

ELECTRICAL AND ELECTRONICS ENGINEERING

OPTION:

▪ ELECTRONICS AND TELECOMMUNICATION TECHNOLOGY

REQF level:6

Class: Y1

Module Name: Microprocessor and Microcontroller

Competence: Apply Microprocessor and Microcontroller

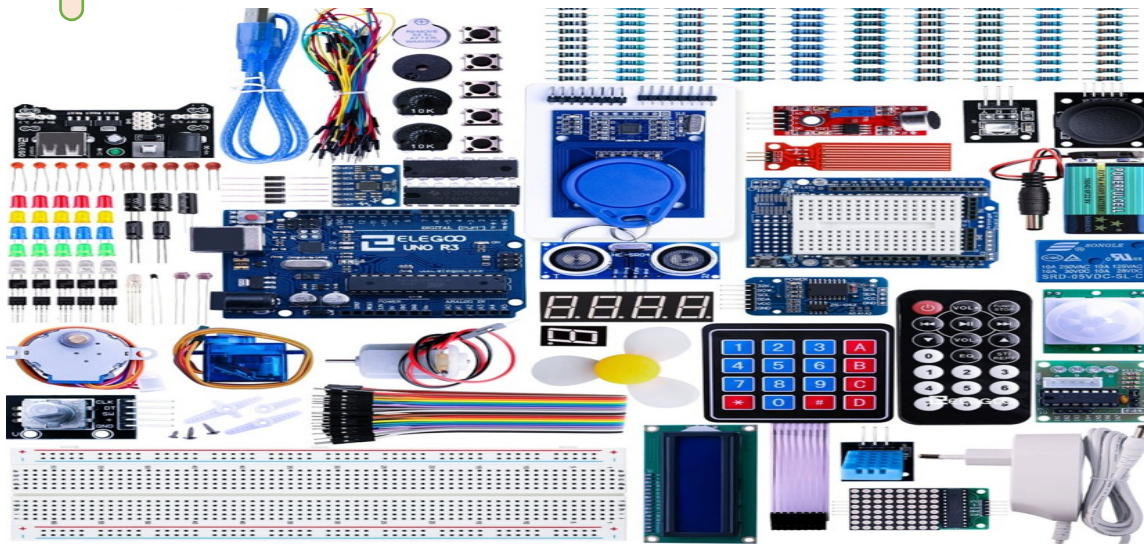
Module code: ETTMM601

Credit: 8

Semester: Two

Academic Year:2025-2026B

LAB MANUAL



Contents

Contents.....	2
PART ONE: MICROPROCESSOR.....	10
1. Introduction.....	10
2. GNUSim8085 Overview.....	10
3. General Procedure for All Experiments.....	10
4. Experiment List.....	11
Experiment 1: Study of 8085 Architecture & Instruction Set.....	11
Experiment 2: Addition of Two 8-bit Numbers.....	11
Experiment 3: Addition of Two 16-bit Numbers.....	12
Experiment 4: Subtraction of Two 8-bit Numbers.....	13
Experiment 5: Multiplication of Two 8-bit Numbers.....	13
Experiment 6: Division of Two 8-bit Numbers.....	14
Experiment 7: Logical Operations.....	15
Experiment 8: Find Largest of Two Numbers.....	16
Experiment 9: Block Data Transfer.....	16
Experiment 10: Sorting of Numbers.....	17
Experiment 11: Delay Program Using Loop.....	18
Experiment 12: I/O Port Programming (LED Simulation Concept).....	19
6. Conclusion.....	19
7. References.....	21
PART TWO: MICROCONTROLLER.....	22
Experiment one: Interfacing LEDs with Arduino and controlling ON/OFF states.....	22
1.1 Objective.....	22
1.2 Required Components.....	22
1.3 Theory.....	22
1.4 Circuit Diagram.....	23
1.5 Procedure.....	23
1.6 Arduino Code.....	24
1.7 Result.....	25
1.8 Applications.....	25
1.9 Review Questions.....	25
1.10 Practical exercises.....	26

Experiment two: Interfacing multiple LEDs and control them in sequence.....	26
2.1 Objective.....	26
2.2 Required Materials.....	26
2.3 Theory.....	26
2.4 Circuit Diagram.....	27
2.5 Procedure.....	27
2.6 Arduino code.....	27
2.7 Result.....	30
2.8 Review Questions.....	30
2.9 Practical exercises.....	30
Experiment Three: Interfacing Pushbutton Control to Manually Toggle an LED	
.....	31
3.1 Objective.....	31
3.2 Required Materials.....	31
3.3 Circuit Diagram.....	32
3.4 Theory.....	33
3.5 Source Code.....	33
3.6 Procedure.....	34
3.7 Precautions.....	34
3.8 Review Questions.....	34
3.9 Practical exercises.....	34
Experiment Four: Use PWM to Control LED Brightness.....	35
4.1 Objective.....	35
4.2 Apparatus/Components Required.....	35
4.3 Theory.....	35
4.4 Circuit Diagram.....	35
4.5 Arduino Code.....	36
4.6 Procedure.....	36
4.7 Observation.....	37
4.8 Result.....	37
4.9 Conclusion.....	37
5.10 Safety Precautions.....	37
Experiment Five: Arduino Serial Communication and Control LEDs.....	38

