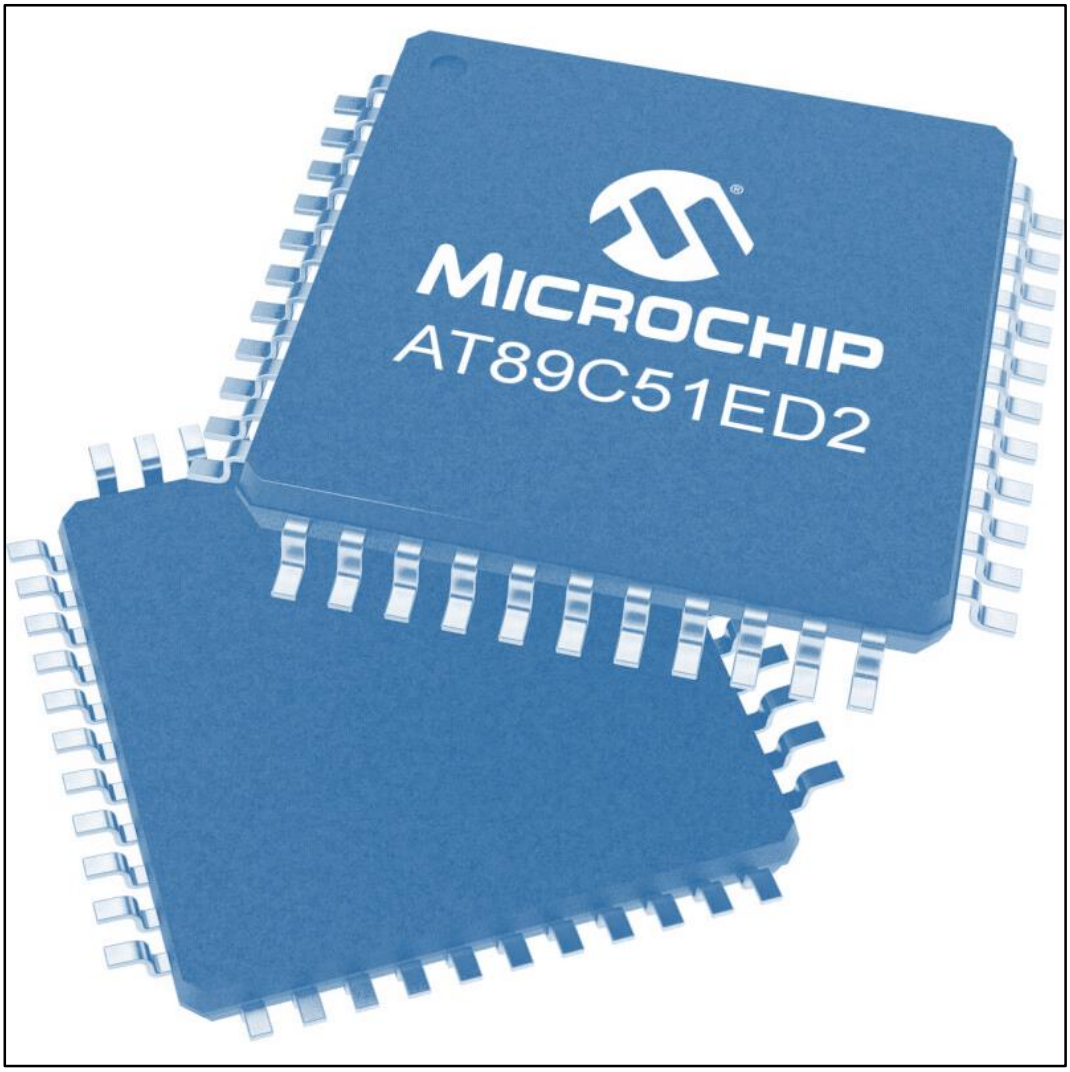
KARNATAK LAW SOCIETY'S

KLS Gogte Institute of Technology, Belagavi

LABORATORY MANUAL



MICROCONTROLLER & EMBEDDED SYSTEMS LABORATORY

V SEMESTER

(COURSE CODE: 22CSL59)

Department of Computer Science and Engineering

KLS GIT, Belagavi

LAB RULES

Always:

1. **Enter the Lab on time and leave on time.**
2. **Keep the bag outside in the respective racks.**
3. **Utilize the lab hours appropriately.**
4. **If you notice a problem with a piece of equipment (sensors or computer) or the room in general (cooling, heating or lighting) Please report to the lab staff immediately. Do not attempt by yourself.**
5. **Disconnect the devices and other peripherals connected to the computer and shut down the system before leaving the lab.**

Table of Content

Course Outcome:

At the end of the course, student will able to

1.	Explain the essential concepts governing microcontrollers and the architectural framework of embedded systems.
2.	Apply programming concepts to effectively program microcontrollers using Embedded 'C'.
3.	Analyze various peripheral devices and determine suitable interfacing methods with microcontrollers.
4.	Develop embedded systems solutions by selecting appropriate hardware components and designing circuits.
5.	Analyze the requirements for a real world problem or a specification and develop a course project as the solution.

Programme Outcomes addressed to the course

POs	Description
1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2	Problem analysis: Identify, formulate, review research literature, and analyze complex Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3	Design/development of solutions: Design solutions for complex 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.
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
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.

RUBRICS FOR CONDUCTION OF EXPERIMENTS, VIVA-VOCE & JOURNAL EVALUATION

COURSE: MICRO-CONTROLLERS AND EMBEDDED SYSTEMS LABORATORY

COURSE CODE: 22CSL59

TO BE ATTACHED FOR ALL TEN EXPERIMENTS:

Criteria	Excellent (85% - 100%)	Good (70% - 84%)	Needs Improvement (50% - 69%)	Poor (Below 50%)	Marks Obtained
Conduction of Experiments (15)	All steps followed accurately, no errors, efficient use of time	Most steps followed accurately, minor errors, good use of time	Several steps followed, some errors, average use of time	Many steps missed, numerous errors, poor use of time	
Knowledge and Understanding (5)	Demonstrates excellent understanding and can answer all questions correctly	Demonstrates good understanding, minor gaps	Basic understanding, several gaps	Poor understanding, unable to answer questions	
Journal Completeness, Calculations, and Results with Graphs and Visuals (5)	All sections completed thoroughly and accurately, detailed and accurate calculations/results, clear and relevant graphs/visuals	Most sections completed with minor errors in calculations or visuals, well-documented	Several sections completed with some inaccuracies and basic documentation, graphs/visuals have issues	Few sections completed with several inaccuracies, poorly documented, missing or poorly constructed graphs/visuals	
Signature of the Faculty:		**Evaluated on:**		**Total:**	

RUBRICS FOR CONDUCTION OF LAB PROJECT/OPEN ENDED EXPERIMENT

COURSE: MICRO-CONTROLLERS AND EMBEDDED SYSTEMS LABORATORY

COURSE CODE: 22CSL59

TO BE ATTACHED FOR ONLY OPEN ENDED EXPERIMENT/ LAB PROJECT:

Criteria	Excellent (85% - 100%)	Good (70% - 84%)	Needs Improvement (50% - 69%)	Poor (Below 50%)	Marks Obtained
Conduction of Open Ended Experiment (5)	All steps followed accurately, no errors, efficient use of time	Most steps followed accurately, minor errors, good use of time	Several steps followed, some errors, average use of time	Many steps missed, numerous errors, poor use of time	
Journal/ Report Completeness, Calculations, and Results with Graphs and Visuals (3)	All sections completed thoroughly and accurately, detailed and accurate calculations/results, clear and relevant graphs/visuals	Most sections completed with minor errors in calculations or visuals, well-documented	Several sections completed with some inaccuracies and basic documentation, graphs/visuals have issues	Few sections completed with several inaccuracies, poorly documented, missing or poorly constructed graphs/visuals	
Knowledge and Understanding (2)	Demonstrates excellent understanding and can answer all questions correctly	Demonstrates good understanding, minor gaps	Basic understanding, several gaps	Poor understanding, unable to answer questions	
Signature of the Faculty:		**Evaluated on:**		**Total:**	

INTEL 8051 MICRCONTROLLER

Introduction:

A decade back the process and control operations were totally implemented by the Microprocessors only. But now a day the situation is totally changed and it is occupied by the new devices called Microcontroller. The development is so drastic that we can't find any electronic gadget without the use of a microcontroller. This microcontroller changed the embedded system design so simple and advanced that the embedded market has become one of the most sought after for not only entrepreneurs but for design engineers also.

What is a Microcontroller?

A single chip computer or a CPU with all the peripherals like RAM, ROM, I/O Ports, Timers, ADCs etc on the same chip. For ex: Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC16X etc.

MICROPROCESSORS & MICROCONTROLLERS:

Microprocessor:

A CPU built into a single VLSI chip is called a microprocessor. It is a general-purpose device and additional external circuitry are added to make it a microcomputer. The microprocessor contains arithmetic and logic unit (ALU), Instruction decoder and control unit, Instruction register, Program counter (PC), clock circuit (internal or external), reset circuit (internal or external) and registers. But the microprocessor has no on chip I/O Ports, Timers, Memory etc. For example, Intel 8085 is an 8-bit microprocessor and Intel 8086/8088 a 16-bit microprocessor. The block diagram of the Microprocessor is shown in Fig.1

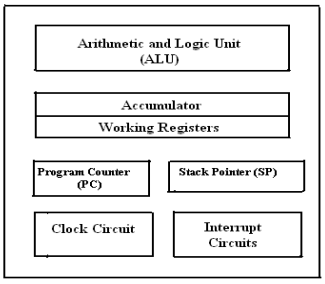


Fig.1 Block diagram of a Microprocessor.

MICROCONTROLLER:

A microcontroller is a highly integrated single chip, which consists of on chip CPU (Central Processing Unit), RAM (Random Access Memory), EPROM/PROM/ROM (Erasable Programmable Read Only Memory), I/O (input/output) – serial and parallel, timers, interrupt controller. For example, Intel 8051 is 8-bit microcontroller and Intel 8096 is 16-bit microcontroller. The block diagram of Microcontroller is shown in Fig.2.

