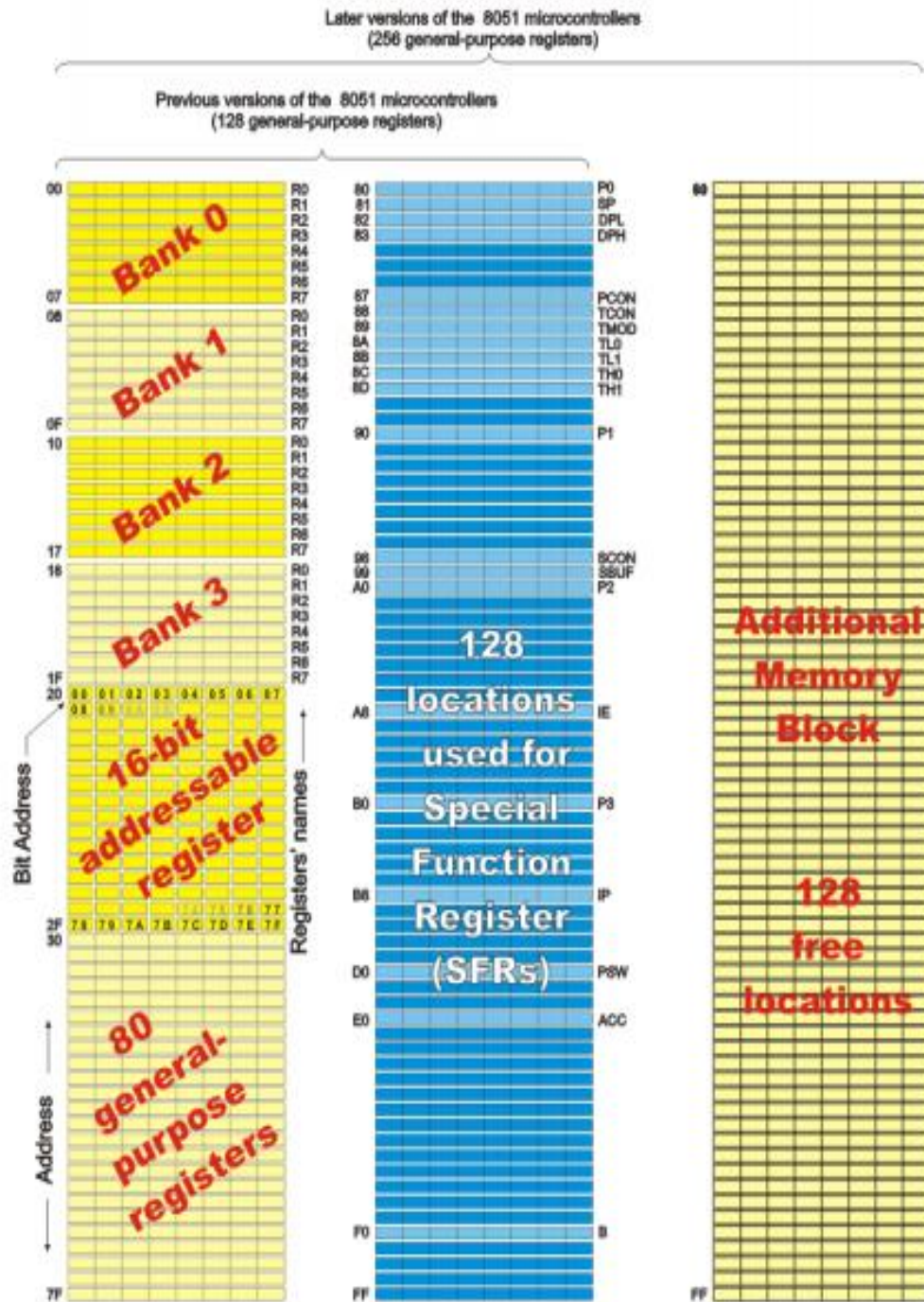
### **Input / Output Ports (I/O Ports)**

All 8051 microcontrollers have 4 I/O ports each comprising 8 bits which can be configured as inputs or outputs. Accordingly, in total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices are available for use.

Pin configuration, i.e. whether it is to be configured as an input (1) or an output (0), depends on its logic state. In order to configure a microcontroller pin as an input, it is necessary to apply a logic zero (0) to appropriate I/O port bit. In this case, voltage level on appropriate pin will be 0. Similarly, in order to configure a microcontroller pin as an output, it is necessary to apply a logic one (1) to appropriate port. In this case, voltage level on appropriate pin will be 5V (as is the case with any TTL input). This may seem confusing but don't lose your patience. It all becomes clear after studying simple electronic circuits connected to an I/O pin.

### **Memory Organization :**

The 8051 has two types of memory and these are Program Memory and Data Memory. Program Memory (ROM) is used to permanently save the program being executed, while Data Memory (RAM) is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Depending on the model in use (we are still talking about the 8051 microcontroller family in general) at most a few Kb of ROM and 128 or 256 bytes of RAM is used. However All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory. It is neither a mistake nor a big ambition of engineers who were working on basic core development. It is a matter of smart memory organization which makes these microcontrollers a real “programmers’ goody“.




### Special Function Registers (SFRs)

Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc. Even though there are 128 memory locations intended to be occupied by them, the basic core, shared by all types of 8051 microcontrollers, has only 21 such registers. Rest of locations is intentionally left unoccupied in order to enable the manufacturers to

further develop microcontrollers keeping them compatible with the previous versions. It also enables programs written a long time ago for microcontrollers which are out of production now to be used today.

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87


 Bit-addressable Registers

### Program Status Word (PSW) Register:

	0	0	0	0	0	0	0	Value after Reset
<b>PSW</b>	CY	AC	F0	RS1	RS0	OV		Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

PSW register is one of the most important SFRs. It contains several status bits that reflect the current state of the CPU. Besides, this register contains Carry bit, Auxiliary Carry, two register bank select bits, Overflow flag, parity bit and user-definable status flag. **P - Parity bit.** If a number stored in the accumulator is even then this bit will be automatically set (1), otherwise it will be cleared (0). It is mainly used during data transmit and receive via serial communication.

**Bit 1.** This bit is intended to be used in the future versions of microcontrollers. **OV Overflow** occurs when the result of an arithmetical operation is larger than 255 and cannot be stored in one register. Overflow condition causes the OV bit to be set (1). Otherwise, it will be cleared (0).

**RS0, RS1 - Register bank select bits.** These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM.

RS1	RS2	Space in RAM
0	0	Bank0 00h-07h
0	1	Bank1 08h-0Fh
1	0	Bank2 10h-17h
1	1	Bank3 18h-1Fh

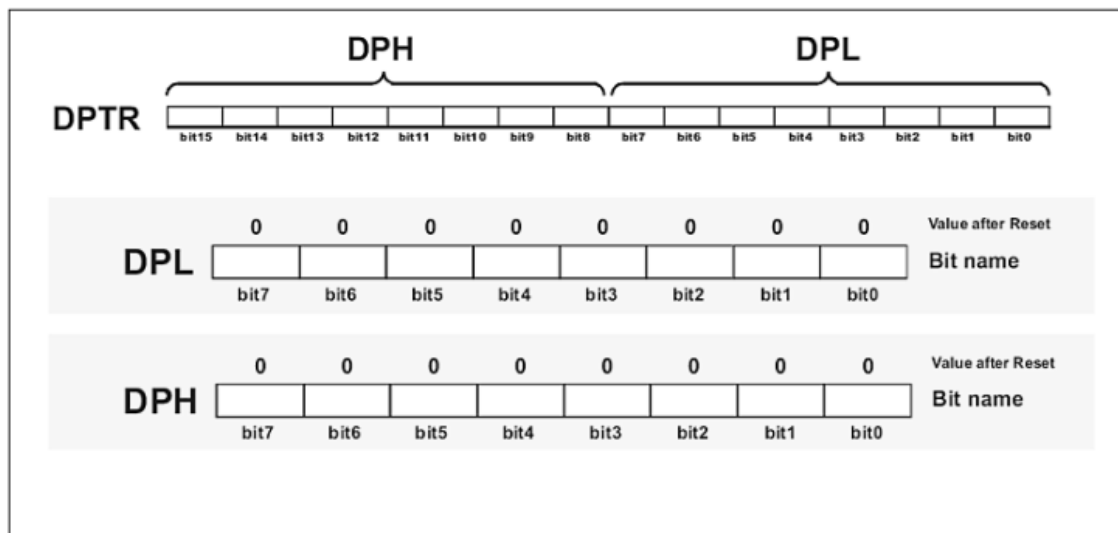
**F0 - Flag 0.** This is a general-purpose bit available for use.

**AC - Auxiliary Carry Flag** is used for BCD operations only.

**CY - Carry Flag** is the (ninth) auxiliary bit used for all arithmetical operations and shift instructions.

### Data Pointer Register (DPTR)

DPTR register is not a true one because it doesn't physically exist. It consists of two separate registers: DPH (Data Pointer High) and (Data Pointer Low). For this reason it may be treated as a 16-bit register or as two independent 8-bit registers. Their 16 bits are primarily used for external memory addressing. Besides, the DPTR Register is usually used for storing data and intermediate results.



### Stack Pointer (SP) Register



A value stored in the Stack Pointer points to the first free stack address and permits stack availability. Stack pushes increment the value in the Stack Pointer by 1. Likewise, stack pops decrement its value by 1. Upon any reset and power-on, the value 7 is stored in the Stack Pointer, which means that the space of RAM reserved for the stack starts at this location. If another value is written to this register, the entire Stack is moved to the new memory location.

### P0, P1, P2, P3 – Input / Output Registers

<b>P0</b>	1	1	1	1	1	1	1	Value after Reset
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller. Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

As mentioned, port bit state affects performance of port pins, i.e. whether they will be configured as inputs or outputs. If a bit is cleared (0), the appropriate pin will be configured as an output, while if it is set (1), the appropriate pin will be configured as an input. Upon reset and power-on, all port bits are set (1), which means that all appropriate pins will be configured as inputs.