5.1 Objective.....	38
5.2 Components Required.....	38
5.3 Theory.....	38
5.4 Circuit Diagram Description.....	41
5.5 Pin Connections.....	41
5.5.1 Send Message from Arduino to Serial Monitor.....	41
5.5.2 Receive Data from Serial Monitor and Control LED.....	42
5.6 Procedure.....	45
5.7 Observations.....	45
5.8 Review Questions.....	46
5.9 Conclusion.....	46
5.10 Practical exercises.....	46
Experiment Six: Interfacing Analog and Digital Sensors to Control LEDs Using Arduino.....	47
✓ SENSOR 1: LDR (Light Dependent Resistor).....	47
6.1 Interfacing LDR Sensor to Control LED.....	47
6.1.1 Objective.....	47
6.1.2 Components Required.....	47
6.1.3 Circuit Diagram Description.....	47
6.1.4 Arduino Code.....	48
6.1.5 Procedure.....	49
6.1.6 Observations.....	49
6.1.7 Review Questions.....	49
6.1.8 Practical exercises.....	49
✓ SENSOR 2: LM35 (Temperature Sensor).....	50
6.2 Interfacing LM35 Sensor to Control LED.....	50
6.2.1 Objective.....	50
6.2.2 Components Required.....	50
6.2.3 Circuit Diagram Description.....	50
6.2.4 Arduino Code.....	51
6.2.5 Procedure.....	52
6.2.6 Observations.....	52

6.2.7 Review Questions.....	52
✓ SENSOR 3: Potentiometer (Analog Input).....	52
6.3 Interfacing Potentiometer with Arduino to Control LED.....	52
6.3.1 Objective.....	52
6.3.2 Components Required.....	53
6.3.3 Circuit Diagram Description.....	53
6. 3.4 Arduino Code.....	54
6.3.5 Procedure.....	55
6.3.6 Observations.....	55
6.3.7 Review Questions.....	55
6.3.8 Practical exercises.....	55
✓ SENSOR 4: DHT11 (Temperature & Humidity Sensor).....	55
6.4 Interfacing DHT11 Sensor with Arduino to Control LED.....	55
6.4.1 Objective.....	55
6.4.2 Components Required.....	56
6.4.3 Circuit Diagram Description.....	56
6. 4.4 Arduino Code.....	57
6.4.5 Procedure.....	58
6.4.6 Observations.....	58
6.4.7 Review Questions.....	58
6.4.8 Practical exercises.....	58
✓ SENSOR 5: DHT22 (Digital Temperature & Humidity Sensor – High Accuracy).....	59
6.5 Interfacing DHT22 Sensor with Arduino to Control LED.....	59
6.5.1 Arduino Code.....	59
6.5.2 Observations.....	60
6.5.3 Review Questions.....	60
✓ SENSOR 6: Ultrasonic Sensor (HC-SR04).....	60
6.6 Interfacing Ultrasonic Sensor to Control LED.....	60
6.6.1 Objective.....	60
6.6.2 Components Required.....	61
6.6.3 Circuit Diagram Description.....	61

6.6.4 Arduino Code.....	61
6.6.5 Procedure.....	63
6.6.6 Observations.....	63
6.6.7 Review Questions.....	63
6.6.8 Practical exercise.....	64
✓ SENSOR 7: Infrared (IR) Obstacle Sensor.....	64
6.7 Interfacing IR Obstacle Sensor to Control LED.....	64
6.7.1 Objective.....	64
6.7.2 Components Required.....	64
6.7.3 Circuit Diagram Description.....	65
6.7.4 Arduino Code.....	65
6.7.5 Procedure.....	66
6.7.6 Observations.....	66
6.7.7 Review Questions.....	66
6.7.8 Practical exercises.....	66
✓ SENSOR 8: PIR Motion Sensor.....	67
6.8. Interfacing PIR Motion Sensor to Control LED.....	67
6.8.1 Objective.....	67
6.8.2 Components Required.....	67
6.8.3 Circuit Diagram Description.....	67
6.8.4 Arduino Code.....	68
6.8.5 Procedure.....	69
6.8.6 Observations.....	69
6.8.7 Review Questions.....	69
6.7.8 Practical exercises.....	69
Experiment Seven: Interfacing Arduino Keypad to Control LEDs.....	69
7.1 Objective.....	69
7.2 Components Required.....	69
7.3 Circuit Diagram Description.....	70
7.4 Arduino Code.....	71
7. 5 Procedure.....	72

7.6 Observations.....	72
7.7 Review Questions.....	73
7.8 Conclusion.....	73
7.9 Practical exercise.....	73
Experiment Eight: Interfacing 16x2 LCD with Arduino to display messages and control LEDs.....	74
8.1 Objective.....	74
8.2 Components Required.....	74
8.3 Circuit Diagram Description.....	74
8.4 Arduino Code.....	75
8.5 Procedure.....	76
8. 6 Observations.....	76
8.7 Review Questions.....	76
Experiment Nine : Interfacing Arduino with I2C LCD Display.....	76
9.1 Objective.....	76
9.2 Components Required.....	76
9.3 Circuit Diagram Description.....	77
9.4 Arduino Code.....	78
9.5 Procedure.....	79
9.6 Observations.....	79
9.7 Review Questions.....	79
9.8 Conclusion.....	79
Experiment Ten: Control LED by servo position using Arduino.....	80
10.1 Objective.....	80
10.2 Components Required.....	80
10.3 Pin Connections.....	80
10.4 Arduino Code.....	80
10.5 Practical exercise.....	81
Experiment Eleven: Interfacing Bluetooth Module (HC-05) with Arduino to Control LEDs.....	81
11.1 Objective.....	81
11.2 Components Required.....	81
11.3 Circuit Diagram Description.....	82