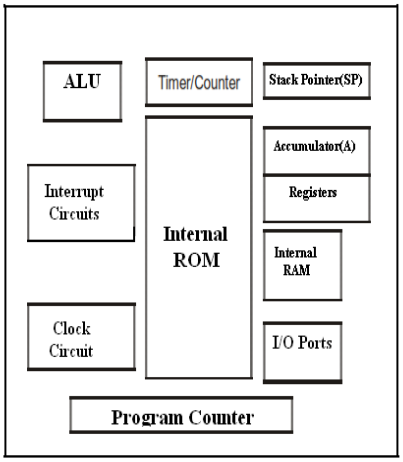


Fig.2.Block Diagram of a Microcontroller

Differences between Microprocessor and Microcontroller

Sl. No	Microprocessor	Microcontroller
1	A microprocessor is a general purpose device which is called a CPU	A microcontroller is a dedicated chip which is also called single chip computer.
2	A microprocessor do not contain on chip I/O Ports, Timers, Memories etc..	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry (in addition to CPU) in a single chip.
3	Microprocessors are most commonly used as the CPU in	Microcontrollers are used in small, minimum component designs performing

	microcomputer systems	control-oriented applications.
4	Microprocessor instructions are mainly nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor instruction sets are mainly intended for catering to large volumes of data.	Microcontrollers have instruction sets catering to the control of inputs and outputs.
6	Microprocessor based system design is complex and expensive	Microcontroller based system design is rather simple and cost effective
7	The Instruction set of microprocessor is complex with large number of instructions.	The instruction set of a Microcontroller is very simple with less number of instructions. For, ex: PIC microcontrollers have only 35 instructions**.**
8	A microprocessor has zero status flag	A microcontroller has no zero flag.

INTEL 8051 MICRCONTROLLER**:**

The 8051 microcontroller is a very popular 8-bit microcontroller introduced by Intel in the year 1981 and it has become almost the academic standard now a day. The 8051 is based on an 8-bit CISC core with Harvard architecture. Its 8-bit architecture is optimized for control applications with extensive Boolean processing. It is available as a 40-pin DIP chip and works at +5 Volts DC. The salient features of 8051 controllers are given below.

SALIENT FEATURES: The salient features of 8051 Microcontroller are

- 4 KB on chip program memory (ROM or EPROM)).
- 128 bytes on chip data memory(RAM).
- 8-bit data bus
- 16-bit address bus
- 32 general purpose registers each of 8 bits
- Two -16 bit timers T_0 and T_1
- Five Interrupts (3 internal and 2 external).
- Four Parallel ports each of 8-bits (PORT0, PORT1, PORT2, PORT3) with a total of 32 I/O lines.
- One 16-bit program counter and One 16-bit DPTR (data pointer)
- One 8-bit stack pointer
- One Microsecond instruction cycle with 12 MHz Crystal.
- One full duplex serial communication port.

ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER:

The architecture of the 8051 microcontroller can be understood from the block diagram. It has Harvard architecture with RISC (Reduced Instruction Set Computer) concept. The block diagram of 8051 microcontrollers is shown in Fig 3. below1.It consists of an 8-bit ALU, one 8-bit PSW (Program Status Register), A and B registers, one 16-bit Program counter , one 16-bit Data pointer register(DPTR),128 bytes of RAM and 4kB of ROM and four parallel I/O ports each of 8-bit width.

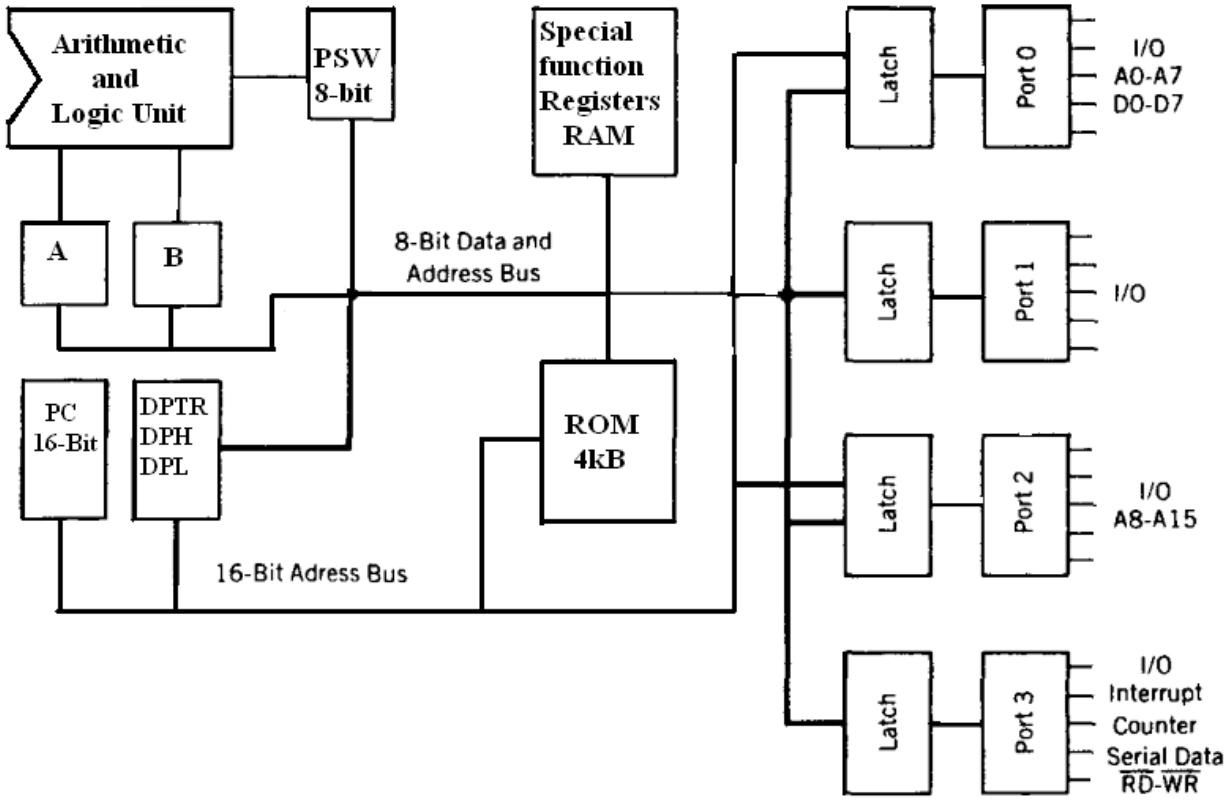


Fig.3. Block Diagram of 8051 Microcontroller

8051 has 8-bit ALU which can perform all the 8-bit arithmetic and logical operations in one machine cycle. The ALU is associated with two registers A & B

A and B Registers: The A and B registers are special function registers which hold the results of many arithmetic and logical operations of 8051.The A register is also called the **Accumulator** and as its name suggests, is used as a general register to accumulate the results of a large number of instructions. By default, it is used for all mathematical operations and also data transfer operations between CPU and any external memory.

The B register is mainly used for multiplication and division operations along with A register.

MUL AB : DIV AB.

It has no other function other than as a location where data may be stored.

The R registers: The "R" registers are a set of eight registers that are named R0, R1, etc. up to

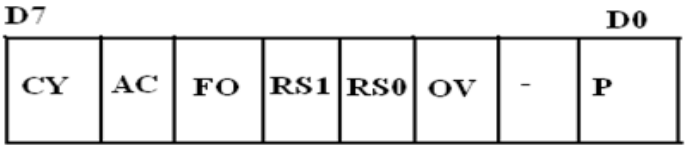
and including R7. These registers are used as auxiliary registers in many operations. The "R" registers are also used to temporarily store values.

Program Counter (PC): 8051 has a 16-bit program counter. The program counter always points to the address of the next instruction to be executed. After execution of one instruction the program counter is incremented to point to the address of the next instruction to be executed. It is the contents of the PC that are placed on the address bus to find and fetch the desired instruction. Since the PC is 16-bit width, 8051 can access program addresses from 0000H to FFFFH, a total of 6kB of code.

Stack Pointer Register (SP): It is an 8-bit register which stores the address of the stack top. i.e. the Stack Pointer is used to indicate where the next value to be removed from the stack should be taken from. When a value is pushed onto the stack, the 8051 first increments the value of SP and then stores the value at the resulting memory location. Similarly, when a value is popped off the stack, the 8051 returns the value from the memory location indicated by SP, and then decrements the value of SP. Since the SP is only 8-bit wide it is incremented or decremented by one. SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI. It is also used intrinsically whenever an interrupt is triggered.

Data Pointer Register (DPTR): It is a 16-bit register which is the only user-accessible. DPTR, as the name suggests, is used to point to data. It is used by a number of commands which allow the 8051 to access external memory. When the 8051 accesses external memory it will access external memory at the address indicated by DPTR. This DPTR can also be used as two 8-registers DPH and DPL.

Program Status Register (PSW): The 8051 has an 8-bit PSW register which is also known as Flag register. In the 8-bit register only 6-bits are used by 8051.The two unused bits are user definable bits. In the 6-bits four of them are conditional flags. They are Carry –CY, Auxiliary Carry-AC, Parity-P, and Overflow-OV. These flag bits indicate some conditions that resulted after an instruction was executed.



The bits PSW3 and PSW4 are denoted as RS0 and RS1 and these bits are used to select the bank registers of the RAM location. The meaning of various bits of PSW register is shown below.

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
FO	PSW.5	Flag 0 available for general purpose.
RS1	PSW.4	Register Bank select bit 1
RS0	PSW.3	Register bank select bit 0
OV	PSW.2	Overflow flag
---	PSW.1	User defined flag
P	PSW.0	Parity flag. set/cleared by hardware.