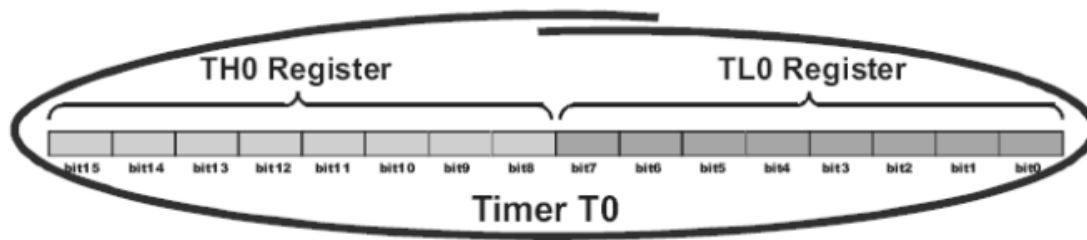
### Counters and Timers:

As you already know, the microcontroller oscillator uses quartz crystal for its operation. As the frequency of this oscillator is precisely defined and very stable, pulses it generates are always of the same width, which makes them ideal for time measurement. Such crystals are also used in quartz watches. In order to measure time between two events it is sufficient to count up pulses coming from this oscillator. That is exactly what the timer does. If the timer is properly programmed, the value stored in its register will be incremented (or decremented) with each coming pulse, i.e. once per each machine cycle.

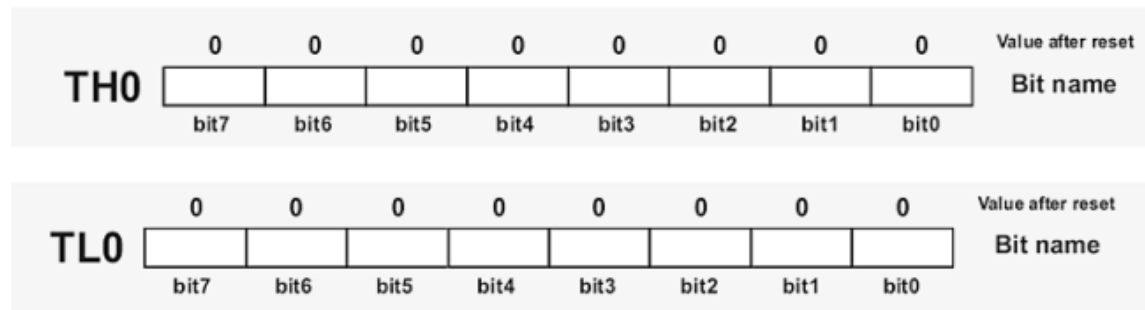
A single machine-cycle instruction lasts for 12 quartz oscillator periods, which means that by embedding quartz with oscillator frequency of 12MHz, a number stored in the timer register will be changed million times per second, i.e. each microsecond. The 8051 microcontroller has 2 timers/counters called T0 and T1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication, so called Baud Rate.

### Timer T0

As seen in figure below, the timer T0 consists of two registers – TH0 and TL0 representing a low and a high byte of one 16-digit binary number.



Accordingly, if the content of the timer T0 is equal to 0 ( $T0=0$ ) then both registers it consists of will contain 0. If the timer contains for example number 1000 (decimal), then the TH0 register (high byte) will contain the number 3, while the TL0 register (low byte) will contain decimal number 232.

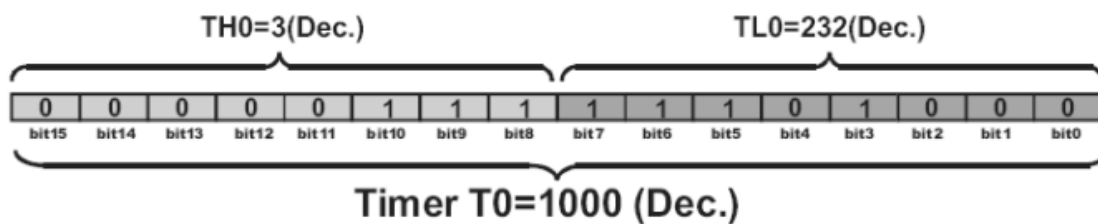


Formula used to calculate values in these two registers is very simple:

$$TH0 \times 256 + TL0 = T$$

Matching the previous example it would be as follows:

$$3 \times 256 + 232 = 1000$$

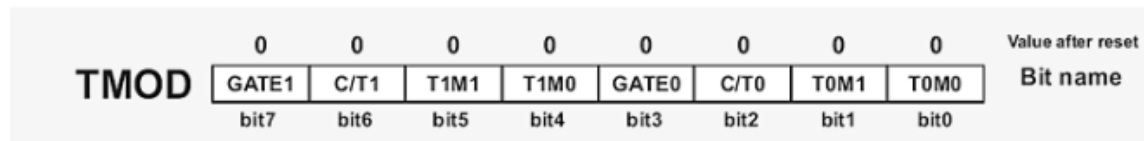


Since the timer T0 is virtually 16-bit register, the largest value it can store is 65 535. In case of exceeding this value, the timer will be automatically cleared and counting starts from 0. This condition is called an overflow. Two registers TMOD and TCON are closely connected to this timer and control its operation.

### TMOD Register (Timer Mode)

The TMOD register selects the operational mode of the timers T0 and T1. As seen in figure below,

the low 4 bits (bit0 - bit3) refer to the timer 0, while the high 4 bits (bit4 - bit7) refer to the timer 1. There are 4 operational modes and each of them is described herein.



Bits of this register have the following function:

- **GATE1** enables and disables Timer 1 by means of a signal brought to the INT1 pin (P3.3):

- o **1** - Timer 1 operates only if the INT1 bit is set.
- o **0** - Timer 1 operates regardless of the logic state of the INT1 bit.

- **C/T1** selects pulses to be counted up by the timer/counter 1:

- o **1** - Timer counts pulses brought to the T1 pin (P3.5).
- o **0** - Timer counts pulses from internal oscillator.

- **T1M1, T1M0** These two bits select the operational mode of the Timer 1.

T1M1	T1M0	Mode	Description
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto reload
1	1	3	Split mode

- **GATE0** enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):

- o **1** - Timer 0 operates only if the INT0 bit is set.
- o **0** - Timer 0 operates regardless of the logic state of the INT0 bit.

- **C/T0** selects pulses to be counted up by the timer/counter 0:

- o **1** - Timer counts pulses brought to the T0 pin (P3.4).
- o **0** - Timer counts pulses from internal oscillator.

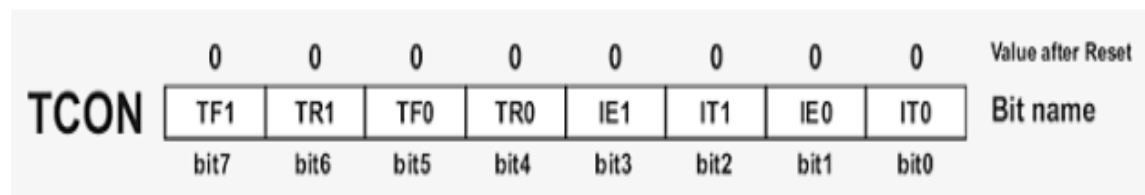


- **T0M1, T0M0** These two bits select the optional mode of the Timer 0.

T0M1	T0M0	Mode	Description
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

### Timer Control (TCON) Register

TCON register is also one of the registers whose bits are directly in control of timer operation. Only 4 bits of this register are used for this purpose, while rest of them is used for interrupt control to be discussed later.



- **TF1** bit is automatically set on the Timer 1 overflow.

- **TR1** bit enables the Timer 1.

o **1** - Timer 1 is enabled.

o **0** - Timer 1 is disabled.

- **TF0** bit is automatically set on the Timer 0 overflow.

- **TR0** bit enables the timer 0.

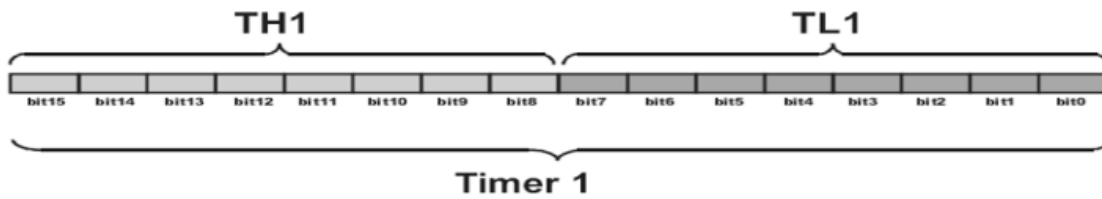
o **1** - Timer 0 is enabled.

o **0** - Timer 0 is disabled.

### Timer

1

Timer 1 is identical to timer 0, except for mode 3 which is a hold-count mode. It means that they have the same function, their operation is controlled by the same registers TMOD and TCON and both of them can operate in one out of 4 different modes.



	0	0	0	0	0	0	0	Value after Reset
<b>TH1</b>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

	0	0	0	0	0	0	0	Value after Reset
<b>TL1</b>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

### Result:

Thus the 8051 Microcontroller Architecture and Pin Description has been studied.

## **Experiment No 2**

**Aim :** To write and Execute the Arithmetic Operation using 8051 Microcontroller.

## Experiment No 2

**Aim :** To write and Execute the Arithmetic Operation using 8051 Microcontroller.

1. Addition of Two 8 bit Numbers.
2. Subtraction of Two 8 bit Numbers.
3. Multiplication of Two 8 bit Numbers.
4. Division of Two 8 bit Numbers.

### Unit 1

#### Apparatus required:

8051 microcontroller kit

(0-5V) DC Power Supply

#### Theory :

#### 8051 Program to Add two 8 Bit numbers

Intel 8051 is an 8-bit microcontroller. It has many powerful instructions and IO accessing techniques. In this section, we will see one of the simplest program using 8051.

Here we will add 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 5FH and D8H at location 20H and 21H. After adding them, the result will be stored at location 30H and 31H.

Address	Value
20H	5FH
21H	D8H
	.
	.
	.
30H	00H
31H	00H

#### Program

MOVR0, #20H;	set source address 20H to R0
MOVR1, #30H;	set destination address 30H to R1
MOVA, @R0;	take the value from source to register A

MOVR5,A;	Move the value from A to R5
MOVR4,#00H;	Clear register R4 to store carry
INCR0;	Point to the next location
MOVA,@R0;	take the value from source to register A
ADDA,R5;	Add R5 with A and store to register A
JNC SAVE	
INCR4;	Increment R4 to get carry
MOVB,R4;	Get carry to register B
MOV@R1,B;	Store the carry first
INCR1;	Increase R1 to point to the next address
SAVE:	MOV@R1,A;Store the result
HALT:	SJMP HALT ; Stop the program

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

### Output

Address	Value
20H	5FH
21H	D8H
	.
	.
	.
30H	01H
31H	37H

### 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 number 73H and BDH at location 20H and 21H, After subtracting the result will be stored at location 30H and 31H.