11.4 Arduino Code.....	82
11.5 Procedure.....	83
11.6 Observations.....	84
11.7 Review Questions.....	84
11.8 Conclusion.....	84
11.9 Practical exercise.....	84
Experiment Twelve: Interfacing Stepper motor with Arduino.....	85
12.1 Objective.....	85
12.2 Components Required.....	85
12.3 Circuit Diagram Description.....	85
12.4 Arduino Code.....	86
12.5 Procedure.....	89
12.6 Observations.....	89
12.7 Review Questions.....	89
12.8 Conclusion.....	90
Experiment Thirteen: Interfacing Seven Segment Display with Arduino to Indicate LED Control.....	90
13.1 Objective.....	90
13.2 Components Required.....	90
13.3 Circuit Diagram Description.....	90
13.4 Arduino Code.....	91
13.5 Procedure.....	97
13.6 Observations.....	97
13.7 Review Questions.....	97
13.8 Practical exercises.....	98
Experiment Fourteen: Interfacing LED Matrix/Dot matrix with arduino.....	98
14.1 Objective.....	98
14.2 Components Required.....	98
14.3 Circuit Diagram Description.....	98
14.4 Arduino Code.....	99
14.5 Observations.....	100
14.6 Review Questions.....	101
14.7 Conclusion.....	101

14.8 Practical exercises.....	101
Experiment Fifteen: Interfacing RFID with Arduino to Control LEDs.....	102
15.1 Objective.....	102
15.2 Components Required.....	102
15.3 Circuit Diagram Description.....	102
15.4 Arduino Code.....	103
15.5 Observations.....	105
15.6 Review Questions.....	105
15.7 Conclusion.....	105
15.8 Practical exercises.....	105
Experiment Sixteen: Interfacing GSM Module to Control LEDs via SMS.....	106
16.1 Objective.....	106
16.2 Components Required.....	106
16.3 Circuit Diagram Description.....	107
16.3 Arduino Code.....	107
16.4 Procedure.....	108
16.5 Observations.....	108
16.6 Review Questions.....	109
16.7 Conclusion.....	109

PART ONE: MICROPROCESSOR

Intel 8085 Microprocessor Experiments using GNUSim8085 Simulator

1. Introduction

The Intel 8085 is an 8-bit microprocessor widely used for teaching microprocessor fundamentals. This lab manual is designed to help students perform standard 8085 microprocessor experiments **using the GNUSim8085 simulator**, eliminating the need for physical hardware while preserving core learning outcomes.

Objectives

- Understand the architecture and instruction set of the 8085 microprocessors.
- Write, assemble, and execute 8085 assembly language programs
- Analyze register, memory, and flag operations
- Simulate I/O operations using ports (LEDs, switches conceptually)

Software Required

- **GNUSim8085 Simulator** (Windows / Linux)

2. GNUSim8085 Overview

GNUSim8085 is a free, open-source graphical simulator for the Intel 8085 microprocessor.

Key Features

- Integrated editor and assembler
- Step-by-step execution
- Register, flag, and memory view
- Breakpoints and debugging
- I/O port simulation

3. General Procedure for All Experiments

1. Open **GNUSim8085**
2. Create a new source file
3. Write the assembly language program

4. Assemble the program (Assemble → Assemble)
5. Load the program into memory
6. Execute using:
 - ✓ Run
 - ✓ Step Execution
7. Observe:
 - ✓ Registers
 - ✓ Flags
 - ✓ Memory
 - ✓ I/O ports

4. Experiment List

Experiment 1: Study of 8085 Architecture & Instruction Set

Aim: To study the architecture and instruction set of the Intel 8085 microprocessor.

Theory: The 8085 consists of ALU, registers, control unit, and buses. Instructions are categorized as data transfer, arithmetic, logical, branching, and machine control instructions.

Procedure:

- Identify registers and flags in GNUSim8085
- Observe PC, SP, and flag changes during execution

Experiment 2: Addition of Two 8-bit Numbers

Aim: To add two 8-bit numbers stored in registers.

Algorithm:

1. Start the program.
2. Load the first 8-bit number into the accumulator (A).
3. Load the second 8-bit number into another register (B).
4. Add the contents of register B to the accumulator using ADD instruction.

5. Store the result in the accumulator.
6. Check the Carry flag for overflow.
7. Stop the program.

Program:

```
MVI A, 25H  
MVI B, 34H  
ADD B  
OUT 08H  
HLT
```

Observation:

- Accumulator contains sum
- Carry flag indicates overflow

Experiment 3: Addition of Two 16-bit Numbers

Aim: To add two 16-bit numbers using register pairs.

Algorithm:

1. Start the program.
2. Load the first 16-bit number into HL register pair.
3. Load the second 16-bit number into DE register pair.
4. Add DE register pair to HL using DAD instruction.
5. Observe the result in HL register pair.
6. Check Carry flag for overflow.
7. Stop the program.

Program:

```
LXI H, 1234H  
LXI D, 4321H  
DAD D  
HLT
```

Experiment 4: Subtraction of Two 8-bit Numbers

Aim: To subtract one 8-bit number from another.

Algorithm:

1. Start the program.
2. Load the minuend into the accumulator.
3. Load the subtrahend into another register.
4. Subtract the register value from accumulator using SUB instruction.
5. Observe the result in the accumulator.
6. Check Carry flag for borrow.
7. Stop the program.

Program:

```
MVI A, 50H
```

```
MVI B, 25H
```

```
SUB B
```

```
HLT
```

Observation: Borrow reflected in carry flag.

Experiment 5: Multiplication of Two 8-bit Numbers

Aim: To multiply two 8-bit numbers using repeated addition.

Algorithm:

1. Start the program.
2. Load the multiplicand into register B.
3. Load the multiplier into register C.
4. Clear accumulator and register pair HL.
5. Add multiplicand to accumulator.
6. Decrement counter.
7. Repeat until counter becomes zero.
8. Store result in HL register pair.