The selection of the register Banks and their addresses are given below.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

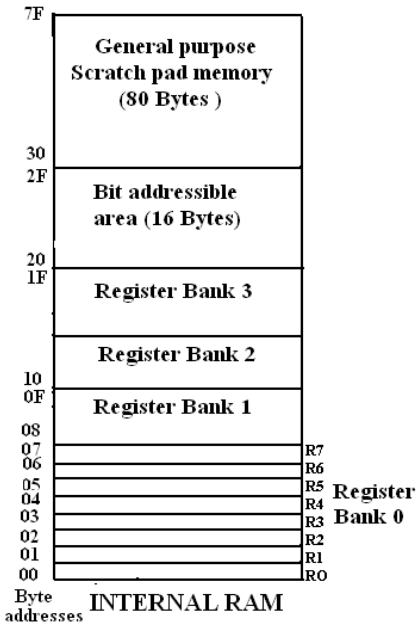
Memory organization: The 8051 microcontroller has 128 bytes of Internal RAM and 4kB of on chip ROM. The RAM is also known as Data memory and the ROM is known as program memory. The program memory is also known as Code memory. This Code memory holds the

actual 8051 program that is to be executed. In 8051 this memory is limited to 64K. Code memory may be found on-chip, as ROM or EPROM. It may also be stored completely off-chip in an external ROM or, more commonly, an external EPROM. The 8051 has only 128 bytes of Internal RAM but it supports 64kB of external RAM. As the name suggests, external RAM is any random access memory which is off-chip. Since the memory is off-chip it is not as flexible in terms of accessing, and is also slower. For example, to increment an Internal RAM location by 1, it requires only 1 instruction and 1 instruction cycle but to increment a 1-byte value stored in External RAM requires 4 instructions and 7 instruction cycles. So, here the external memory is 7 times slower.

Internal RAM OF 8051:

This Internal RAM is found on-chip on the 8051. So it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying its contents. Internal RAM is volatile, so when the 8051 is reset this memory is cleared. The 128 bytes of internal RAM is organized as below.

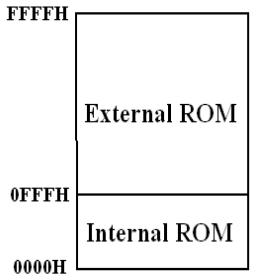
- Four register banks (Bank0, Bank1, Bank2 and Bank3) each of 8-bits (total 32 bytes). The default bank register is Bank0. The remaining Banks are selected with the help of RS0 and RS1 bits of PSW Register.
- 16 bytes of bit addressable area and
- 80 bytes of general purpose area (Scratch pad memory) as shown in the diagram below. This area is also utilized by the microcontroller as a storage area for the operating stack.



The 32 bytes of RAM from address 00 H to 1FH are used as working registers organized as four banks of eight registers each. The registers are named as R0-R7. Each register can be addressed by its name or by its RAM address.

For EX: MOV A, R7 or MOV R7, #05H

Internal ROM (On –chip ROM): The 8051 microcontroller has 4kB of on chip ROM but it can be extended up to 64kB.This ROM is also called program memory or code memory. The CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data. The external ROM is accessed when the EA (active low) pin is connected to ground or the contents of program counter exceeds 0FFFH.When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.



SPECIAL FUNCTION REGISTERS (SFRs): In 8051 microcontrollers their certain registers which uses the RAM addresses from 80h to FFh and they are meant for certain specific operations. These registers are called Special function registers (SFRs). Some of these registers are bit addressable also.

The list of SFRs and their functional names are given below. In these SFRs some of them are related to I/O ports (P0, P1, P2 and P3) and some of them are meant for control operations (TCON, SCON, PCON.) and remaining are the auxiliary SFRs, in the sense that they don't directly configure the 8051.

S.No	Symbol		Name of SFR	Address (Hex)
1	ACC*		Accumulator	**0E0**
2	B*		B-Register	**0F0**
3	PSW*		Program Status word register	**0DO**
4	SP		Stack Pointer Register	**81**
5	DPTR	DPL	Data pointer low byte	**82**
		DPH	Data pointer high byte	**83**
6	P0*		Port 0	**80**
	P1*		Port 1	**90**
8	P2*		Port 2	**0A**
9	P3*		Port 3	**0B**
10	IP*		Interrupt Priority control	**0B8**
11	IE*		Interrupt Enable control	**0A8**
12	TMOD		Tmier mode register	**89**
13	TCON*		Timer control register	**88**
14	TH0		Timer 0 Higher byte	**8C**

15	TL0	Timer 0 Lower byte	**8A**
16	TH1	Timer 1Higher byte	**8D**
17	TL1	Timer 1 lower byte	**8B**
18	SCON*	Serial control register	**98**
19	SBUF	Serial buffer register	**99**
20	PCON	Power control register	**87**

The * indicates the bit addressable SFRs

Table:SFRs of 8051 Microcontroller

PARALLEL I /O PORTS:

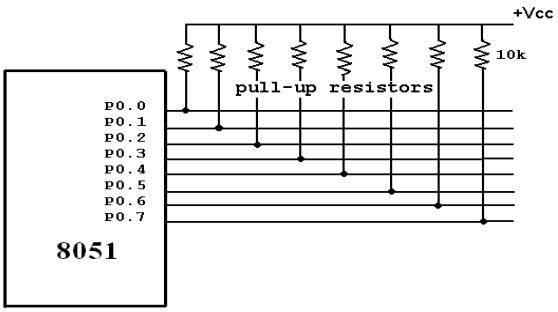
The 8051 microcontroller has four parallel I/O ports, each of 8-bits. So, it provides the user 32 I/O lines for connecting the microcontroller to the peripherals. The four ports are P0 (Port 0), P1 (Port1), P2(Port 2) and P3 (Port3). Upon reset all the ports are output ports. In order to make them input, all the ports must be set i.e. a high bit must be sent to all the port pins. This is normally done by the instruction "SETB".

```
Ex: MOV A, #0FFH        ;  A =  FF
    MOV P0,A            ; make P0 an input port
```

PORT 0:

Port 0 is an 8-bit I/O port with dual purpose. If external memory is used, these port pins are used for the lower address byte address/data (AD_0-AD_7), otherwise all bits of the port are either input or output. Unlike other ports, Port 0 is not provided with pull-up resistors internally, so for PORT0 pull-up resistors of nearly 10k are to be connected externally as shown in the fig.2.

Dual role of port 0: Port 0 can also be used as address/data bus (AD0-AD7), allowing it to be used for both address and data. When connecting the 8051 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save the pins. ALE indicates whether P0 has address or data. When ALE = 0, it provides data D0-D7, and when ALE =1 it provides address and data with the help of a 74LS373 latch.



Port 1: Port 1 occupies a total of 8 pins (pins 1 through 8). It has no dual application and acts only as input or output port. In contrast to port 0, this port does not need any pull-up resistors since pull-up resistors connected internally. Upon reset, Port 1 is configured as an output port. To configure it as an input port, port bits must be set i.e. a high bit must be sent to all the port pins. This is normally done by the instruction "SETB".

For Ex : MOV A, #0FFH ; A=FF HEX

MOV P1,A ; make P1 an input port by writing 1's to all of its pins

Port 2 : Port 2 is also an eight bit parallel port. (pins 21- 28). It can be used as input or output port. As this port is provided with internal pull-up resistors it does not need any external pull-up resistors. Upon reset, Port 2 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For ex,

MOV A, #0FFH ; A=FF hex

MOV P_2, A ; make P_2 an input port by writing all 1's to it

Dual role of port 2 : Port2 lines are also associated with the higher order address lines A8-A15. In systems based on the 8751, 8951, and DS5000, Port2 is used as simple I/O port.. But, in 8031-based systems, port 2 is used along with P0 to provide the 16-bit address for the external memory. Since an 8031 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address. In other words, when 8031 is connected to external memory, Port 2 is used for the upper 8 bits of the 16 bit address, and it cannot be used for I/O operations.

PORT 3 : Port3 is also an 8-bit parallel port with dual function.(pins 10 to 17). The port pins can be used for I/O operations as well as for control operations. The details of these

additional operations are given below in the table. Port 3 also do not need any external pull-up resistors as they are provided internally similar to the case of Port2 & Port 1. Upon reset port 3 is configured as an output port . If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For ex,

MOV A, #0FFH ; A= FF hex

MOV P_3, A ; make P_3 an input port by writing all 1's to it

Alternate Functions of Port 3 : P3.0 and P3.1 are used for the RxD (Receive Data) and TxD (Transmit Data) serial communications signals. Bits P3.2 and P3.3 are meant for external interrupts. Bits P3.4 and P3.5 are used for Timers 0 and 1 and P3.6 and P3.7 are used to provide the write and read signals of external memories connected in 8031 based systems

S.No	Port 3 bit	Pin No	Function
1	P3.0	10	RxD
2	P3.1	11	TxD
3	P3.2	12	$\overline{\text{INT0}}$
4	P3.3	13	$\overline{\text{INT1}}$
5	P3.4	14	T0
6	P3.5	15	T1
7	P3.6	16	$\overline{\text{WR}}$
8	P3.7	17	$\overline{\text{RD}}$

Table: PORT 3 alternate functions

Interrupt Structure: An interrupt is an external or internal event that disturbs the microcontroller to inform it that a device needs its service. The program which is associated with the interrupt is called the **interrupt service routine** (ISR) or **interrupt handler**. Upon receiving the interrupt signal the Microcontroller, finish current instruction and saves the PC on stack. Jumps to a fixed location in memory depending on type of interruptStarts to execute the interrupt service routine until RETI (return from interrupt) Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack

The 8051 microcontroller has **FIVE** interrupts in addition to Reset. They are

- Timer 0 overflow Interrupt
- Timer 1 overflow Interrupt
- External Interrupt 0(INT0)
- External Interrupt 1(INT1)
- Serial Port events (buffer full, buffer empty, etc) Interrupt

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

- External Interrupt 0: 0003 H
- Timer 0 overflow: 000B H
- External Interrupt 1: 0013 H
- Timer 1 overflow: 001B H
- Serial Interrupt : 0023 H

Upon reset all Interrupts are disabled & do not respond to the Microcontroller. These interrupts must be enabled by software in order for the Microcontroller to respond to them. This is done by an 8-bit register called Interrupt Enable Register (IE).

Interrupt Enable Register :

EA	—	ET2	ES	ET1	EX1	ET0	EX0

- EA : Global enable/disable. To enable the interrupts this bit must be set High.
- --- : Undefined-reserved for future use.
- ET2 : Enable /disable Timer 2 overflow interrupt.
- ES : Enable/disable Serial port interrupt.
- ET1 : Enable /disable Timer 1 overflow interrupt.
- EX1 : Enable/disable External interrupt1.