Address	Value
20H	73H
21H	BDH

Address	Value
	.
30H	00H
31H	00H

#### Program

```

MOVR0, #20H;      set source address 20H to R0
MOVR1, #30H;      set destination address 30H to R1

MOVA, @R0;        take the value from source to register A
MOVR5, A;         Move the value from A to R5
MOVR4, #00H;      Clear register R4 to store borrow

INCR0;            Point to the next location
MOVA, @R0;        take the value from source to register A
MOVR3, A;         store second byte
MOVA, R5;         get back the first operand
SUBBA, R3;        Subtract R3 from A

        JNC SAVE
        INCR4;      Increment R4 to get borrow
        MOVB, R4;   Get borrow to register B
        MOV@R1, B;  Store the borrow first
        INCR1;      Increase R1 to point to the next address

SAVE:  MOV@R1, A;   Store the result
HALT:  SJMP HALT ;  Stop the program

```

So by subtracting 73H –BDH, the result will be B6H. At location 30H, we will get 01H. This indicates that the result is negative. To 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
20H	73H
21H	BDH

Address	Value
	.
30H	01H
31H	B6H

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 number FFH and FFH at location 20H and 21H, After multiplying the result will be stored at location 30H and 31H.

Address	Value
20H	FFH
21H	FFH
	.
30H	00H
31H	00H

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
INCR0;           Point to the next location
MOV B,@R0;       take the second operand from source to register
                  B
MUL AB ;         Multiply A and B
MOV @R1, B;      Store higher order byte to 30H
INC R1;          Increase R1 to point to the next location
MOV @R1, A;      Store lower order byte to 31H
HALT:           SJMP HALT ; Stop the program

```

8051 provides **MULAB** instruction. By using this instruction, the multiplication can be done. In some other microprocessors like 8085, there was no MUL instruction. In that microprocessor, we need to use repetitive ADD operations to get the result of the multiplication.

When the result is below 255, the overflow flag OV is low, otherwise, it is 1.

Output

Address	Value
20H	FFH
21H	FFH
	.
30H	FEH
31H	01H

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 this 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 numbers 0EH and 03H at location 20H and 21H. After dividing the result will be stored at location 30H and 31H.

Address	Value
20H	0EH
21H	03H
	.
30H	00H
31H	00H



### 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
INC R0;       Point to the next location
MOV B, @R0;   take the second operand from source to register B

DIV AB ;      Divide A by B

MOV @R1, A;   Store Quotient to 30H
INC R1;       Increase R1 to point to the next location
MOV @R1, B;   Store Remainder to 31H
HALT:        SJMP HALT ;Stop the program
```

8051 provides **DI VAB** instruction. By using this instruction, the division can be done. In some other microprocessors like 8085, there was no DIV instruction. In that microprocessor, we need to use repetitive Subtraction operations to get the result of the division.

When the denominator is 00H, the overflow flag OV will be 1. otherwise it is 0 for the division.

### Output

Address	Value
20H	0EH
21H	03H
	.
	.
30H	04H
31H	02H

### Result:

Thus the Arithmetic Operation like Addition, Subtraction, Multiplication and Division using 8051 Microcontroller has been Executed.

### **Experiment No 3**

**Aim ;** Write and execute ALP for 8051 to convert HEX numbers into its equivalent ASCII code.

### Experiment No 3

**Aim ;** Write and execute ALP for 8051 to convert HEX numbers into its equivalent ASCII code.

#### Unit 1

##### Apparatus required:

8051 microcontroller kit

(0-5V) DC Power Supply

##### Theory :

##### Hex to ASCII conversion in 8051

Now we will see how to convert Hexadecimal number to its ASCII equivalent using 8051. This program can convert 0-9 and A-F to its ASCII value.

We know that the ASCII of number 00H is 30H (48D), and ASCII of 09H is 39H (57D). So all other numbers are in the range 30H to 39H. The ASCII value of 0AH is 41H (65D) and ASCII of 0FH is 46H (70D), so all other alphabets (B, C, D, E) are in the range 41H to 46H.

Here we are providing hexadecimal digit at memory location 20H, The ASCII equivalent is storing at location 30H.

Address	Value
20H	0EH
21H	
	.

##### Program

```
MOV R0, #20H;   Initialize the address of the data
MOVA, @R0;      Get the data from an address, which is stored in R0
MOV R2, A;       Store the content of A into R2
CLR C;          Clear the Carry Flag
SUBB A, #0AH;    Subtract 0AH from A
```

```

JCNUM ;      When a carry is present, A is numeric
ADDA,#41H;    Add41H for Alphabet
SJMPSTORE;    Jump to store the value
NUM:MOVA,R2;  Copy R2 to A
ADDA,#30H;    Add 30H with A to get ASCII
STORE:MOVR0,#30H; Point the destination location
MOV@R0,A;     Store A content to the memory location pointed by R0
HALT:    SJMPHALT

```

The logic behind HEX to ASCII conversion is very simple. We are just checking whether the number is in range 0 – 9 or not. When the number is in that range, then the hexadecimal digit is numeric, and we are just simply adding 30H with it to get the ASCII value. When the number is not in range 0 – 9, then the number is range A – F, so for that case, we are converting the number to 41H onwards.

In the program, at first, we are clearing the carry flag. Then subtracting 0AH from the given number. If the value is numeric, then after subtraction the result will be negative, so the carry flag will be set. Now by checking the carry status, we can just add 30H with the value to get ASCII value.

In other hands when the result of the subtraction is positive or 0, then we are adding 41H with the result of the subtraction.

Output

Address	Value
20H	0EH
21H	
	.
30H	45H
31H	

Result :Thus the HEX numbers into its equivalent ASCII code using 8051 Microcontroller has been Executed.

#### **Experiment No 4**

**AIM:** Write an assembly language program to find the square of a number.

## Experiment No 4

**AIM:** Write an assembly language program to find the square of a number.

### Unit 1

#### Apparatus required:

8051 microcontroller kit

(0-5V) DC Power Supply

#### PROGRAM:

```
ORG 0000H
MOV A,60H
MOV B,A
MUL AB
MOV 70H,A
MOV 71H,B
END
```

#### OUTPUT:

	LOCATION	DATA
INPUT	60H	03
OUTPUT	70H	09
	71H	00

Result : Thus the square of the number using 8051 Microcontroller has been Executed.

### **Experiment No 5**

**AIM:** Write an assembly language program to find the square root of a number.

## Experiment No 5

**AIM:** Write an assembly language program to find the square root of a number.

### Unit 1

#### Apparatus required:

8051 microcontroller kit, (0-5V) DC Power Supply

#### PROGRAM:

```
ORG 0000H
MOV A,60H
MOV R1,#00H
MOV R0,#01H
L1:
SUBB A,R0
JC L2
INC R0
INC R0
INC R1
JNZ L1
MOV A,R1
SJMP L3
L2:
MOV A,#0FFH
L3:
MOV 70H,A
END
```

#### OUTPUT

	LOCATION	DATA
INPUT	60H	09
OUTPUT	70H	03

Result : Thus the square root of the number using 8051 Microcontroller has been Executed.



## **Experiment No 6**

**AIM:** Write an assembly language program to sort n numbers.

## Experiment No 6

**AIM:** Write an assembly language program to sort n numbers.

### Unit 1

#### Apparatus required:

8051 microcontroller kit, (0-5V) DC Power Supply

#### PROGRAM:

```
ORG 0000H
MOV A,50H
MOV R2,A
DEC R2
L3:
MOV A,R2
MOV R3,A
MOV R0,#51H
L2:
MOV A,@R0
MOV R4,A
INC R0
MOV A,@R0
MOV R5,A
SUBB A,R4
JNC L1
MOV A,R4
MOV @R0,A
DEC R0
MOV A,R5
MOV @R0,A
INC R0
L1:
DJNZ R3,L2
DJNZ R2,L3
```

END

**OUTPUT:**

	LOCATION	DATA
<b>INPUT</b>	50H	05
	51H	03
	52H	02
	53H	05
	54H	01
	55H	04
<b>OUTPUT</b>	50H	05
	51H	01
	52H	02
	53H	03
	54H	04
	55H	0

Result : Thus the number are sort in ascending order using 8051 Microcontroller has been Executed.

## **Experiment No 7**

**AIM:** Write an assembly language program to for toggling the LED connected to one of the port

## Experiment No 7

**AIM:** Write an assembly language program to toggle the LED connected to one of the port pins of 8051.

### Unit 2

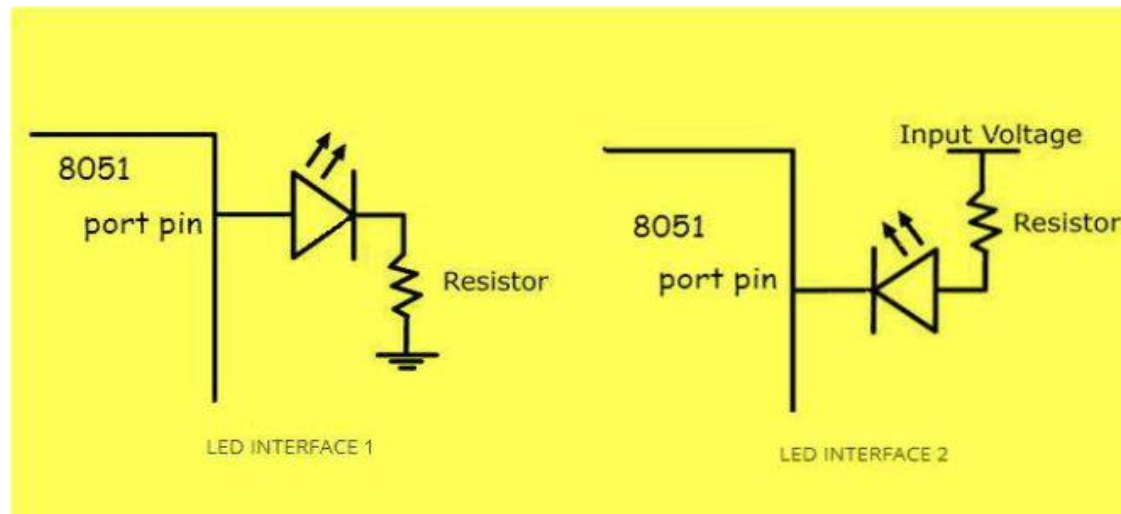
#### Apparatus required:

8051 microcontroller, LED, (0-5V) DC Power Supply

#### THEORY:

Interfacing comprises of hardware (Interface device) and Software (source code to communicate, also called as the Driver). Simply, to use an LED as the output device, LED should be connected to Microcontroller port and the MC has to be programmed inside make LED ON or OFF or blink or dim. This program is called as the driver / firmware. The driver software can be developed using any programming language like Assembly, C etc.