9. Stop the program.

Program:

```
MVI B, 05H ; Multiplicand
MVI C, 04H ; Multiplier
MVI A, 00H
LXI H, 0000H
LOOP: ADD B
JNC NEXT
INX H
NEXT: DCR C
JNZ LOOP
MOV L, A
HLT
```

Algorithm:

- Load numbers
- Repeatedly add until counter reaches zero

Experiment 6: Division of Two 8-bit Numbers

Aim: To divide two 8-bit numbers using repeated subtraction.

Algorithm:

1. Start the program.
2. Load dividend into accumulator.
3. Load divisor into register B.
4. Clear register C for quotient.
5. Subtract divisor repeatedly.
6. Increment quotient count.
7. Stop when remainder is less than divisor.

Program:

```
MVI A, 14H ; Dividend
MVI B, 04H ; Divisor
MVI C, 00H ; Quotient
LOOP: CMP B
JC END
SUB B
INR C
JMP LOOP
END: HLT
```

Experiment 7: Logical Operations

Aim: To perform AND, OR, XOR logical operations.

Algorithm:

1. Start the program.
2. Load first data into accumulator.
3. Load second data into another register.
4. Perform logical operation (ANA / ORA / XRA).
5. Observe result and flag status.
6. Stop the program.

Program:

```
MVI A, 55H
MVI B, 0FH
ANA B
HLT
```

Experiment 8: Find Largest of Two Numbers

Aim: To find the largest of two 8-bit numbers.

Algorithm:

1. Start the program.
2. Load first number into accumulator.
3. Load second number into register B.
4. Compare numbers using CMP.
5. Store larger number.
6. Stop the program.

Program:

```
MVI A, 25H
MVI B, 30H
CMP B
JC B_LARGE
MOV C, A
HLT
B_LARGE: MOV C, B
HLT
```

Experiment 9: Block Data Transfer

Aim: To transfer a block of data from one memory location to another.

Algorithm:

1. Start the program.
2. Load source address into HL.
3. Load destination address into DE.
4. Load count into register C.
5. Transfer data byte-by-byte.
6. Stop the program.

Program:

```
LXI H, 2050H
LXI D, 3050H
MVI C, 05H
LOOP: MOV A, M
STAX D
INX H
INX D
DCR C
JNZ LOOP
HLT
```

Experiment 10: Sorting of Numbers

Aim: To sort numbers in ascending order.

Algorithm:

1. Start the program.
2. Load count.
3. Compare adjacent numbers.
4. Swap if required.
5. Repeat passes.
6. Stop the program.

Program (Bubble Sort – 3 Numbers):

```
LXI H, 2050H
MVI C, 03H
OUTER: DCR C
JZ END
MOV D, C
LXI H, 2050H
```

```
INNER: MOV A, M
      INX H
      CMP M
      JC SKIP
      MOV B, M
      MOV M, A
      DCX H
      MOV M, B
      INX H
      SKIP: DCR D
      JNZ INNER
      JMP OUTER
      END: HLT
```

Experiment 11: Delay Program Using Loop

Aim: To generate a time delay using loop instructions.

Algorithm:

1. Start the program.
2. Load count into register pair.
3. Decrement continuously.
4. Stop when zero.

Program:

```
LXI B, FFFFH
LOOP: DCX B
      MOV A, B
      ORA C
      JNZ LOOP
      HLT
```

Experiment 12: I/O Port Programming (LED Simulation Concept)

Aim: To simulate LED control using output ports.

Algorithm:

1. Start the program.
2. Load LED pattern into accumulator.
3. Output to port.
4. Stop program.

Program:

```
MVI A, 0AAH; LED pattern
OUT 01H
HLT
```

Observation: Port 01H shows output pattern representing LED ON/OFF.

Example 1: Turn ON all LEDs

```
MVI A, FFH
OUT 01H
HLT
```

//FFH = 11111111 → all LEDs ON.

Example 2: Turn OFF all LEDs

```
MVI A, 00H
OUT 01H
HLT
```

//00H = 00000000 → all LEDs OFF.

Example 3: Turn ON alternate LEDs

```
MVI A, AAH
```

OUT 01H

HLT

//AAH = 10101010

These lights alternate LEDs.

Example 4: Running LED Pattern

MVI A, 01H

START:

OUT 01H

RLC

JMP START

This rotates the bit and creates moving LEDs.

Adding a **delay subroutine** between OUT and RLC make the LEDs to move too fast because the 8085 executes instructions very quickly.

MVI A,01H

LOOP:

OUT 01H

CALL DELAY

RLC

JMP LOOP

DELAY:

MVI B, FFH

D1:

DCR B

JNZ D1

RET

How it works

- OUT 01H → sends data to LEDs
- CALL DELAY → pauses for some time
- RLC → rotates LED bit
- JMP LOOP → repeats forever

Result

The LEDs will now move slowly one by one.

To Make It Slower

Increase the delay using **nested loops**:

```
MVI A,01H
```

```
LOOP:
```

```
    OUT 01H
```

```
    CALL DELAY
```

```
    RLC
```

```
    JMP LOOP
```

```
DELAY:
```

```
    MVI B, FFH
```

```
L1:
```

```
    MVI C, FFH
```

```
L2:
```

```
    DCR C
```

```
    JNZ L2
```

```
    DCR B
```

```
    JNZ L1
```

```
    RET
```

This creates a much better visible LED running effect in Sim8085.