- ET0 : Enable /disable Timer 0 overflow interrupt.
- EX0 : Enable/disable External interrupt0

Upon reset the interrupts have the following priority.(Top to down). The interrupt with the highest PRIORITY gets serviced first.

1. External interrupt 0 (INT0)
2. Timer interrupt0 (TF0)
3. External interrupt 1 (INT1)
4. Timer interrupt1 (TF1)
5. Serial communication (RI+TI)

Priority can also be set to “high” or “low” by 8-bit IP register.- Interrupt priority register

—	—	PT2	PS	PT1	PX1	PT0	PX0

IP.7: reserved

IP.6: reserved

IP.5: Timer 2 interrupt priority bit (8052 only)

IP.4: Serial port interrupt priority bit

IP.3: Timer 1 interrupt priority bit

IP.2: External interrupt 1 priority bit

IP.1: Timser 0 interrupt priority bit

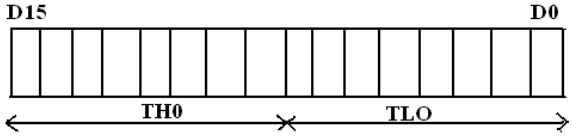
IP.0: External interrupt 0 priority bit

TIMERS in 8051 Microcontrollers : The 8051 microcontroller has two 16-bit timers Timer 0 (T0) and Timer 1(T1) which can be used either to generate accurate time delays or as event counters. These timers are accessed as two 8-bit registers TLO, THO & TL1 ,TH1 because the 8051 microcontroller has 8-bit architecture.

TIMER 0 : The Timer 0 is a 16-bit register and can be treated as two 8-bit registers (TL0 & TH0) and these registers can be accessed similar to any other registers like A,B or R1,R2,R3 etc…

Ex : The instruction Mov TL0,#07 moves the value 07 into lower byte of Timer0.

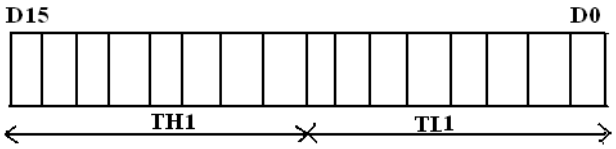
Similarly Mov R5,TH0 saves the contents of TH0 in the R5 register.



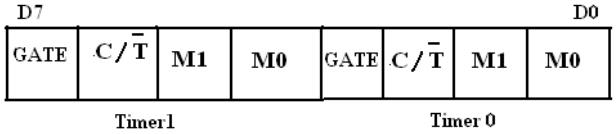
TIMER 1 : The Timer 1 is also a 16-bit register and can be treated as two 8-bit registers (TL1 & TH1) and these registers can be accessed similar to any other registers like A,B or R1,R2,R3 etc…

Ex : The instruction MOV TL1,#05 moves the value 05 into lower byte of Timer1.

Similarly MOV R0,TH1 saves the contents of TH1 in the R0 register



TMOD Register : The various operating modes of both the timers T0 and T1 are set by an 8-bit register called TMOD register**.** In this TMOD register the lower 4-bits are meant for Timer 0 and the higher 4-bits are meant for Timer1.



GATE: This bit is used to start or stop the timers by hardware .When GATE= 1 ,the timers can be started / stopped by the external sources. When GATE= 0, the timers can be started or stopped by software instructions like SETB TR0 or SETB TR1

C/T (clock/Timer) : This bit decides whether the timer is used as delay generator or event counter. When **C/T = 0 ,**the Timer is used as delay generator and if C/T=1 the timer is used as an event counter. The clock source for the time delay is the crystal frequency of 8051.

M1,M0 (Mode) : These two bits are the timer mode bits. The timers of the 8051 can be configured in three modes.Mode0, Mode1 and Mode2.The selection and operation of the modes is shown below.

S.No	M0	M1	Mode	Operation
1	0	0	0	13-bit Timer mode 8-bit Timer/counter. THx with TLx as 5-bit prescalar
2	0	1	1	16-bit Timer mode.16-bit timer /counter without pre-scalar
3	1	0	2	8-bit auto reload. THx contains a value that is to be loaded into TLx each time it overflows
4	1	1	3	Split timer mode

2. CONTROLLER SPECIFICATIONS:

2.1 AT89C51ED2:

- 80C51 Central Processing Unit

  -64K bytes On-chip Flash Program/Data Memory.

  -256 bytes scratch pad RAM.

  -On-chip 1792 bytes Expanded Ram(XRAM)

- In-System Programmable (ISP) Flash memory
- 12-clock operation with selectable 6-clock operation (via In-System Programming or via parallel programmer)
- On-chip 2048 Bytes EEPROM Block for Data Storage – 100K Write Cycle
- Power control modes:
  - Clock can be stopped and resumed
  -Idle mode
  -Power-down mode
- Two speed ranges
  - 0 to 20 MHz with 6-clock operation
  - 0 to 33 MHz with 12-clock operation
- Dual Data Pointers
- Four interrupt priority levels
- Eight interrupt sources
- Four 8-bit I/O ports
- Watchdog timer.
- Full-duplex enhanced UART
  - Framing error detection
  - Automatic address recognition Three 16-bit timers/counters T0, T1 (standard 80C51) and Additional T2 (capture and compare)

- Serial Peripheral Interface Module (SPI)
- Programmable Counter Array
- Asynchronous port reset
- Low EMI (inhibit ALE, 6-clock mode)
- Wake-up from Power Down by an external interrupt//

3. KEIL SIMULATOR:

I. CREATION OF NEW PROJECT IN KEILUV3 IDE:

1. Create a new folder before creating project.
2. Open **Keil uVision4 IDE** software by double clicking on "Keil uVision 4" icon
3. Go to **Project menu** & select **New Project**, navigate to the desired project folder & give the project name in the File Name window. Click Save.
4. Select device for target window will open, click on **Atmel** to drop down the menu, select **AT89C51ED2** and press OK. Another window opens asking to add startup files, click YES button, to add the "Startup.a51" file.
5. Right click on **Target1** in Project Window & select **"Options for Target 'Target1'".**
    - In '**Target**' field select **Xtal (MHz): 11.0592**
    - Check the box Use **On-chip ROM (0x0-0xFFFF)**
    - Check the box Use **On-chip XRAM (0x0-0x1EFF)**
    - In '**Output**' window check the box **"Create HEX File"**
6. Go to File menu, click on New to open an editor window. Create your source file(s) and use the **header file 'at89c51xd2.h'** in the source file and save the file(s). (Color syntax highlighting will be enabled once the file is saved with a recognized extension such as .c).

7. Right click on **"Source Group 1"** and select the option **Add Files to Group 'Source Group 1'** and add the .c source file(s) to the group.
8. After adding source file(s) go to **"Project" -> "Translate"** to compile the file(s). Go to **"Project" - > "Build Target"** for building all source files such as **.c** files, **.asm** files, **.h** files etc. which have been added to the target build. This will create the .HEX file to be downloaded to the target device.

II. PROGRAM DOWNLOADING:

1. Press the **reset switch** SW1 on 8051 Microcontroller board.
2. Open the **Atmel Flip 2.4.2** tool by double clicking on "Atmel Flip" icon.
3. Go to **"Device" option -> "Select",** select the specific device **"AT89C51ED2"** & press OK.
4. Go to **"File" -> "Load HEX File",** navigate to desired .HEX file of the project.
5. Go to **"Settings" option -> "Communication" -> "RS232",** a window will open, and make sure that no other application is using COM port. Click on COM and select the proper COM port (E.g.: **COM1**), click on Connect.
6. In "Operations Flow" region, check the options "Erase", "Blank Check", "Program" & "Verify".
7. **Select Start Application.**

Caution: Do not reset the device during Flash Programming.

Term work 1: Simple I/O Programming:

Objective: To send the values 00 to FF on Port P1.

Theory:

Programming 8051 in 'C': The reasons for writing programs in C

- It is easier and less time consuming to write in C than Assembly
- C is easier to modify and update
- You can use code available in function libraries
- C code is portable to other microcontroller with little or no modification

Data Types:

A good understanding of C data types for 8051 can help programmers to create smaller hex files

- unsigned char
- signed char
- unsigned int
- signed int
- sbit (single bit)
- bit and sfr

Data Type	Size (Bits)	Range	Description
Unsigned char	8	0 to 255	Stores only positive values (0 to 255).
Signed char	8	-128 to 127	Stores both positive and negative values.
Unsigned int	16	0 to 65535	Stores only positive integers.
Signed int	16	-32,768 to 32,767	Stores both positive and negative integers.
Sbit	1	0 or 1	Used to access a single bit in SFR (Special Function Register).
Bit	1	0 or 1	Bit-addressable data type; used in bit-addressable RAM (20H-2FH).
SFR (Special Function Register)	8	8-bit register (e.g., P0, P1, P2, P3, TCON, etc.)	Used to control 8051 peripherals like timers, ports, and serial communication.

Example: Develop an 8051 C program to send values 00 – FF to port P1.

```
#include <reg51.h>

void main(void)

{

unsigned char z;

for (z=0;z<=255;z++)

P1=z;

}
```

Result:

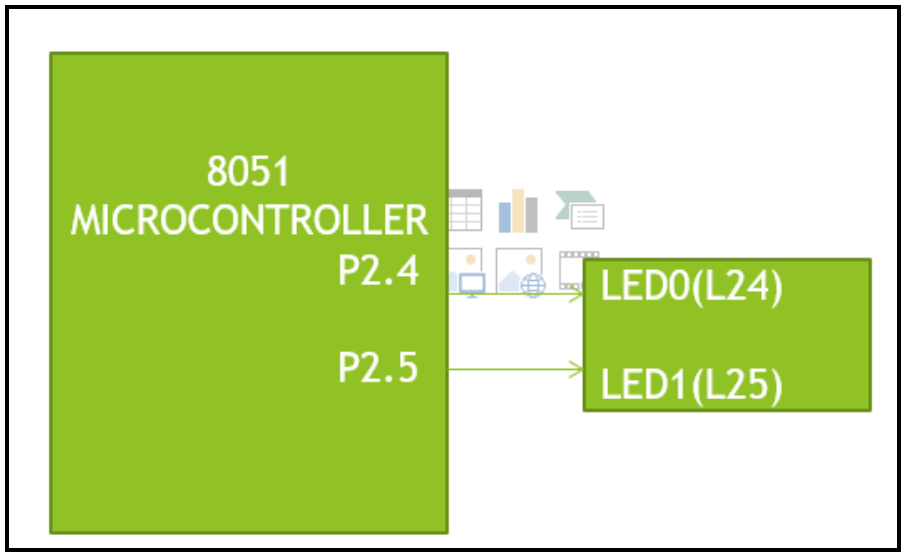
Observe the values 0 to 255 on P1.