There are two ways which we can interface LED to the Microcontroller 8051. But the connections and programming techniques will be different. This article provides the information on LED interfacing with 8051 and LED blinking code for AT89C52/ AT89C51 Microcontroller.



#### Interfacing LED to 8051 Methods.

Observe carefully the interface LED 2 is in forward biased because the input voltage of 5v connected to the positive terminal of the LED, So here the Microcontroller pin should be at LOW level and vice versa with the interface 1 connections.

The resistor is important in LED interfacing to limit the flowing current and avoid damaging the

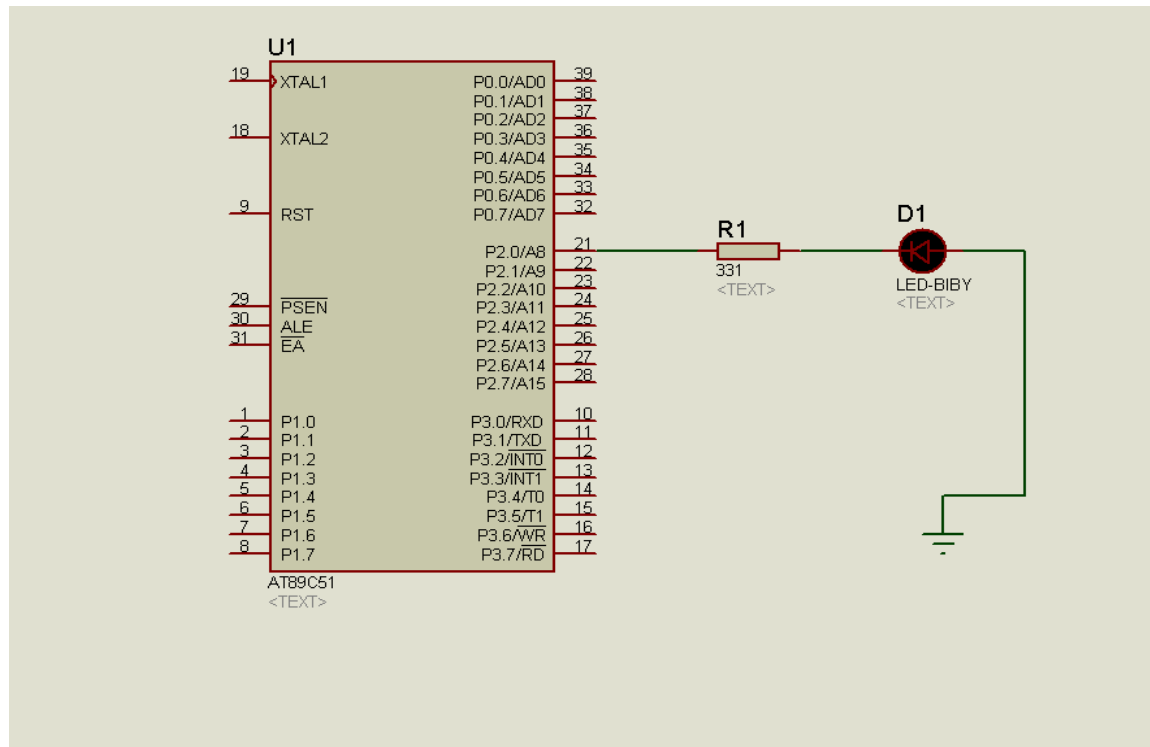
LED and/or MCU.

- Interface 1 will glow LED, only if the PIN value of the MC is HIGH as current flows towards the ground.
- Interface 2 will glow LED, only if the PIN value of the MC is LOW as current flows towards PIN due to its lower potential.

### **PROGRAM:**

```
ORG 0000H
LOOP: SETB P1.0
ACALL DELAY
CLR P1.0
ACALL DELAY
SJMP LOOP
DELAY: MOV R0,#100
AGAIN: MOV R1,#200
BACK: DJNZ R1,BACK
DJNZ R0,AGAIN
RET
END
```

### **SCHEMATIC DIAGRAM:**



Result : Thus the toggling of LED connected to one of the port pins of 8051 Microcontroller has been Executed.

### **Experiment No 8**

**AIM:** Write an assembly language program for displaying the decimal numbers in 7 Segment display.



## Experiment No 8

**AIM:** Write an assembly language program for displaying the decimal numbers in 7 Segment display.

### Unit 2

#### Apparatus required:

8051 microcontroller, 7 Segment display, Resistors, (0-5V) DC Power Supply

#### THEORY:

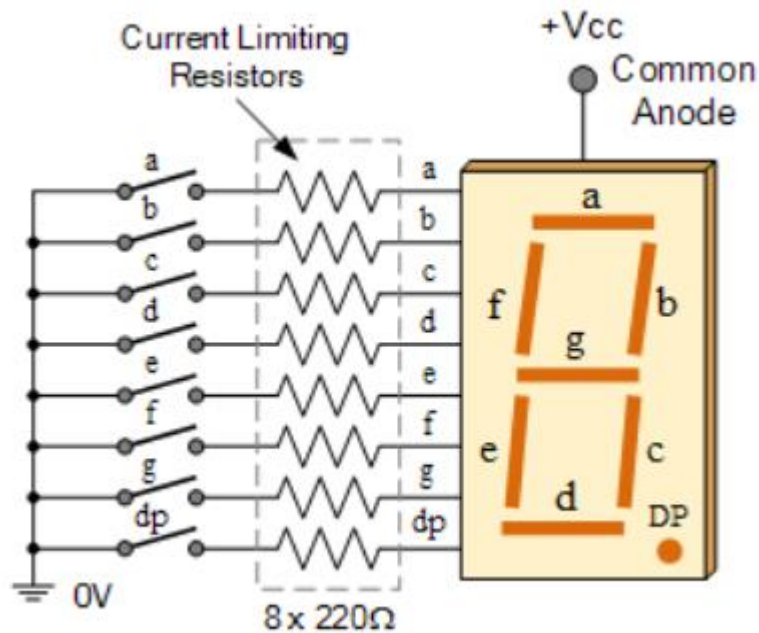
The *7-segment display*, also written as “seven segment display”, consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point, (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

Each one of the seven LEDs in the display is given a positional segment with one of its connection pins being brought straight out of the rectangular plastic package. These individually LED pins are labelled from a through to g representing each individual LED. The other LED pins are connected together and wired to form a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will be light and others will be dark allowing the desired character pattern of the number to be generated on the display. This then allows us to display each of the ten decimal digits 0 through to 9 on the same 7-segment display.

The displays common pin is generally used to identify which type of 7-segment display it is. As each LED has two connecting pins, one called the “Anode” and the other called the “Cathode”, there are therefore two types of LED 7-segment display called: **Common Cathode** (CC) and **Common Anode** (CA).

The difference between the two displays, as their name suggests, is that the common cathode has all the cathodes of the 7-segments connected directly together and the common anode has all the anodes of the 7-segments connected together and is illuminated as follows. The Common Cathode (CC) – In the common cathode display, all the cathode connections of the LED

segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the individual Anode terminals (a-g).



### PROGRAM:

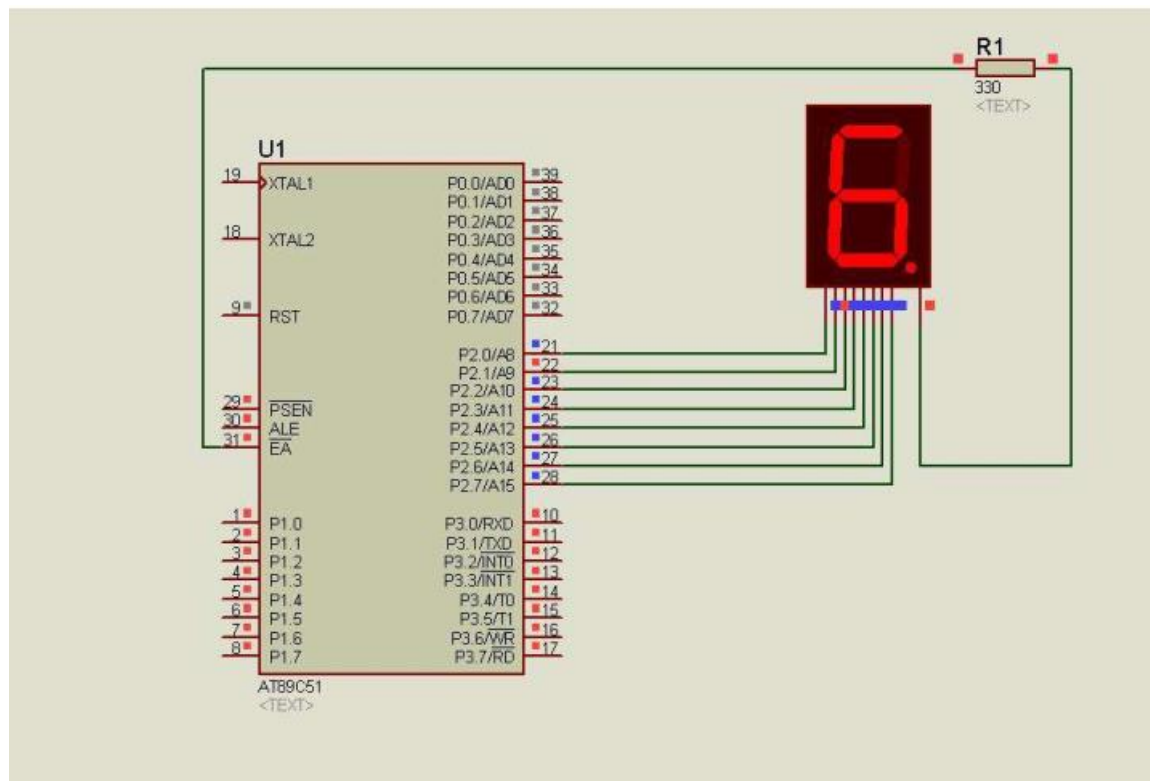
```
ORG 0000H
LOOP: MOV DPTR, #0100H
MOV R1, #0AH
BACK: CLR A
MOVC A, @A+DPTR
MOV P2, A
ACALL DELAY
INC DPTR
DJNZ R1, BACK
SJMP LOOP
DELAY: MOV R5, #05
BACK2: MOV R3, #255
BACK1: MOV R4, #255
AGAIN: DJNZ R4, AGAIN
```

```

DJNZ R3,BACK1
DJNZ R5,BACK2
RET
ORG 0100H
DB 40H,79H,24H,30H,19H,12H,02H,58H,00H,10H
END

```

### SCHEMATIC DIAGRAM:



Result : Thus the displaying the decimal numbers in 7 Segment display using 8051 Microcontroller has been Executed.