If you want LEDs to turn **ON and OFF repeatedly (blinking)**, use these 8085 programs:

START:

MVI A,FFH ; All LEDs ON

OUT 01H

CALL DELAY

MVI A,00H ; All LEDs OFF

OUT 01H

CALL DELAY

JMP START

DELAY:

MVI B,FFH

L1:

MVI C,FFH

L2:

DCR C

JNZ L2

DCR B

JNZ L1

RET

What happens

- FFH → turns all LEDs ON
- 00H → turns all LEDs OFF
- Delay makes blinking visible

- JMP START repeats forever

Result

LEDs will:

ON → OFF → ON → OFF → ... continuously in the LED Array.

The program is running forever because of: JMP START

To **see the LED ON/OFF blinking slowly**, you should:

1. Output one value → LEDs ON
2. Add a BIG delay
3. Output another value → LEDs OFF
4. Add another BIG delay
5. Repeat

START:

; LEDs ON

MVI A, 0FFH

OUT 01H

CALL DELAY

; LEDs OFF

MVI A, 00H

OUT 01H

CALL DELAY

JMP START

; -----

; LONG DELAY SUBROUTINE

; -----

DELAY:

MVI B, 0FFH

DELAY1:

```
MVI C, 0FFH
```

```
DELAY2:
```

```
DCR C
```

```
JNZ DELAY2
```

```
DCR B
```

```
JNZ DELAY1
```

```
RET
```

What this does:

- 0FFH → all LEDs ON
- 00H → all LEDs OFF
- Large **nested loops** create a **slow visible blink**
- CALL DELAY makes the interval high

To blink LEDS n-times

```
START:
```

```
; -----
```

```
; BLINK LED1 FIVE TIMES
```

```
; -----
```

```
MVI D, 05H
```

```
LED1:
```

```
MVI A, 01H ; LED 1 ON
```

```
OUT 01H
```

```
CALL DELAY
```

```
MVI A, 00H ; LED OFF
```

```
OUT 01H
```

```
CALL DELAY
```

DCR D

JNZ LED1

; -----

; BLINK LED2 FIVE TIMES

; -----

MVI D, 05H

LED2:

MVI A, 02H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED2

; -----

; BLINK LED3 FIVE TIMES

; -----

MVI D, 05H

LED3:

MVI A, 04H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED3

; -----

; BLINK LED4 FIVE TIMES

; -----

MVI D, 05H

LED4:

MVI A, 08H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED4

JMP START

; =====

; LONG DELAY SUBROUTINE

; =====

DELAY:

MVI B, 0FFH

DELAY1:

MVI C, 0FFH

DELAY2:

DCR C

JNZ DELAY2

DCR B

JNZ DELAY1

RET

EXPLANATION OF THE CODE:

START:

- This is a **label**.
- Program execution starts from here.
- START is used later with JMP START.

MVI D, 05H

- MVI = Move Immediate data.
- Load register D with 05H.
- 05H means decimal 5.
- Register D acts as the **counter** for blinking 5 times.

LED1 SECTION

LED1:

- Label for LED1 blinking loop.

MVI A, 01H

- Load accumulator A with 01H.
- Binary value: 00000001
- This turns ON only the **first LED**.

OUT 01H

- Send contents of accumulator A to output port 01H.
- LED array connected to port 01H.
- LED1 glows.

CALL DELAY

- Calls the delay subroutine.
- Program jumps to DELAY.
- Creates visible ON time.

MVI A, 00H

- Load 00H into accumulator.
- Binary:00000000
- All LEDs OFF.

OUT 01H

- Output 00H to port.
- LED1 turns OFF.

CALL DELAY

- Wait again.
- Creates visible OFF time.

DCR D

- DCR = Decrement register.

- Reduce counter D by 1.

Example:

05 → 04 → 03 → 02 → 01 → 00

JNZ LED1

- JNZ = Jump if Not Zero.
- If D ≠ 0, go back to LED1.
- Repeats blinking.

When D = 0, execution moves to next section.

LED2 SECTION

MVI D, 05H

- Reload counter with 5.

LED2:

- Label for LED2 loop.

MVI A, 02H

- Binary:00000010
- Turns ON second LED.

OUT 01H

CALL DELAY

- LED2 ON and wait.

MVI A, 00H

OUT 01H

CALL DELAY

- LED2 OFF and wait.

DCR D

JNZ LED2

- Repeat LED2 blinking 5 times.

LED3 SECTION

MVI D, 05H

- Reset counter.

LED3:

MVI A, 04H

Binary:00000100

- Turns ON third LED.

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED3

- Blink LED3 five times.

LED4 SECTION

MVI D, 05H

- Counter reset.

LED4:

MVI A, 08H

Binary: 00001000

- Turns ON fourth LED.

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED4

- Blink LED4 five times.

REPEAT PROGRAM

JMP START

- Unconditional jump.
- Program repeats forever from beginning.

DELAY SUBROUTINE

DELAY:

- Label for delay routine.

MVI B, 0FFH

- Load register B with maximum value 255.
- Outer delay loop counter.

DELAY1:

- Outer loop label.

MVI C, 0FFH

- Load register C with 255.
- Inner delay loop counter.

DELAY2:

- Inner loop label.

DCR C

- Decrease C by 1.

JNZ DELAY2

- Repeat inner loop until C = 0.

This creates small delay.

DCR B

- After inner loop completes,
- Decrease outer counter B.

JNZ DELAY1

- Repeat whole inner loop again.
- Creates long delay.

RET

- Return from subroutine.
- Control goes back to instruction after CALL DELAY.