TERMWORK 2: LED Interfacing

Objective: To implement MOD-4 counter on LEDs connected to Port 2 using Software delay.

Theory: Mod-4 counter consists of 4 states i.e., 00,01,10,11. It can be displayed on two led's connected to P2.5 and P2.4. Between each count delay has to be included. The delay has to be generated using for loop (software delay)

Interfacing Diagram:



Algorithm:

STEP 1 : INCLUDE THE HEADER FILE ''at89c51ed2.h''

STEP 2 : DECLARE THE DELAY ROUTINE

STEP 3 : DECLARE VARIABLES i, j, itime

STEP 4 : BEGIN MAIN

STEP 5 : REPEAT LOOP FOREVER USING WHILE(1)

STEP 6: SEND THE VALUE 0X00, 0X10,0X20,0X30 ON P2

STEP 7: CALL DELAY BETWEEN EACH VALUE

STEP 8: END

Code:

```
#include "at89c51ed2.h“
void delay(unsigned int);
void main(void)
{
  while(1)
  {
        P2=0x00;
        delay(250);
        P2=0x10;
        delay(250);
        P2=0x20;
        delay(250);
        P2=0x30;
        delay(250);
    }
}
//SOFTWARE DELAY GENERATION
```

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

CONNECTION DETAILS: PORT 2 to CN11

Result: Observe the count 00,01,10,11 on the 4 LEDs connected to P2.

TERMWORK 3: Timer Programming

Objectives:

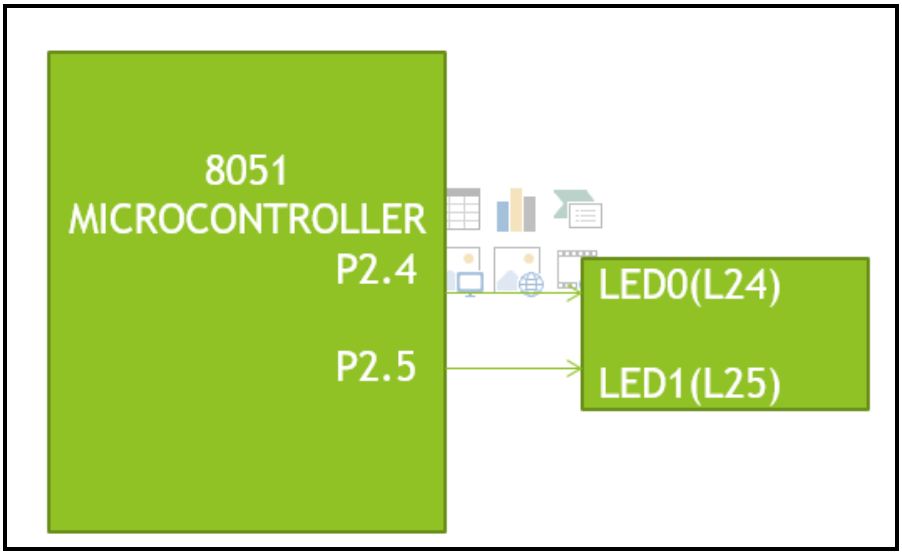
1. **To demonstrate the interfacing of LEDs connected to Port2 of 8051 Microcontroller and Timer Programming.**
2. **To develop an 8051 'C' code to display the MOD-4 count 00,01,10,11 on LEDs connected to Port2**

Theory: There are two Timers/ Counters T0 and T1 available in 8051. The SFRs associated with counter programming are: **TCON** and **TMOD** registers. Mod-4 counter consists of 4 states i.e., 00,01,10,11. It can be displayed on two led's connected to P2.5 and P2.4. Between each count delay has to be included. The delay has to be generated using Timer (Hardware delay)

Delay Calculation using Timer0 in Mode 1: The delay is calculated using Timer 0 in Mode 1 as shown below, assuming XTAL=11.0592MHz.

Delay=(FFFF-YYXX+1)x1.085µsec
Where YY = COUNT TO BE LOADED INTO TH0
XX= COUNT TO BE LOADED INTO TL0
1.085µsec =clock cycles per machine cycle/XTAL frequency
=12/11.0592MHz
FOR COUNT =4BFE
DELAY=(FFFF-4BFE+1)x1.085µsec
=50ms
NOTE: 50ms DELAY IS NOT ENOUGH TO VIEW THE OUTPUT.
CALL DELAY ROUTINE MULTIPLE TIMES TO INCLUDE ENOUGH DELAY

Interfacing Diagram:



CONNECTION DETAILS: PORT 2 to CN11

Result: Observe the count 00,01,10,11 on the 4 LEDs connected to P2.

TERMWORK 4: Counter Programming

Objective: To program counters of 8051 Microcontroller using Embedded 'C'.

Theory: There are two Timers/ Counters T0 and T1 available in 8051. The SFRs associated with counter programming are: **TCON** and **TMOD** registers.

Steps to program Counter in Mode 1:

1. Load the TMOD value register indicating which counter (counter 0 or counter1) is to be used and which counter mode (0 or 1) is selected

2. Load registers TL and TH with initial count value

3. Start the counter

4. Keep monitoring the timer flag (TF) with the while(TF==0); instruction to see if it is raised. Get out of the loop when TF becomes high

5. Stop the counter.

6. Clear the TF flag for the next round.

7. Go back to Step 2 to load TH and TL again

To program counter in Mode 2

1. Load the TMOD value register indicating which counter (counter 0 or counter 1) is to be used, and the counter mode (mode 2) is selected.

2. Load the TH registers with the initial count value

3. Start counter; 4. Keep monitoring the timer flag (TF) with the while(TF==0); instruction to see whether it is raised. Get out of the loop when TF goes high

5. Clear the TF flag

6. Go back to Step4, since mode 2 is auto reload

Code:

```
#include<reg51.h>
void main(void)
{
        T0=1; //make T0 as input
        TMOD=0x05; // configure Timer 0 as counter0 in mode1
        TL0=0;
        TH0=0; //clear TH and TL
        while(1)
        {
                do
                {
                        TR0=1;//start the counter
                        P1=TL0;
                        P2=TH0;// send TH and TL values on P2 and P1 resp.
                }while(TF0==0);
                TR0=0; //stop the counter
                TF0=0; //clear TF0
        }
}
```

```
#include<reg51.h>
void main(void)
{
        T1=1; // make T1 as input
        TMOD=0x60; //configure timer1 as counter1 in mode2
        TH1=0;//start counting from zero
        while(1)
        {
                do
                {
                        TR1=1;//start the counter
                        P1=TL1;// send count on P1
                }while(TF1==0);
                TR1=0; //stop the counter1
                TF1=0; //clear TF1
        }
}
```

Result: Observe the count on P1 and P2 (for Mode-1) and on P1 (for Mode-2)

TERMWORK 5: LCD Interfacing

Objective:

1. To demonstrate the interfacing of 16x2 LCD with 8051 Microcontroller.
2. To develop an 8051 'C' code to display the message "GITCSE"

Theory:

LCD is very commonly used electronic display module and having a wide range of applications such as calculators, laptops, mobile phones etc. 16×2 character is very basic module which is commonly used in electronics devices and projects. It can display 2 lines of 16 characters. Each character is displayed using 5×7 or 5×10-pixel matrix. LCD can be interfaced with microcontroller in 4 Bit or 8 Bit mode. These differs in how data is sent to LCD. In 8-bit mode to write a character, 8 bit ASCII data are sent through the data lines D0 – D7 and data strobe is given through EN of the LCD. LCD commands which are also 8 bit are written to LCD in similar way. 4 Bit Mode uses only 4 data lines D4 – D7. In this mode 8-bit character ASCII data and command data are divided into two parts and send sequentially through data lines. The idea of 4-bit communication is used save pins of microcontroller. 4-bit communication is a bit slower than 8-bit communication but this speed difference can be neglected since LCDs are slow speed devices. Thus 4-bit mode data transfer is most commonly used.

Lcd4_Init (): These function will initialize the LCD module which is connected to pins defined by following bit addressable variables.