## **Experiment No 9**

**AIM:** Write an assembly language program for interfacing stepper motor with 8051.

## Experiment No 9

**AIM:** Write an assembly language program for interfacing stepper motor with 8051.

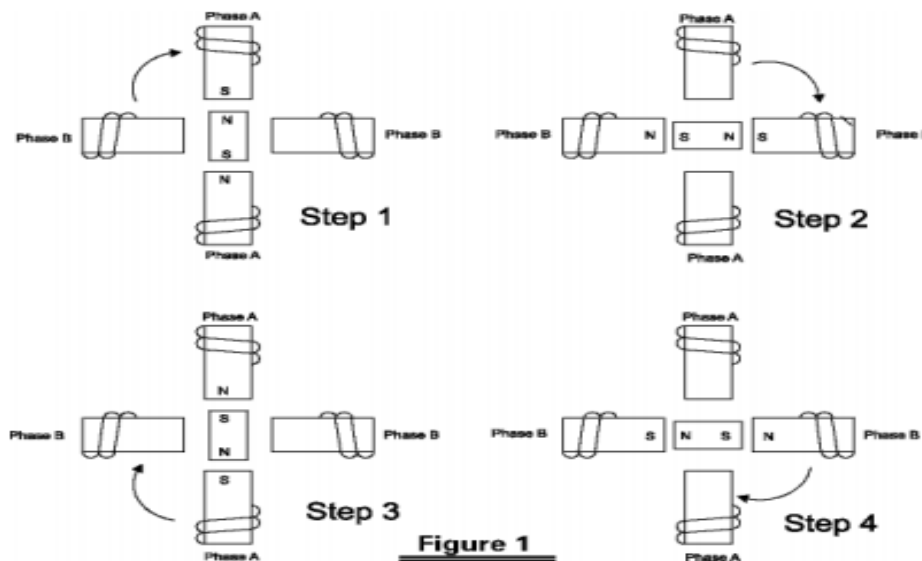
### Unit 2

#### Apparatus required:

8051 microcontroller, Stepper Motor, Resistors, (0-5V) DC Power Supply

#### THEORY:

A **Stepper Motor** or a **step motor** is a brushless, synchronous motor which divides a full rotation into a number of steps. Unlike a brushless DC motor which rotates continuously when a fixed DC voltage is applied to it, a step motor rotates in discrete step angles. The **Stepper Motors** therefore are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees per step. The stepper motor can be controlled with or without feedback. Stepper motors work on the principle of electromagnetism. There is a soft iron or magnetic rotor shaft surrounded by the electromagnetic stators. The rotor and stator have poles which may be teathed or not depending upon the type of stepper. When the stators are energized the rotor moves to align itself along with the stator (in case of a permanent magnet type stepper) or moves to have a minimum gap with the stator (in case of a variable reluctance stepper). This way the stators are energized in a sequence to rotate the stepper motor



#### PROGRAM:

```
ORG 0000H  
MOV A,#66H
```

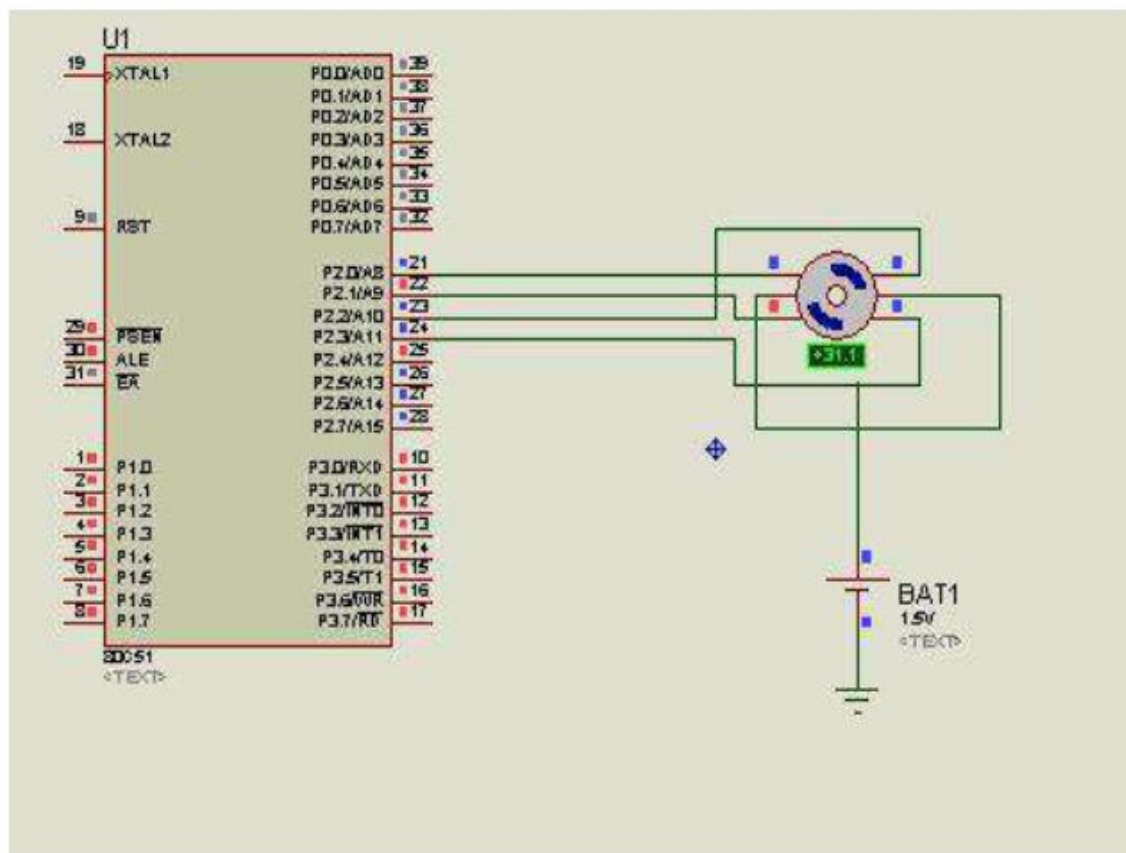
```

LOOP:    MOV P2,A
         ACALL DELAY
         RR A
         SJMP LOOP

DELAY:   MOV R5,#0AH
AGAIN:   MOV R3,#0FFH
BACK:    DJNZ R3,BACK
         DJNZ R5,AGAIN
         RET
         END

```

### SCHEMATIC DIAGRAM:



Result : Thus the interfacing stepper motor using 8051 Microcontroller has been Executed.

### **Experiment No 10**

**AIM:** Write an assembly language program to generate a square wave using 8051.

## Experiment No 10

**AIM:** Write an assembly language program to generate a square wave using 8051.

### Unit 2

#### Apparatus required:

8051 microcontroller, CRO, Resistors, (0-5V) DC Power Supply

#### THEORY:

A timer can be generalized as a multi-bit counter which increments/decrements itself on receiving a clock signal and produces an interrupt signal up on roll over. When the counter is running on the processor's clock, it is called a "Timer", which counts a predefined number of processor clock pulses and generates a programmable delay. When the counter is running on an external clock source (may be a periodic or aperiodic external signal) it is called a "Counter" itself and it can be used for counting external events.

In 8051, the oscillator output is divided by 12 using a divide by 12 network and then fed to the Timer as the clock signal. That means for an 8051 running at 12MHz, the timer clock input will be 1MHz. That means the timer advances once in every 1μs and the maximum time delay possible using a single 8051 timer is  $(2^{16}) \times (1\mu\text{s}) = 65536\mu\text{s}$ . Delays longer than this can be implemented by writing up a basic delay program using timer and then looping it for a required number of time. We will see all these in detail in next sections of this article.

Designing a delay program using 8051 timers. While designing delay programs in 8051, calculating the initial value that has to be loaded in to TH and TL registers forms a very important thing. Let us see how it is done.