Overall Working

Sequence:

LED1 blinks 5 times

↓

LED2 blinks 5 times

↓

LED3 blinks 5 times

↓

LED4 blinks 5 times

↓

Program repeats forever

START:

```
; =====  
; LED1 BLINK 3 TIMES
```

```
; =====  
    MVI D, 03H
```

LED1:

```
    MVI A, 01H  
    OUT 01H  
    CALL DELAY
```

```
    MVI A, 00H  
    OUT 01H  
    CALL DELAY
```

```
    DCR D  
    JNZ LED1
```

```
; =====  
; LED2 BLINK 5 TIMES
```

```
; =====  
    MVI D, 05H
```

LED2:

```
    MVI A, 02H  
    OUT 01H  
    CALL DELAY
```

```
MVI A, 00H
OUT 01H
CALL DELAY
```

```
DCR D
JNZ LED2
```

```
; =====
; LED3 BLINK 6 TIMES
; =====
```

```
MVI D, 06H
```

LED3:

```
MVI A, 04H
OUT 01H
CALL DELAY
```

```
MVI A, 00H
OUT 01H
CALL DELAY
```

```
DCR D
JNZ LED3
```

```
; =====
```

```

; LED4 BLINK 9 TIMES

; =====

    MVI D, 09H

LED4:

    MVI A, 08H
    OUT 01H
    CALL DELAY

    MVI A, 00H
    OUT 01H
    CALL DELAY

    DCR D
    JNZ LED4

; =====
; REPEAT FOREVER
; =====

    JMP START

; =====
; DELAY SUBROUTINE
; =====

DELAY:

    MVI B, 0FFH

```

DELAY1:

MVI C, 0FFH

DELAY2:

DCR C

JNZ DELAY2

DCR B

JNZ DELAY1

RET

START:

; =====

; ALL LEDs BLINK 4 TIMES

; =====

MVI D, 04H

ALL_LED:

; ALL LEDs ON

MVI A, 0FFH

OUT 01H

CALL DELAY

; ALL LEDs OFF

```
MVI A, 00H
OUT 01H
CALL DELAY
```

```
DCR D
JNZ ALL_LED
```

```
; =====
; LED1 BLINK 3 TIMES
; =====
    MVI D, 03H
```

LED1:

```
MVI A, 01H
OUT 01H
CALL DELAY
```

```
MVI A, 00H
OUT 01H
CALL DELAY
```

```
DCR D
JNZ LED1
```

```
; =====
; LED2 BLINK 5 TIMES
```

; =====

MVI D, 05H

LED2:

MVI A, 02H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED2

; =====

; LED3 BLINK 6 TIMES

; =====

MVI D, 06H

LED3:

MVI A, 04H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED3

; =====

; LED4 BLINK 9 TIMES

; =====

MVI D, 09H

LED4:

MVI A, 08H

OUT 01H

CALL DELAY

MVI A, 00H

OUT 01H

CALL DELAY

DCR D

JNZ LED4

; =====

; REPEAT FOREVER

; =====

JMP START

```
; =====
```

```
; DELAY SUBROUTINE
```

```
; =====
```

```
DELAY:
```

```
    MVI B, 0FFH
```

```
DELAY1:
```

```
    MVI C, 0FFH
```

```
DELAY2:
```

```
    DCR C
```

```
    JNZ DELAY2
```

```
    DCR B
```

```
    JNZ DELAY1
```

```
    RET
```

Compare Two Numbers Using Registers and Turn ON LED if the Number is Larger

```
; =====
```

```
; COMPARE TWO NUMBERS USING REGISTERS
```

```
; NUMBERS: 23H and 35H
```

```
; TURN ON LED IF 35H IS LARGER
```

```
; =====
```

```
START:
```

```
MVI B, 23H      ; Load first number into B
MVI A, 35H      ; Load second number into A
```

```
CMP B           ; Compare A with B
```

```
JC FIRST_BIG    ; If A < B jump
```

```
; =====
```

```
; SECOND NUMBER IS LARGER
```

```
; =====
```

```
MVI A, 0FFH     ; ALL LEDs FOR PORT1 WILL BE ON
OUT 01H
```

```
HLT
```

```
; =====
```

```
; FIRST NUMBER IS LARGER
```

```
; =====
```

```
FIRST_BIG:
```

```
MVI A, 01H      ; ONE LED FOR PORT8 WILL BE ON
OUT 08H
```

```
HLT
```

Working

- 23H stored in register B
- 35H stored in accumulator A
- CMP B compares: A - B

Since:

35H > 23H

ALL LEDs turn ON using:

```
MVI A,0FFH
```

```
OUT 01H
```

6. Conclusion

This lab manual enables effective understanding of 8085 microprocessor concepts using GNUSim8085. It replaces hardware-based experiments with a safe, flexible, and repeatable simulation-based approach.

N.B: Addition with carry if any

Add Two 8-bit Numbers (8085 Assembly Language)

Problem Statement

Add two 8-bit numbers stored in memory locations **2050H** and **2051H**. Store the **sum at 2052H** and the **carry (if any) at 2053H**.

Algorithm

1. Load the first 8-bit number into the accumulator.
2. Add the second 8-bit number to the accumulator.
3. Store the sum in memory.
4. Check carry flag.
5. If carry exists, store 01H; otherwise store 00H.
6. Stop execution.

8085 Assembly Language Program

```
LDA 2050H    ; Load first number into A
MOV B, A     ; Save it in register B
LDA 2051H    ; Load second number into A
ADD B        ; A = A + B
STA 2052H    ; Store sum at 2052H

JNC NO_CARRY ; Jump if no carry
```

```
MVI A, 01H    ; Carry = 1
STA 2053H     ; Store carry
JMP END
```

NO_CARRY:

```
MVI A, 00H    ; Carry = 0
STA 2053H
```

END:

```
HLT           ; Stop execution
```

Explanation

- LDA loads data from memory into the accumulator.
- ADD performs 8-bit addition.
- Carry is checked using the **CY flag**.
- JNC jumps if carry flag = 0.
- Carry result is stored separately.