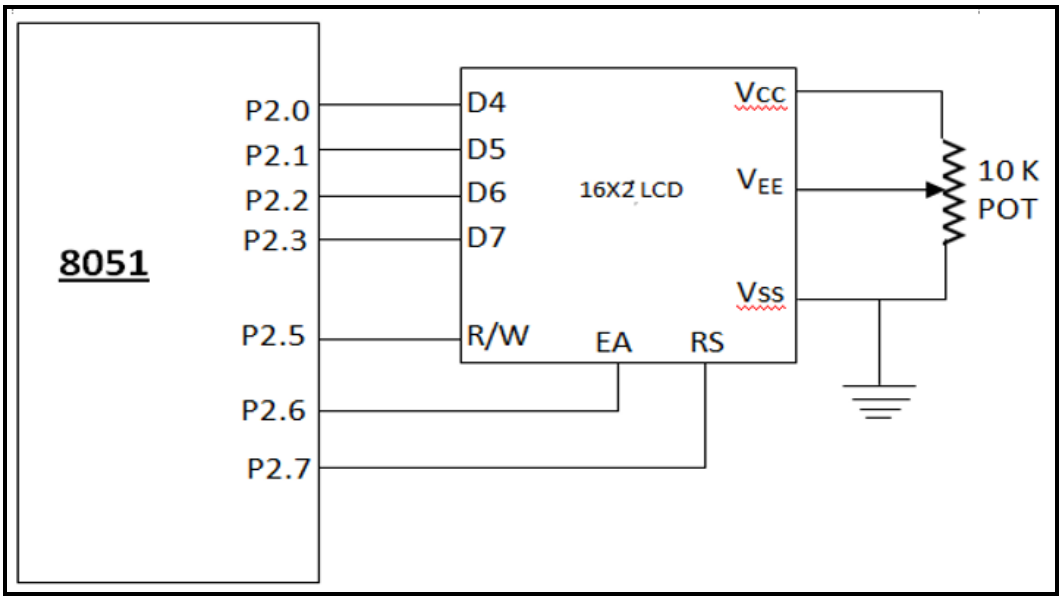
LCD Module Connections:

Register Select (RS)	P2.7
Enable (EN)	P2.6
Read/ Write (R/W)	P2.5
Data D4	P2.0

Data D5	P2.1
Data D6	P2.2
Data D7	P2.3
Data D0 to D3	NC

- RS= 0 for sending Command to the LCD
- RS=1 for sending Data to the LCD
- R/W= 0 for reading from the LCD
- R/W=1 for writing to the LCD
- EN=0 for disabling the LCD
- EN=1 for enabling the LCD

Interfacing Block Diagram:



Algorithm:

STEP 1 : INCLUDE THE HEADER FILE ''at89c51ed2.h''

STEP 2 : INCLUDE THE HEADER FILE <intrins.h> AND ADD LCD ROUTINE FILE IN THE SOURCE GROUP (The INTRINS.H include file contains prototypes for routines that instruct the compiler to generate in-line intrinsic code.)

STEP 3 : DECLARE LCD FUNCTION PROTOTYPES: INITIALIZATION, COMMAND, AND DATA . lcd_init(), lcd_comm() AND lcd_data().

STEP 4 : DECLARE VARIABLES arr, temp1, temp2, i=0.

STEP 5 : BEGIN MAIN

STEP 6 : INITIALIZE AUXR=0x10 TO ACCESS FULL EXTERNAL RAM. lcd_init()

STEP 7: temp1=0X80; WRITE COMMAND; lcd_comm(); TO DISPLAY THE DATA "GITCSE" FROM FIRST LINE.

STEP 8: INITIALIZE FOR LOOP TO DISPLAY 6 CHARACTERS. For(i=0; i<6; i++)

STEP 9: LOAD temp2 WITH CHARACTERS: arr[i]

STEP10: WRITE DATA; lcd_data

STEP11: TO DISPLAY ANOTHER STRING(EXAMPLE: MICROCONTROLLER) IN SECOND LINE REPEAT STEPS 7 TO 10 WITH temp1=0XC0 AND FOR LOOP INITIALIZED WITH 0 TO NUMBER OF CHARACTERS IN SECOND STRING .

STEP12: REPEAT FOREVER; while(1)

Code:

```
#include "at89c51ed2.h"
#include <intrins.h>
// LCD FUNCTION PROTOTYPE
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);
unsigned char xdata arr[16]={“GITCSE"};
unsigned char temp1=0x00;
unsigned char temp2;
unsigned int i=0;
void main(void)
{
                                AUXR = 0x10;                    // Accessing Full XRAM
    lcd_init();
    temp1 = 0x80;                    // To display from the first line
    lcd_comm();                     //  Command writing
    for(i=0;i<6;i++)
                                {
                                    temp2 = arr[i];
```

```
        lcd_data();                    // Data writing
    }
    while(1)
  {
  }
 }
```

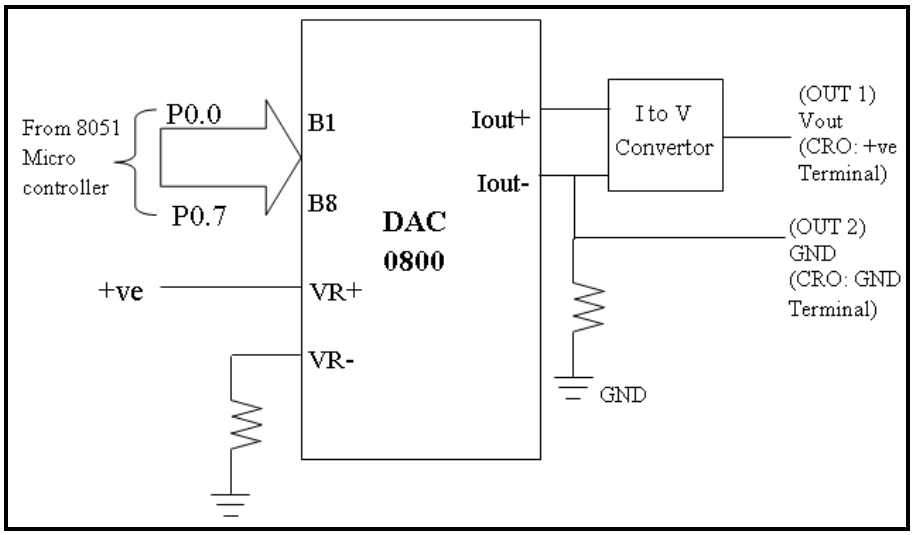
Connection Details: Port 2 to CN6 of Microcontroller Evaluation Board.

Result: Observe the output on LCD.

TERMWORK 6: Digital to Analog Converter (DAC) interfacing to 8051 Microcontroller to generate various waveforms like square, triangular, staircase and sine wave.

Theory: DAC 0800 is an 8-bit high speed current output digital to analog converter. DAC 0800 is used to convert the digital data into analog signal. Digital data from any port is given to DAC input.

Interfacing Diagram:



ALGORITHM:

1) SQUARE WAVE GENERATION

STEP 1: INCLUDE HEADER FILE “at89c51ed2.h”

STEP 2: FUNCTION DECALARATION OF DELAY ROUTINE.

STEP 3: BEGIN MAIN

STEP 4: SEND THE VALUE 0X00 ON PORT 0

STEP 5: CALL DELAY

STEP 6: SEND THE VALUE 0XFF ON PORT 0

STEP 7: CALL DELAY

STEP 8: REPEAT STEPS 4 TO 7 FOREVER

STEP 9: END MAIN

STEP 10: DELAY ROUTINE: DECLARE VARIABLES

STEP 11: INITIALIZE FOR LOOP TO INCLUDE CERTAIN DELAY

STEP 12: END DELAY ROUTINE.

2) STAIRCASE WAVEFORM

STEP 1: INCLUDE HEADER FILE “at89c51ed2.h”

STEP 2: DECLARE VARIABLES

STEP 3: BEGIN MAIN

STEP 4: INITIALIZE FOR LOOP; for (i=0;i<255;i+50) FOR STEP SIZE OF 50 ON Y-AXIS

STEP 5: SEND THE VALUES 0 TO 255 ON PORT 0

STEP 6: INITIALIZE FOR LOOP TO INCLUDE CERTAIN DELAY

STEP 7: SEND i+30 ON PORT 0 TO INCLUDE INCREMENT OF 30 ON X-AXIS.

STEP 8: END MAIN.

3) TRIANGULAR WAVEFORM

STEP 1: INCLUDE HEADER FILE “at89c51ed2.h”

STEP 2: DECLARE THE VARIABLE COUNT

STEP 3: BEGIN MAIN

STEP 4: INITIALIZE FOR LOOP TO COUNT FROM 0 TO 255;

for (count=0; count! =0xff; count++)

STEP 5: SEND COUNT ON PORT 0

STEP 6: INITIALIZE FOR LOOP TO COUNT FROM 255 TO 0;

for (count=0xff; count! =0; count--)

STEP 7: SEND COUNT ON PORT 0

STEP 8: REPEAT STEPS 4 TO 7 FOREVER.

STEP 9: END MAIN

4) SINE WAVEFORM

STEP 1: INCLUDE HEADER FILE "at89c51ed2.h"

STEP 2: DECLARE THE EXTERNAL RAM ARRAY VALUES. ɵ=7.2 DEGREES. 50

VALUES ARE DECLARED IN AN ARRAY.

STEP 3: DECLARE INDEX VARIABLE

STEP 4: BEGIN MAIN

STEP 5: INITIALIZE FOR LOOP for (count=0; count<49; count++)

STEP 6: SEND ARRAY VALUES ON PORT 0

STEP 7: REPEAT STEPS 5 TO 6 FOREVER

STEP 8: END MAIN

CONNECTION DETAILS: PORT 0 TO CN15.

Result: Observe the waveforms on the logic analyzer window.

TERMWORK 7: Serial Data Transmission

Objective: Develop an 8051 'C' program to illustrate serial data transmission.

Theory: Full duplex serial data transmission is supported in 8051. The SFRs associated with serial data transmission are SBUF and SCON register. Both are 8-bit registers.

Sample Calculation for different baud rates:

a. **(i) 9600 baud rate:**

XTAL Freq/ 12= 11.0592MHz/12= 921.6kHz

Further it is divided by 32 by UART

921.6kHz/32= 28,800Hz

28,800/9600= 3

TH1= -3 or 0xFD

(ii) 2400 baud rate:

28,800/2400=12

TH1=-12 or0xF4

SCON Register:

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 (SCON.7)	**Serial port mode specifier**
SM1 (SCON.6)	Serial port mode specifier
SM2 (SCON.5)	Used for multiprocessor communication
REN (SCON.4)	Set/cleared by software to enable/disable reception
TB8 (SCON.3)	Not widely used
RB8 (SCON.2)	Not widely used
TI (SCON.1)	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI (SCON.0)	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
Note: Make SM2, TB8, and RB8 =0	

- SM0, SM1
- They determine the framing of data by specifying the number of bits per character, and the start and stop bits.

SM0	SM1	DESCRIPTION
0	0	Serial Mode 0
0	1	**Serial Mode 1: 8-BIT DATA, 1-START BIT, 1-STOP BIT**
1	0	Serial Mode 2
1	1	Serial Mode 3

- SM2
- This enables the multiprocessing capability of the 8051.

Programming 8051 to transfer characters serially:

- In programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.

2. The TH1 is loaded with appropriate value to set baud rate for serial data transfer.

3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.