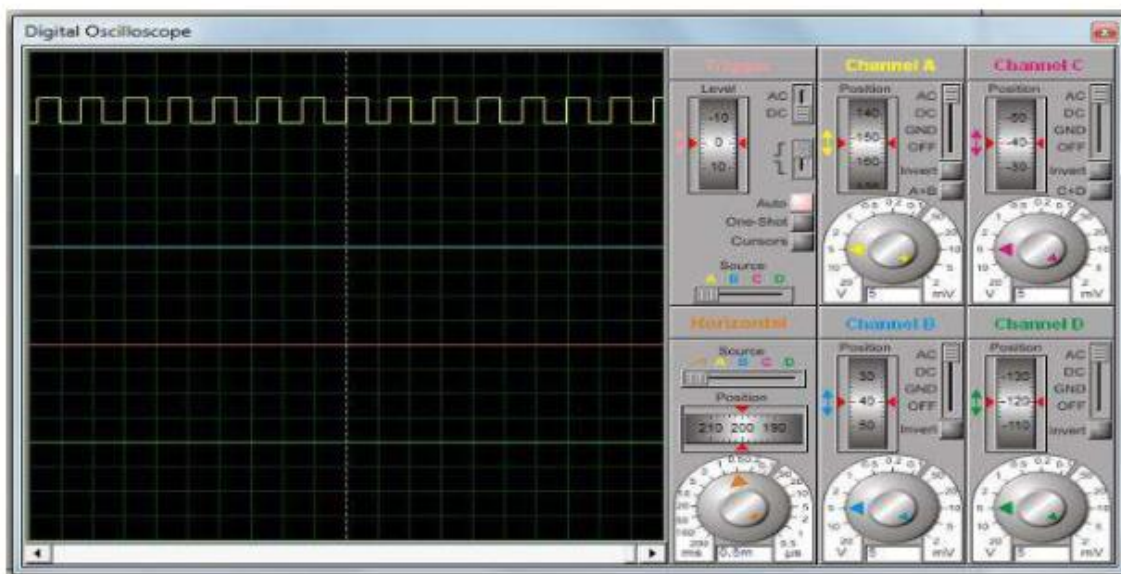
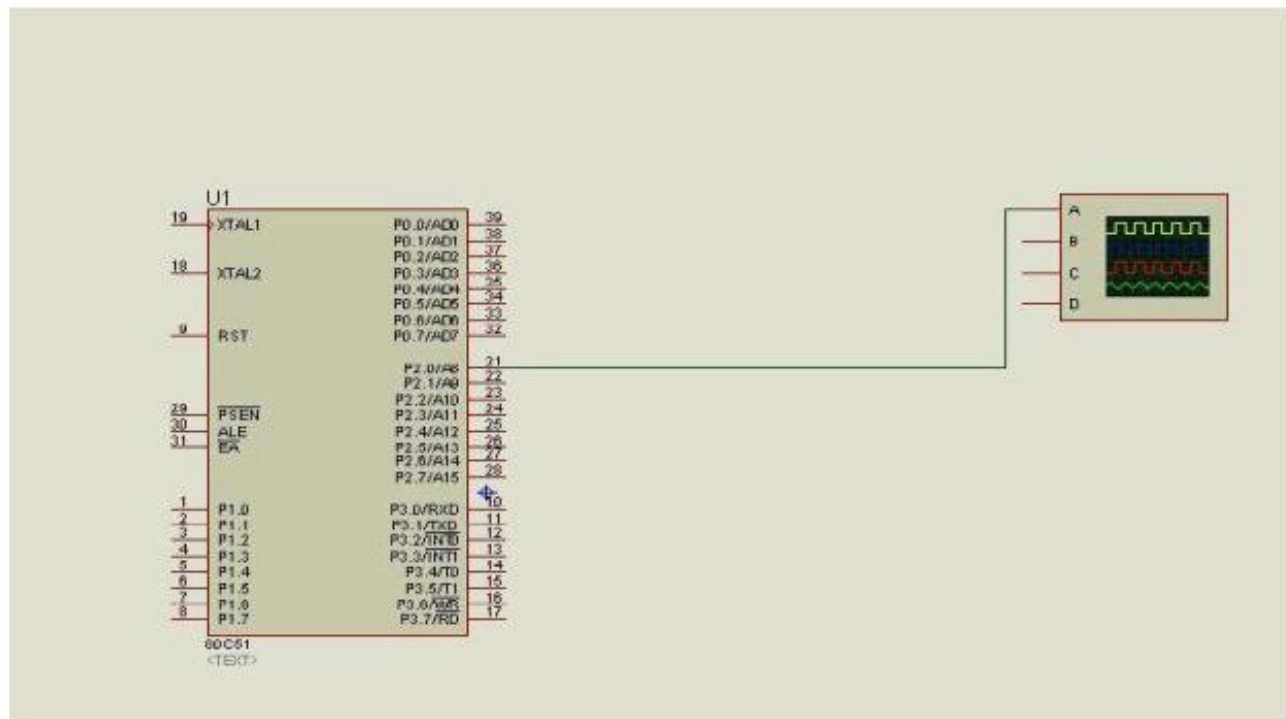
- Assume the processor is clocked by a 12MHz crystal.
- That means, the timer clock input will be  $12\text{MHz}/12 = 1\text{MHz}$
- That means, the time taken for the timer to make one increment =  $1/1\text{MHz} = 1\mu\text{s}$
- For a time delay of "X" μs the timer has to make "X" increments.
- $2^{16} = 65536$  is the maximum number of counts possible for a 16 bit timer.
- Let TH be the value that has to be loaded to TH register and TL be the value that has to be loaded to TL register.
- Then,  $\text{THTL} = \text{Hexadecimal equivalent of } (65536 - X)$  where (65536-X) is considered in decimal.



**PROGRAM:**

```
                ORG 0000H
                MOV TMOD,#20H
                MOV TH1,#1AH
MAIN:           CPL P1.3
                ACALL DELAY
                SJMP MAIN
DELAY:          SETB TR1
AGAIN:          JNB TF1,AGAIN
                CLR TR1
                CLR TF1
                RET
                END
```

**SCHEMATIC DIAGRAM:**



Result : Thus the generation a square wave using 8051 Microcontroller has been Executed.

### **Experiment No 11**

**AIM:** Write an assembly language program to display a message in LCD display.

## Experiment No 11

**AIM:** Write an assembly language program to display a message in LCD display.

### Unit 2

#### Apparatus required:

8051 microcontroller, LCD Display, Resistors, (0-5V) DC Power Supply.

#### THEORY:

16×2 LCD module is a very common type of LCD module that is used in 8051 based embedded projects. It consists of 16 rows and 2 columns of 5×7 or 5×8 LCD dot matrices. The module we are talking about here is type number JHD162A which is a very popular one. It is available in a 16 pin package with back light, contrast adjustment function and each dot matrix has 5×8 dot resolution.

Command	Function
0F	LCD ON, Cursor blinking ON, Cursor ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Cursor blinking OFF, Display ON
80	Force cursor to the beginning of 1 <sup>st</sup> line
C0	Force cursor to the beginning of 2 <sup>nd</sup> line
08	Display OFF, Cursor OFF
83	Cursor line 1 position 3
3C	Activate second line
38	Use 2 lines and 5x7 matrix
C1	Jump to second line, position1
0C	Cursor OFF, Display ON
C2	Jump to second line, position2

#### LCD initialization

The steps that have to be done for initializing the LCD display is given below and these steps are common for almost all applications.

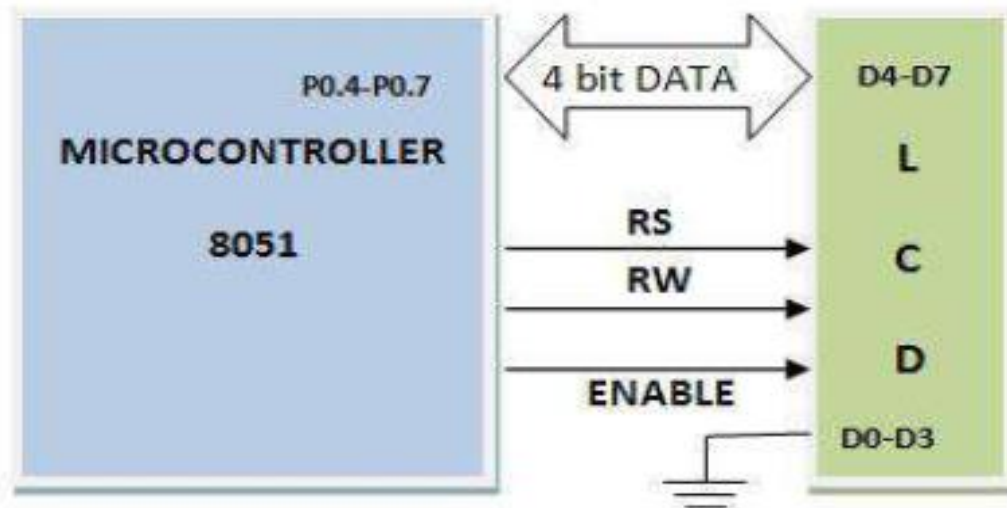
- Send 38H to the 8 bit data line for initialization
- Send 0FH for making LCD ON, cursor ON and cursor blinking ON.

- Send 06H for incrementing cursor position.
- Send 01H for clearing the display and return the cursor.

### **Sending data to the LCD**

The steps for sending data to the LCD module is given below. I have already said that the LCD module has pins namely RS, R/W and E. It is the logic state of these pins that make the module to determine whether a given data input is a command or data to be displayed.

- Make R/W low.
- Make RS=0 if data byte is a command and make RS=1 if the data byte is a data to be displayed.
- Place data byte on the data register.
- Pulse E from high to low.
- Repeat above steps for sending another data.



### **PROGRAM:**

```

ORG 0000H
MOV A,#38H
ACALL COMNWRT
ACALL DELAY
MOV A,#0EH
ACALL COMNWRT
ACALL DELAY
MOV A,#01

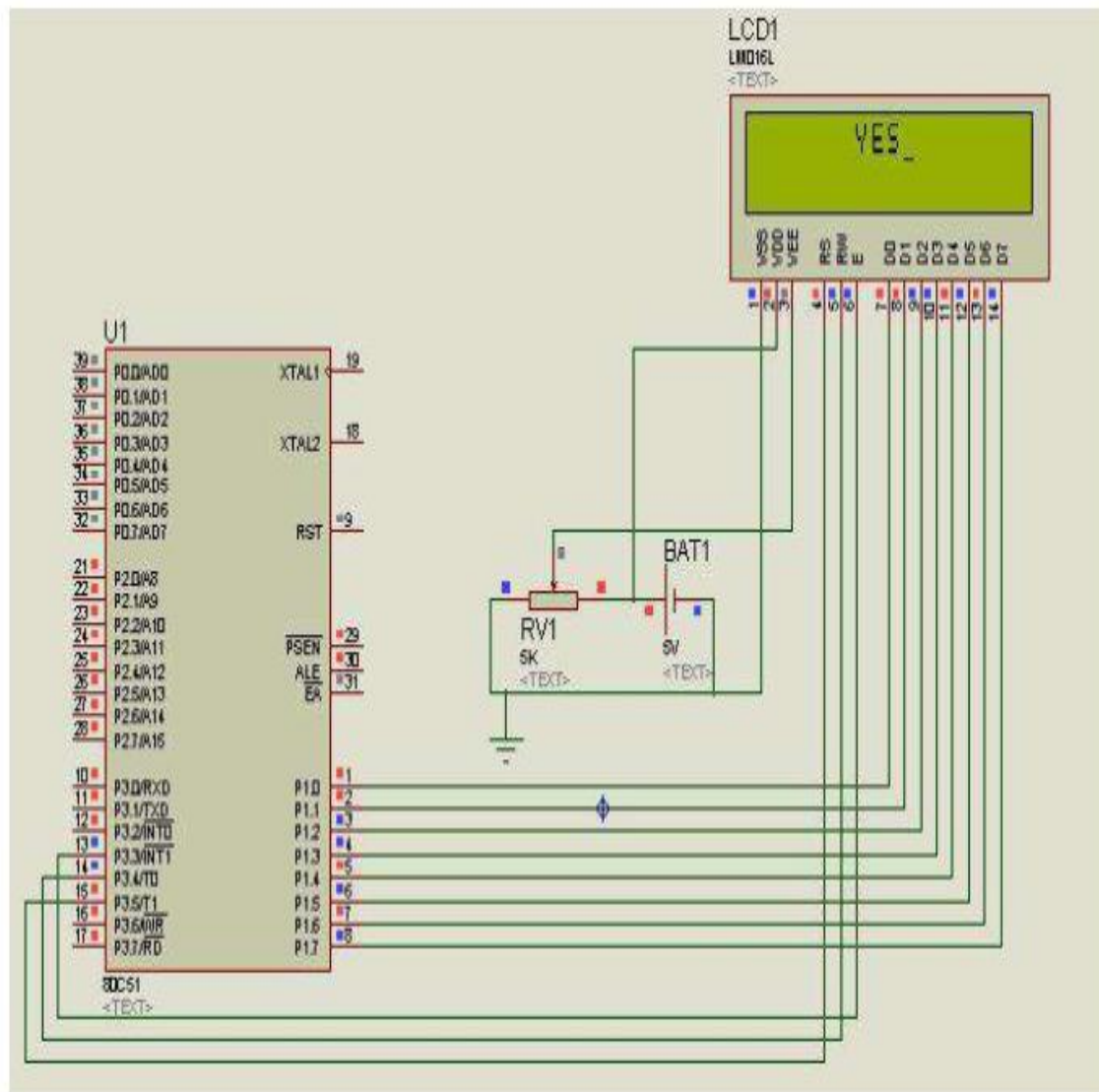
```

```

                                ACALL COMNWRT
                                ACALL DELAY
                                MOV A,#06H
                                ACALL COMNWRT
                                ACALL DELAY
                                MOV A,#86H
                                ACALL COMNWRT
                                ACALL DELAY
                                MOV A,#'Y'
                                ACALL DATAWRT
                                ACALL DELAY
                                MOV A,#'E'
                                ACALL DATAWRT
                                ACALL DELAY
                                MOV A,#'S'
                                ACALL DATAWRT
AGAIN:                          SJMP AGAIN
COMNWRT:                        MOV P1,A
                                CLR P3.5
                                CLR P3.4
                                SETB P3.3
                                ACALL DELAY
                                CLR P3.3
                                RET
DATAWRT:                        MOV P1,A
                                SETB P3.5
                                CLR P3.4
                                SETB P3.3
                                ACALL DELAY
                                CLR P3.3
                                RET