Example Input and Output

Memory Location Value (Hex)

2050H	8AH
2051H	79H

Result:

- Sum at **2052H** → 03H
- Carry at **2053H** → 01H

Simpler Version (Without Carry Storage)

LDA 2050H

ADD M ; If HL points to 2051H

STA 2052H

HLT

7. References

1. Intel 8085 Microprocessor User Manual
2. GNUSim8085 Documentation
3. Ramesh Gaonkar – Microprocessor Architecture, Programming and Applications

PART TWO: MICROCONTROLLER

Experiment one: Interfacing LEDs with Arduino and controlling ON/OFF states

1.1 Objective

To interface one or more LEDs with an Arduino microcontroller and control their ON/OFF states through programming using Arduino IDE.

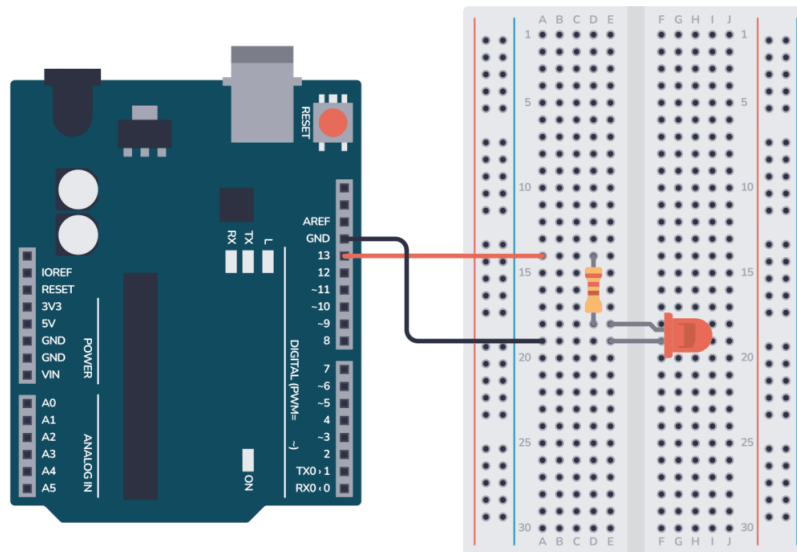
1.2 Required Components

Component	Quantity
Arduino Uno (or compatible board)	1
LED (any color)	1–3
220-ohm resistor	1–3
Breadboard	1
Jumper wires	As required
USB Cable (for Arduino)	1
Computer with Arduino IDE installed	1

1.3 Theory

An LED (Light Emitting Diode) is a semiconductor device that emits light when current passes through it. Arduino can control the state (ON/OFF) of an LED through its digital output pins using programming logic (HIGH for ON, LOW for OFF).

1.4 Circuit Diagram



1.5 Procedure

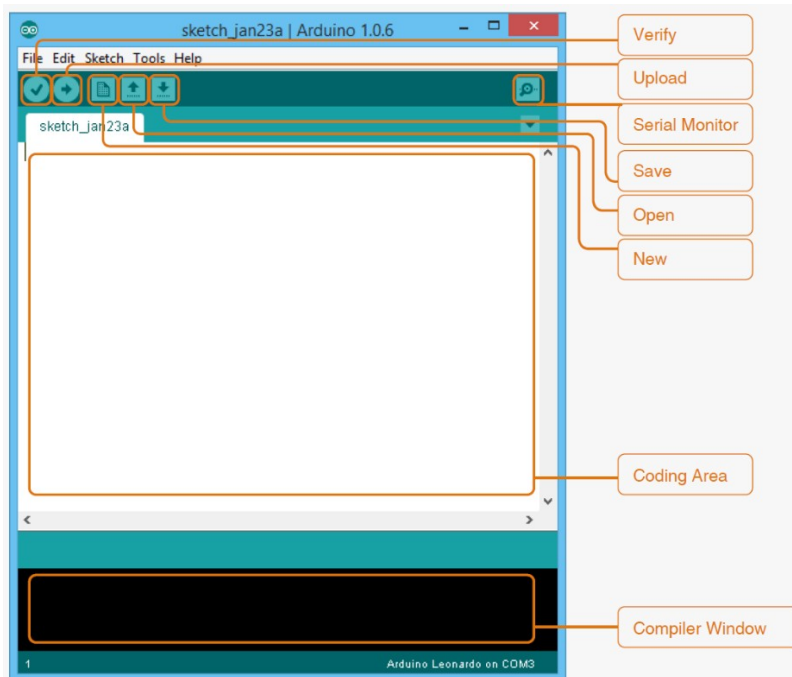
✓ Hardware Setup:

1. Place the LED on the breadboard.
2. Connect the **anode** (long leg) of the LED to **Digital Pin 13** of Arduino through a **220Ω resistor**.
3. Connect the **cathode** (short leg) to the **GND** on Arduino.
4. Connect Arduino to the computer using the USB cable.