4. TR1 is set to 1 to start timer 1.

5. TI is cleared by CLR TI instruction

6. The character byte to be transferred serially is written into

SBUF register.

7. The TI flag bit is monitored with the use of ***while(TI==0);*** to see if the character has been transferred completely.

8. To transfer the next byte, go to step 5.

Programming 8051 to receive characters serially:

- In programming the 8051 to receive character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.

2. TH1 is loaded to set baud rate.

3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.

4. TR1 is set to 1 to start timer 1

5. RI is cleared by CLR RI instruction

6. The RI flag bit is monitored with the use of ***while(RI==0);*** to see if an entire character has been received yet.

7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.

8. To receive the next character, go to step 5.

Code:

```
//To transmit a single character.
#include<reg51.h>
void main(void)
{
        TMOD=0x20; //timer1 in mode2 to set the baud rate
        TH1=0xFA; //baud rate is 4800
        //11.0592MHz/12=921.6kHz
        //921.6kHz/32 (UART)=28,800Hz
        //28,800/4800=6 (converted into 2s comp=0xFA)
        SCON=0x50; // serial mode 1, REN enabled
        TR1=1; //start timer1
        while(1)
        {
                SBUF='A'; // place the byte in SBUF:ASCII =65D
                while(TI==0);//monitor Transmit flag
                TI=0;// clear TI after the completion of transmission
        }
}
```

```
// To transmit the characters GIT
#include<reg51.h>
void SerTx(unsigned char);
void main(void)
{
        TMOD=0x20; //timer1 in mode2 to set baud
        TH1=0xFD; // 9600 baud
        SCON=0x50; // serial mode 1 REN enabled
        TR1=1; // start timer1
        while(1)
        {
                SerTx('G');
                SerTx('I');
                SerTx('T');
                SerTx(' ');
        }
}
void SerTx(unsigned char x)
{
        SBUF=x; // SBUF is loaded with char to be transmitted
        while(TI==0); // monitor TI
        TI=0;
}
```

```
//To receive a byte of data.
#include<reg51.h>
void main(void)
{
        unsigned char mybyte;
        TMOD=0x20; // timer1 in mode 2 to set the baud rate
        TH1=0xFA; // baud rate is 4800
        SCON=0x50; // REN is enabled, serial mode 1 is selected
        TR1=1; // start timer
        while(1)
        {
                while(RI==0); //wait to receive
                mybyte=SBUF; // write SBUF content to mybyte
                P1=mybyte; //place the byte on P1
                RI=0; // clear RI to receive next byte
        }
}
```

Result: Observe the output on the UART window.

TERMWORK 8: Interrupt Programming

Objectives:

1. To illustrate the concept of interrupt mechanism in 8051 Microcontroller.
2. To develop an 8051 ‘C’ program to test Timer/ Serial Interrupts of 8051 Microcontroller.

Theory:

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service. A single microcontroller can serve several devices. There are two ways to do that:

- Polling
- Interrupt

In Polling:

- The microcontroller continuously monitors the status of device.
- When the status condition is met, it performs the service.
- After that it moves to next device until each one is serviced.
- Polling can monitor the status of several devices and serve each of them as certain conditions are met.
- The polling method is not efficient, since it wastes much of the microcontroller’s time by polling devices that do not need service.

In Interrupt method:

- Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal.
- Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
- The program associated with the interrupt is called ***Interrupt Service Routine (ISR).***

The advantage of interrupt is that the microcontroller can serve many devices (not all at the same time, of course). Each device can get the attention of microcontroller based on the priority assigned to it. For the polling method, it is not possible to assign priority since it checks all

devices in a round-robin fashion. The microcontroller can also ignore (mask) a device request for service. This is not possible for the polling method.

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the micro-controller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called Interrupt Vector Table.

- Upon activation of an interrupt, the microcontroller goes through the following steps:
    - It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
    - It also saves the current status of all the interrupts internally (i.e.: not on the stack).
    - It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR.
    - The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it
    - It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
    - Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted
    - *f*First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC
    - Then it starts to execute from that address
- Six interrupts are allocated as follows
    - ***Reset***
- Two interrupts are set aside for the timers:
    - ***Timer 0***
    - ***Timer 1***
- Two interrupts are set aside for hardware external interrupts

- ***External hardware interrupt INT0 (or EX0)***
- ***External hardware interrupt INT1 (or EX1)***

- Serial communication has a single interrupt that belongs to both receive and transfer.
  - ***TI/RI***

Interrupt Vector Table:

Interrupt	ROM Location	Pin
RESET	0000	9
EXTERNAL HARDWARE(INT0)	0003	P3.2(12)
TIMER 0(TF0)	000B	
EXTERNAL HARDWARE(INT1)	0013	P3.3(13)
TIMER 1(TF1)	001B	
SERIAL COM (RI and TI)	0023	

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated.

- The interrupts must be enabled by software in order for the microcontroller to respond to them.
- There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts.

IE (Interrupt Enable) Register:

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA (enable all) must be set to 1 in order to enable interrupts that need to be enabled(Global Enable)

EA	IE.7	Disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

To enable an interrupt, we take the following steps:

- Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect.
- The value of EA
  - If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high
  - If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high

Code:

```
#include <reg51.h>
sbit WAVE =P0^1;

void timer0() interrupt 1 {
  WAVE=~WAVE;   //toggle pin
}

void serial0() interrupt 4 {
  if (TI==1) {
    TI=0;       //clear interrupt
  }
  else {
    P0=SBUF;    //put value on pins
    RI=0;       //clear interrupt
  }
}

    void main() {
      unsigned char x;
      P1=0xFF;      //make P1 an input
      TMOD=0x22;
      TH1=0xF6;     //4800 baud rate
      SCON=0x50;
      TH0=0xA4;     //5 kHz has T=200us
      IE=0x92;      //enable interrupts
      TR1=1;        //start timer 1
      TR0=1;        //start timer 0
      while (1) {
        x=P1;       //read value from pins
        SBUF=x;     //put value in buffer
        P2=x;       //write value to pins
      }
    }
```

Result: Observe the output on the debugger/ logic analyzer window

Arduino UNO R3:

Arduino is an open-source hardware, software and content platform with a global community.

Specifications:

- Micro-controller: ATmega328.
- Operating Voltage: 5V.
- Input Voltage (recommended): 7-12V.
- Input Voltage (limits): 6-20V.
- Digital I/O Pins: 14 (of which 6 provide PWM output).
- Analog Input Pins: 6. • DC Current per I/O Pin: 40 mA.
- DC Current for 3.3V Pin: 50 mA.
- Flash Memory: 32 KB of which 0.5 KB used by boot-loader.
- SRAM: 2 KB (ATmega328).
- EEPROM: 1 KB (ATmega328).
- Clock Speed: 16 MHz

Figure 1. Arduino UNO R3

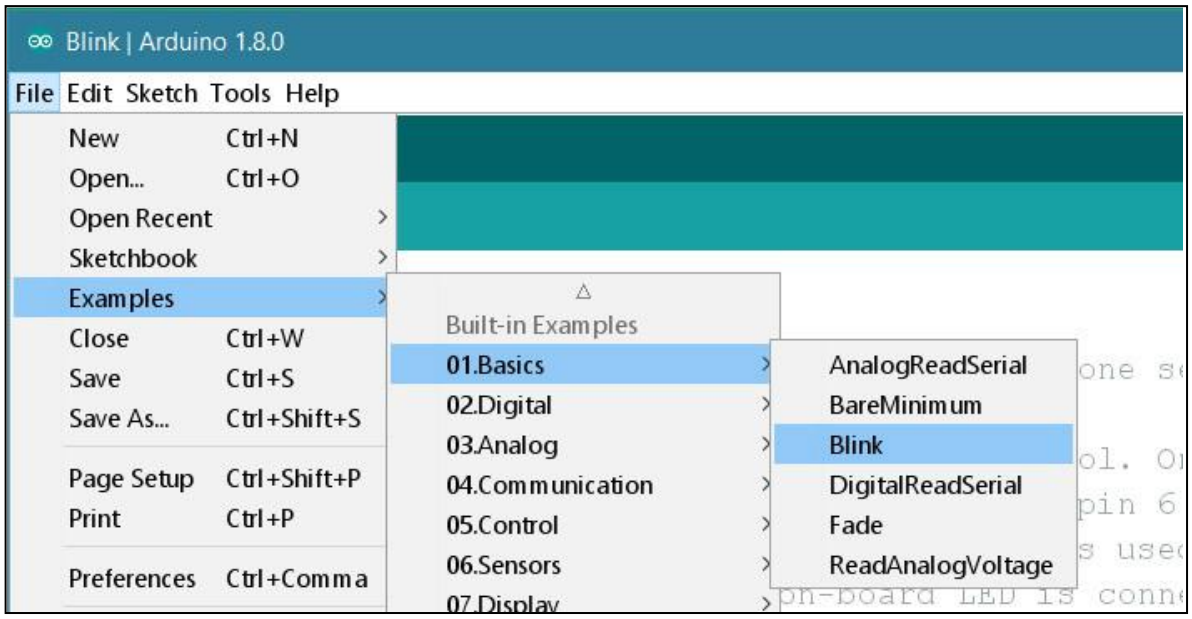
Getting Started with Arduino IDE.