```

## SCHEMATIC DIAGRAM:



Result : Thus display a message in LCD display using 8051 Microcontroller has been Executed.

## **Experiment No. 12**

**Aim :** To Study of AVR.



## Experiment No. 12

**Aim :** To Study of AVR.

**Apparatus :** AVR **Theory:**

The advanced version of a microprocessor is a microcontroller that includes a CPU, Interrupts controller, RAM, ROM, I/O unit, etc. A microcontroller is mainly used for the operation of high-speed signal processing in an embedded system. So it performs like a major component while designing an embedded system. There are different kinds of microcontrollers available which are used based on requirements like 8051, PIC, AVR, etc.

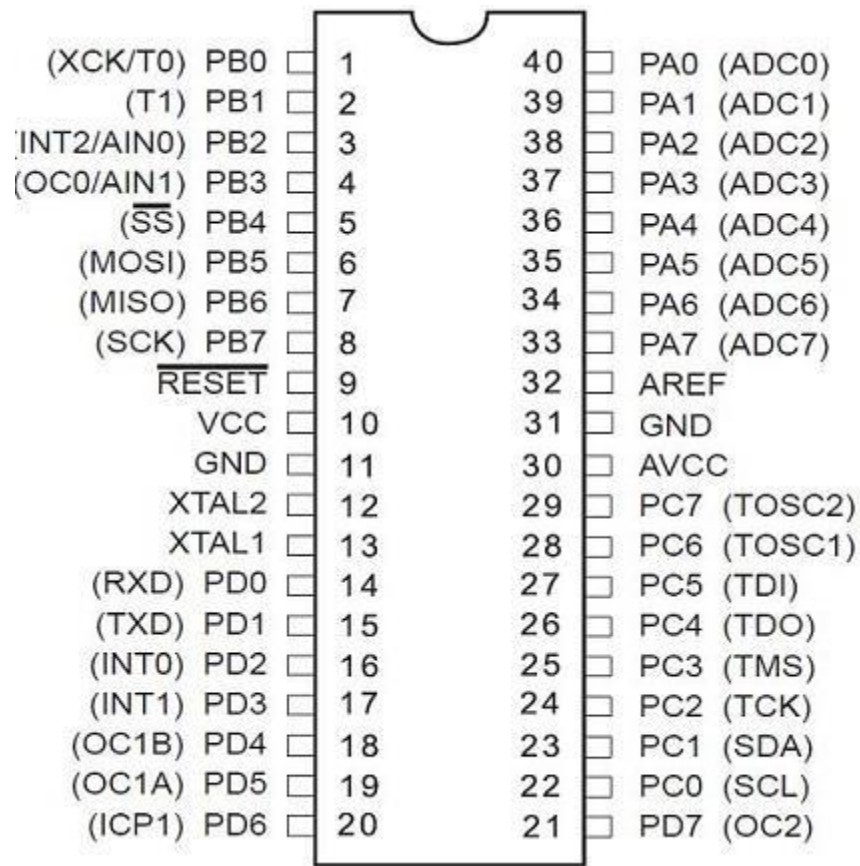
AVR microcontroller was manufactured by Atmel Corporation in 1996 and its architecture was developed by “Alf-Egil Bogen & Vegard Wollan. The name of this microcontroller was taken from its developers namely Alf-Egil Bogen & Vegard Wollan RISC microcontroller. The first microcontroller based on AVR architecture is **AT90S8515** but the first commercial microcontroller was **AT90S1200**.

The Atmel’s AVR is one of the most popular microcontroller families today. The reason behind this huge popularity is the relative ease of use and the low cost level of the microcontrollers, which can be purchased in 8-pin packages that range from \$1 to \$10 dollars.

Microcontroller includes a processor, programmable I/O peripherals & memory. The AVR microcontroller provides digital control over any kind of electrical, automotive, or mechanical system, industrial plants, different devices, electronic gadgets, etc. These microcontrollers are available in 8, 16, and 32-bit ICs. So, the most commonly used AVR microcontrollers are; ATmega8, ATmega16, ATmega32 & ATmega328 microcontrollers.

### AVR pin configuration

The AVR Atmega 32 microcontroller **pin configuration** is shown below. This microcontroller includes four ports port-A, port-B, port-C, and port-D. Port-A mainly includes pins from PA7 to PA0, port-B includes PB7 to PB0, port-C includes from PC7 to PC0 and port-D includes from PD7 to PD0.



**AVR Microcontroller Pin Configuration**

### **Port-A (PA7-PA0)**

In the above AVR microcontroller, the pins in the Port-A mainly include PA7 to PA0 which works like an 8-bit bi-directional I/O port, and also the analog inputs to the analog to digital converter, if this A/D converter is not utilized. These pins provide internal pull-up resistors.

### **Port B (PB7-PB0) & Port D (PD7-PD0)**

The pins in these two ports mainly include PB7 to PB0 and PD7-PD0. These ports are 8-bit bi-directional I/O ports including internal pull-up resistors. The output of these two port buffers mainly includes symmetrical drive characteristics including both high sink & source capability. Like inputs, the pins of this port that are pulled low externally will provide current if the resistors are activated. The pins in these two ports are tri-stated whenever a reset condition turns active, even if the CLK is not running.

### **Port C (PC7-PC0)**

The pins in Port C mainly include from PC7 to PC0 and it is an 8-bit bidirectional I/O port. If the JTAG (Joint Test Action Group) interface is allowed, the pull-up resistors on pins like PC3(TMS), PC2 (TCK) & PC5 (TDI) will be triggered even if a reset takes place.

VCC: It is a digital voltage supply pin

GND: It is a GND pin.

RESET : It is a RESET pin, used to set the ATmega32 microcontroller to its main value.

Throughout the starting of an application, this pin is to be set high for two machine revolutions.

XTAL1 : This is an input pin to the inverting oscillator amplifier & also to the internal CLK operating circuit.

XTAL2 : This pin is output from the inverting oscillator amplifier.

AVCC : This is the voltage supply pin for Port A as well as the A/D Converter. The connection of this pin must be done to VCC externally, even if the A/D converter is not utilized. If the A/D converter is utilized, then it must be connected to VCC with a low-pass filter.

AREF : This is the analog reference pin used for the analog to digital converter.

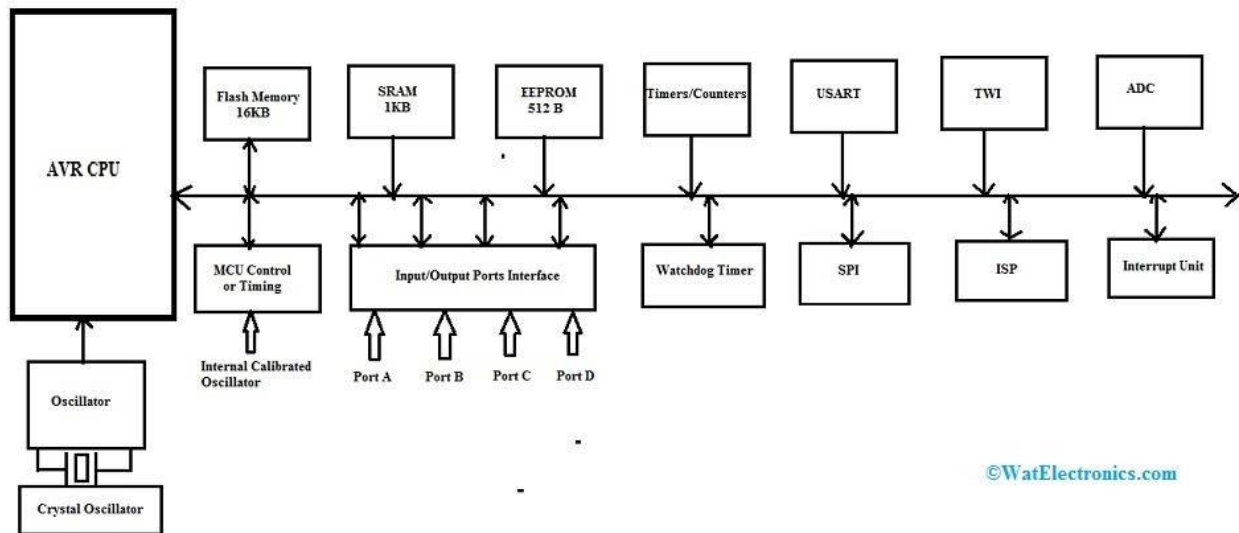
### **AVR Microcontroller Architecture**

The architecture of AVR microcontrollers is based on the advanced RISC & it includes 32 x 8-bit general-purpose registers. In a single CLK cycle, this microcontroller can get inputs from two registers to connect them to ALU for the requested operation & move back the result to an arbitrary register. Here, the ALU performs arithmetic & logical operations on the inputs from the registers.

AVR can execute single cycle execution which means this microcontroller can perform 1 million instructions for each second if the frequency of the cycle is 1MHz. If the operating

frequency of the controller is higher, then its processing speed will be higher. So the power consumption needs to optimize with processing speed & thus need to choose the operating frequency accordingly.

The architecture of the AVR microcontroller includes different building blocks and each block is explained in the AVR microcontroller block diagram shown below.



AVR Microcontroller Architecture

## I/O Ports

AVR microcontroller includes four 8-bit input-output ports like PORT-A, PORT-B, PORT-C & PORT-D.

## Internal Calibrated Oscillator

AVR microcontroller includes an internal oscillator used for driving its CLK. This microcontroller is set to work at a 1 MHz internal calibrated oscillator. So the maximum internal oscillator frequency is 8 MHz

## ADC Interface

This microcontroller includes an 8-channel ADC with a 10-bits resolution. The main function of this ADC is to read the analog input.

## **Timers/Counters**

The microcontroller includes two 8-bit & one 16-bit timer/counter. The main function of timers in this controller is to generate precision actions like time delays created in between two operations.

## **Watchdog Timer**

In this microcontroller, the watchdog timer is present with an internal oscillator. The main function of this is to monitor and reset the controller continuously if the code gets trapped while executing in a defined time interval.

## **Interrupts**

This microcontroller includes 21 interrupts where 16 interrupts are internal and the remaining interrupts are external. Here internal interrupts support different peripherals like ADC, USART, Timers, etc.

## **USART**

The term USART stands for “Universal Synchronous and Asynchronous Receiver” & interface of the transmitter is obtainable to interface with an external device that is capable of communicating serially.

## **General Purpose Registers**

This microcontroller has 32 general-purpose registers where these registers are connected with the ALU of the CPU directly.

## **Memory**

The memory of this microcontroller includes three different sections

### **Flash EEPROM**

This type of memory is helpful in storing the program dumped by the user into the AVR microcontroller. This program can be simply removed electrically like a single unit. This memory is non-volatile which means if the power is gone then the program will not erase. This microcontroller includes 16KB of in-system programmable Flash EEPROM.

### **Byte Addressable EEPROM**

Byte addressable EEPROM is a nonvolatile memory that is mainly used for data storage. This microcontroller includes EEPROM- 512 bytes, so this memory can be simply helpful in storing the lock code if we are designing an electronic door lock application.

## **SRAM**

SRAM stands for Static Random Access Memory which is the volatile memory of the AVR microcontroller so the data will be lost once power is deactivated. This microcontroller includes 1KB – of internal SRAM. A small part of SRAM is reserved for general purpose registers which are used by the CPU & also someother peripheral subsystems.

## **ISP**

These microcontrollers include In-System Programmable Flash Memory or ISP that can be simply programmed without detaching the chip from the circuit; This allows for reprogramming of the microcontroller when it is within the applicationcircuit.

## **SPI**

The term SPI stands for Serial Peripheral Interface, which is mainly used for serial communication between two different devices on a common CLK source. The SPI data transmission speed is high as compared to USART.

## **TWI**

TWI is a Two-Wire Interface that can be used to connect a network for devices, so several devices can be simply connected above this interface to form this network so that the transmission of data can be done simultaneously by devices with their own unique address.

## **DAC**

The DAC or Digital to Analog Converter in the microcontroller is used to perform the reverse action of ADC. This converter is simply used whenever there is a requirement of changing a signal from digital to analog.

**Result :** Thus we have studied the AVR.

## **Experiment No. : 13**

**Aim : To** study of Arduino.

## Experiment No. : 13

**Aim :** To study of Arduino.

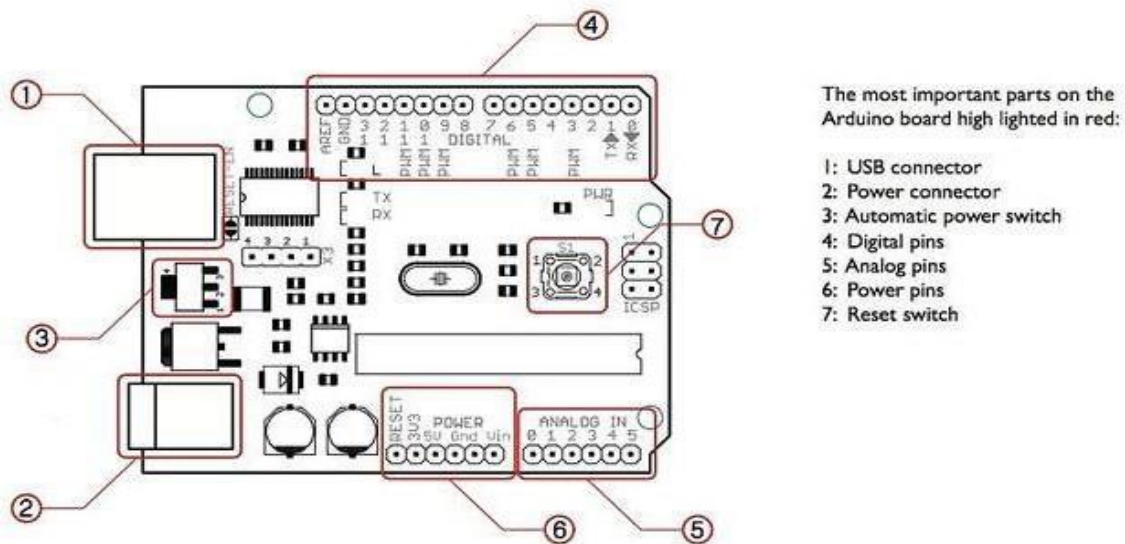
**Apparatus :** Arduino Uno

### Theory :

An Arduino board is a one type of microcontroller based kit. The first Arduino technology was developed in the year 2005 by David Cuartillas and Massimo Banzi. The designers thought to provide easy and low cost board for students, hobbyists and professionals to build devices. Arduino board can be purchased from the seller or directly we can make at home using various basic components. The best examples of Arduino for beginners and hobbyists includes motor detectors and thermostats, and simple robots. In the year 2011, Adafruit industries expected that over 3lakhs Arduino boards had been produced. But, 7lakhs boards were in user's hands in the year 2013. Arduino technology is used in many operating devices like communication or controlling.

### Arduino Technology

A typical example of the Arduino board is Arduino Uno. It includes an ATmega328 microcontroller and it has 28-pins



**Arduino Pin Diagram**



The pin configuration of the Arduino Uno board is shown in the above. It consists of 14-digital i/o pins. Wherein 6 pins are used as pulse width modulation o/p and 6 analog i/p, a USB connection, a power jack, a 16MHz crystal oscillator, a reset button, and an ICSP header. Arduino board can be powered either from the personal computer through a USB or external source like a battery or an adaptor. This board can operate with an external supply of 7-12V by giving voltage reference through the IOREf pin or through the pin Vin.

#### Digital I/Ps

It comprises of 14-digital I/O pins, each pin take up and provides 40mA current. Some of the pins have special functions like pins 0 & 1, which acts as a transmitter and receiver respectively. For serial communication, pins-2 & 3 are external interrupts, 3,5,6,9,11 pins delivers PWM o/p and pin-13 is used to connect LED.

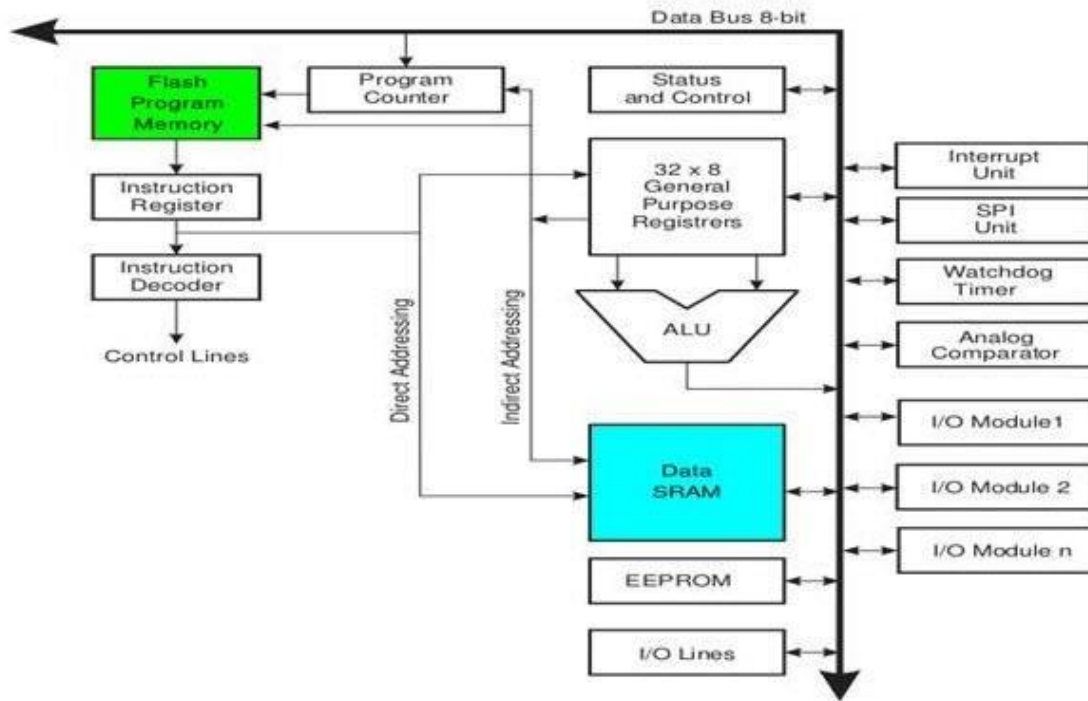
**Analog i/ps:** It has 6-analog I/O pins, each pin provide a 10 bits resolution.

**Aref:** This pin gives a reference to the analog i/ps.

**Reset:** When the pin is low, then it resets the microcontroller.

#### Arduino Architecture

Basically, the processor of the Arduino board uses the Harvard architecture where the program code and program data have separate memory. It consists of two memories such as program memory and data memory. Wherein the data is stored in data memory and the code is stored in the flash program memory. The Atmega328 microcontroller has 32kb of flash memory, 2kb of SRAM 1kb of EPROM and operates with a 16MHz clock speed.



### Arduino Architecture

**Result :** Thus we have studied Arduino.