Course Learning Objectives of the Experiment:

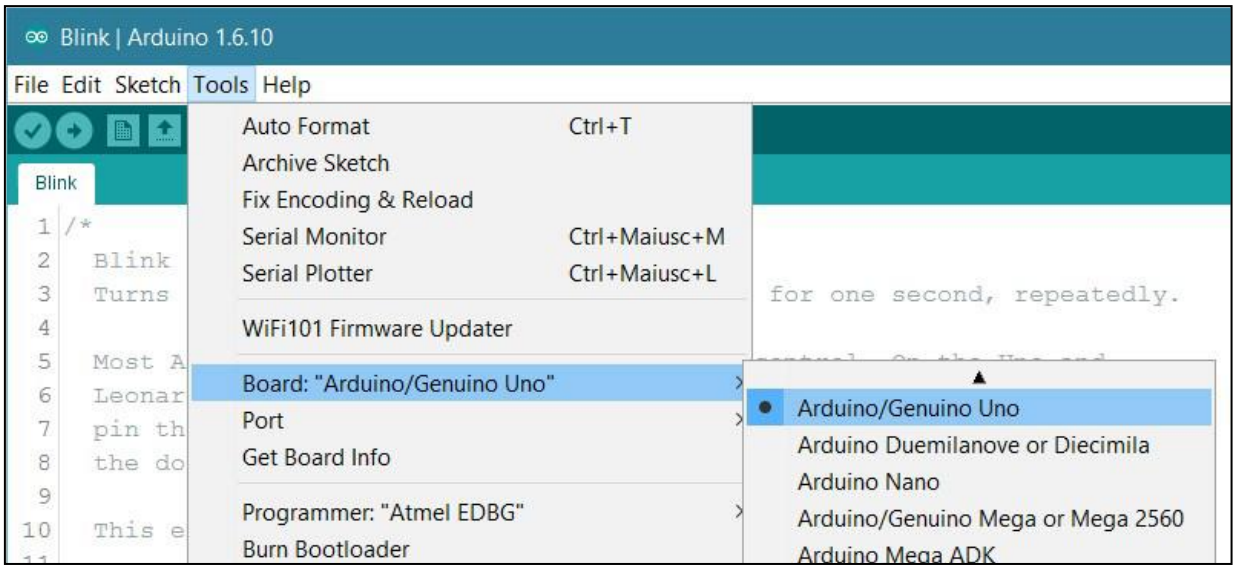
1. To understand the Arduino Environment.
2. To blink the built-in LED of Arduino UNO.

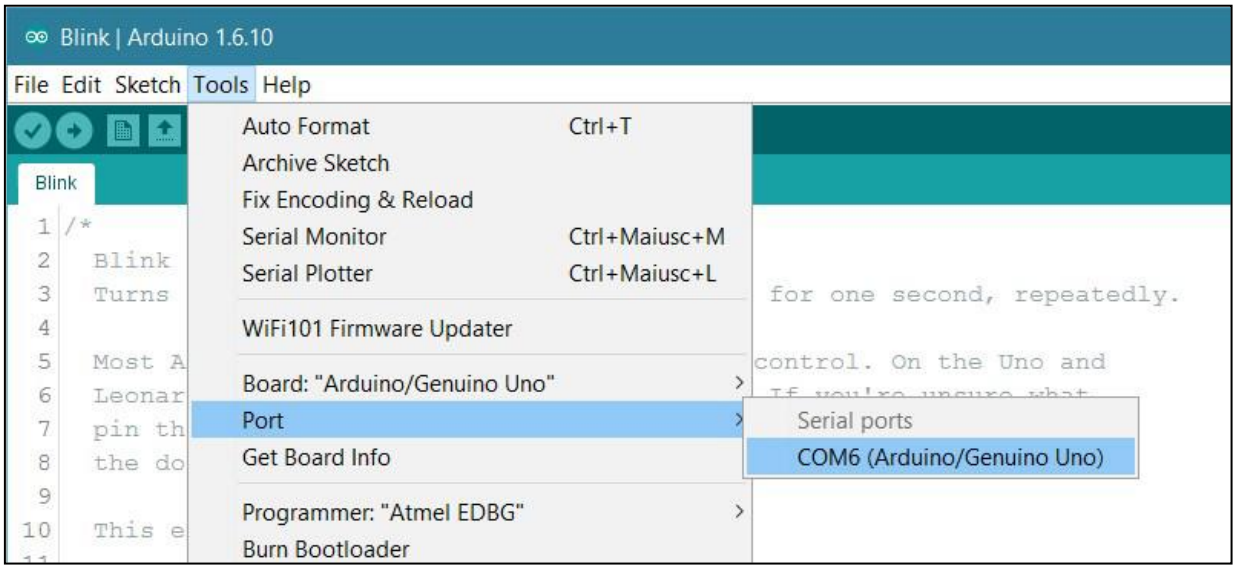
Arduino software IDE on windows:

- For installation details, refer to the manual.
- Getting Started with Sketch
    - Open the LED blink example sketch: File > Examples >01.Basics > Blink.
    - Select the Board Type and Port



    - You'll need to select the entry in the Tools > Board menu that corresponds to your Arduino or Genuino board.
- Select the serial device of the board from the Tools → Serial Port menu.
- This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports).
- To find out, you can disconnect your board and re-open the menu; the entry that disappears should be the Arduino or Genuino board. Reconnect the board and select that serial port.





- Upload the Program
- Now, simply click the "Upload" button in the environment. Wait for few seconds you should see the RX and TX LEDs on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.
- A few seconds after the upload finishes, you should see the pin 13 LED on the board starts blinking (in orange).

TERMWORK 9: Sensor Interfacing

Objectives:

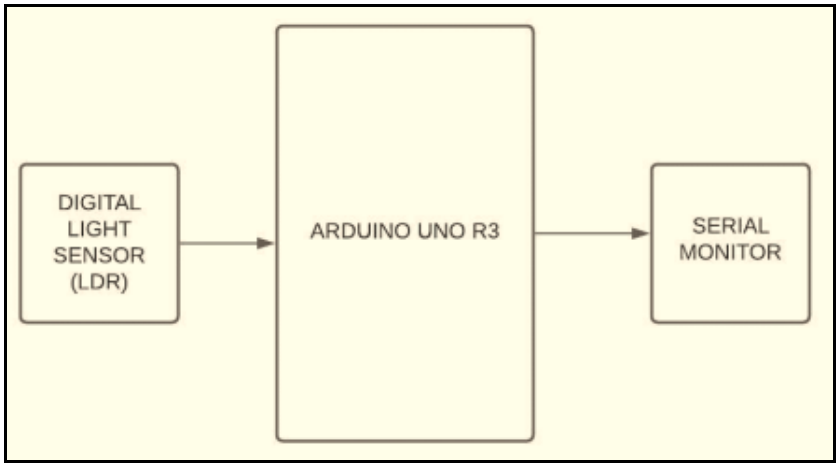
1. To demonstrate the interfacing of LDR to Arduino UNO R3.
2. To develop an Embedded 'C' program to display the data acquired from sensor on serial monitor.

Theory:

Light dependent resistors, LDRs or photo resistors are often used in electronic circuit designs where it is necessary to detect the presence or the level of light. A Light Sensor generates an output signal indicating the intensity of light by measuring the radiant energy that exists in a very narrow range of frequencies basically called "light", and which ranges in frequency from "Infra-red" to "Visible" up to "Ultraviolet" light spectrum. The light sensor converts this "light energy" whether visible or in the infrared parts of the spectrum into an electrical signal output.

Example: LDRs are used in street lights. During day time due to the sunlight falling on LDRs, resistance is high and lights are off. During night, resistance of LDR decreases and lights are turned on.

Interfacing Block diagram:



Algorithm:

- **CONNECT LIGHT SENSOR TO ARDUINO PIN 19 (D5)**
- **CONFIGURE D5 AS INPUT TO READ THE VALUE FROM LIGHT SENSOR.**
- **READ THE SENSOR OUTPUT.**
- **IF SENSOR OUTPUT IS TRUE/LOGIC 1/HIGH**

- **PRINT THE MESSAGE FOR DARKNESS ON SERIAL MONITOR ("LIGHT NOT DETECTED")**

- **ELSE**
  - **PRINT THE MESSAGE FOR BRIGHTNESS ON SERIAL MONITOR ("LIGHT DETECTED")**
- **INCLUDE DELAY.**
- **REPEAT FOREVER.**

Code:

a. LDR for Street Light

```
int light_pin = 5;     //Arduino board pin #19 / D5
void setup() {
  pinMode(light_pin, INPUT);
  Serial.begin(9600);
}
void loop() {
  int light_data = digitalRead(light_pin);
  if(light_data)
    Serial.println("Light Not Detected!");
  else
    Serial.println("Light Detected!");
      delay(1000);
}
```

Connection details: Connect RM3 to RM20

Result: Observe the output on Serial Monitor

TERMWORK 10: Actuator Interfacing

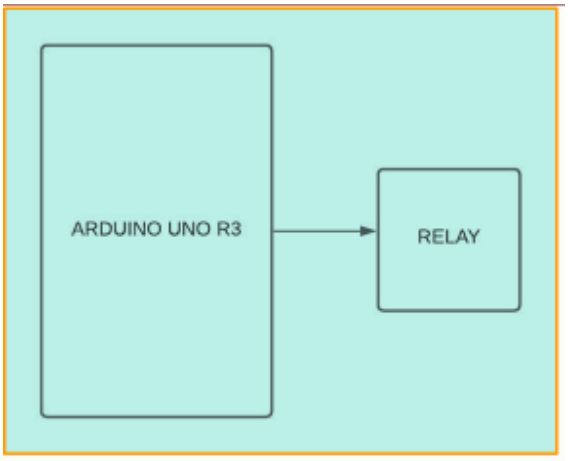
Objectives:

1. To demonstrate the interfacing of Relay to Arduino UNO R3.
2. To develop an Embedded 'C' program to turn ON/OFF the Relay.

Theory:

A relay basically isolate or change the state of an electric circuit from one state to another. Relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relay are used to control a circuit by a separate low-power signal.

Interfacing Block Diagram:



Algorithm:

1. **CONNECT RELAY TO ARDUINO PIN #23/ D9/ BOARD PIN 38.**
2. **MAKE RELAY PIN AS OUTPUT PIN.**
3. **MAKE RELAY PIN HIGH TO TURN ON THE RELAY USING DIGITAL WRITE.**
4. **INCLUDE DELAY.**
5. **MAKE RELAY PIN LOW TO TURN OFF THE RELAY USING DIGITAL WRITE.**

Code:

```
int relay_pin = 8;     // Arduino pin #23/D9  or Board pin P38

void setup()
{
  pinMode(relay_pin, OUTPUT);
  Serial.begin(9600);
  digitalWrite(relay_pin,HIGH);
}

void loop() {
  digitalWrite(relay_pin, LOW);
  delay(1000);
  digitalWrite(relay_pin, HIGH);
  delay(1000);
}
```

Result: Observe the relay turning ON/ OFF.