✓ Software Setup:

1. Open **Arduino IDE** on your computer.
2. Select the correct board type: **Tools** → **Board** → **Type of arduino board(Arduino Uno)**

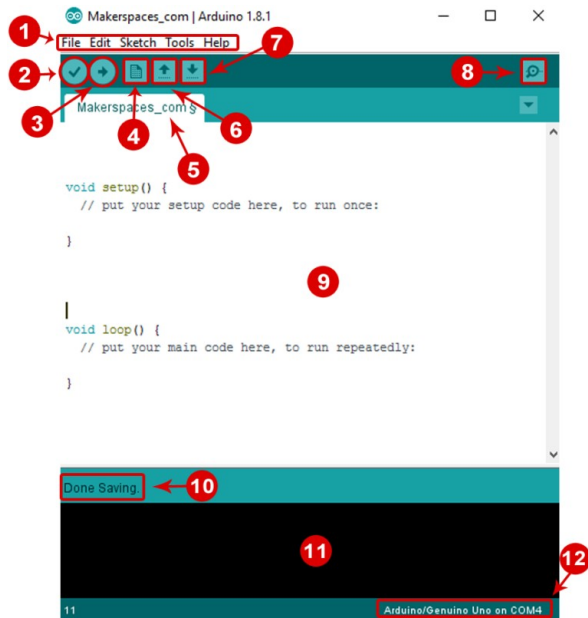
3. Select the correct COM port: **Tools** → **Port** → **COMx**
4. Write arduino code
5. Compile and upload the code to microcontroller IC using the **Compile/verify** and **Upload** buttons.



For more details on above figure, click on the link below

<https://www.makerspaces.com/simple-arduino-projects-beginners/>

Exercise: From figure below, name the numbers 1 to 12 with their role



1.6 Arduino Code

int ledPin = 13; // Set digital pin for LED

void setup() {

pinMode(ledPin, OUTPUT); // Set pin as OUTPUT

}

void loop() {

digitalWrite(ledPin, HIGH); // Turn ON LED

delay(1000); // Wait for 1 second

digitalWrite(ledPin, LOW); // Turn OFF LED

delay(1000); // Wait for 1 second

}

1.7 Result

- LED successfully turns ON and OFF repeatedly.
- Code execution verified the ability to interface and control LEDs with Arduino.

1.8 Applications

- Basic building block for larger embedded systems projects.
- Visual indicators in automation (status, alerts, feedback).
- Educational tool for beginners learning microcontroller I/O.

1.9 Review Questions

1. What is the function of pinMode() in Arduino?
2. Why do we use a resistor with an LED?
3. What is the difference between digitalWrite() and analogWrite()?
4. Can you connect an LED without a resistor? What may happen?
5. How would you control multiple LEDs independently?

1.10 Practical exercises

1. As an electronic technician, you are requested to provide an electronic LED flashing product with one LED in the way that the loop consists of red LED stays turning ON in eight seconds and OFF in 2seconds.
 - a. Draw clearly the design circuit for above situation in proteus software
 - b. Develop correctly the code for the design circuit
 - c. Simulate properly the designed circuit.

Experiment two: Interfacing multiple LEDs and control them in sequence

2.1 Objective

To design and implement a system that interfaces multiple LEDs with a microcontroller and programs them to create a chaser light (running light) effect.

2.2 Required Materials

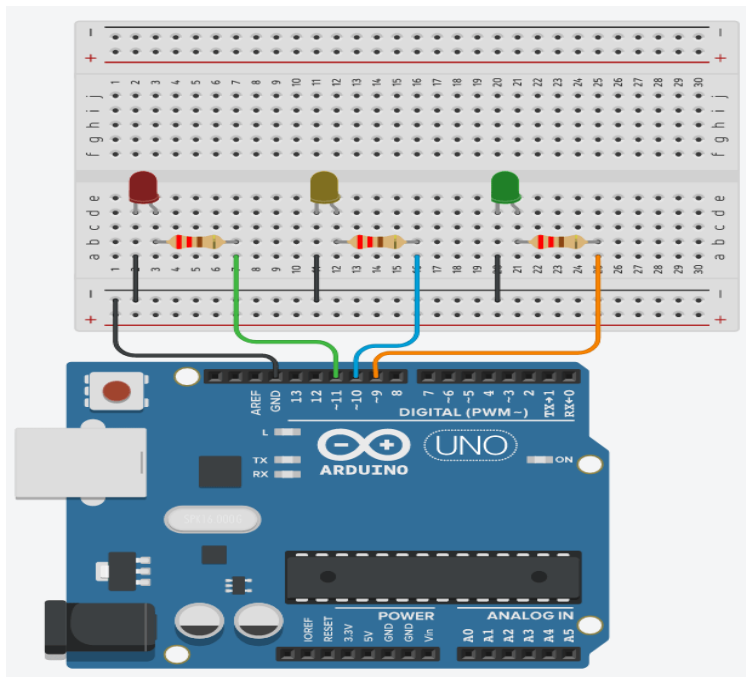
No.	Component	Quantity
1.	Arduino Uno / Atmega328 board	1
2.	LEDs (Red/Green)	8
3.	Resistors (220Ω or 330Ω)	8
4.	Breadboard	1

5.	Jumper Wires	As needed
6.	USB Cable / Power Supply	1
7.	Computer with Arduino IDE	1

2.3 Theory

The **LED chaser effect** is a sequence where multiple LEDs light up one after the other to create a visual "moving" effect. It is commonly used in decorative lighting, indicators, and embedded system learning. This is achieved by programming digital output pins to toggle the LEDs on and off in a timed sequence.

2.4 Circuit Diagram



2.5 Procedure

1. Connect the **anode** of each LED to **Arduino digital pins D9 to D11** through resistors.
2. Connect the **cathode** of each LED to **GND**.
3. Power up the Arduino using a USB cable.
4. Open Arduino IDE, create a new sketch.

5. Write and upload the program (see below).
6. Observe the LEDs blinking one after the other in a chaser pattern.

2.6 Arduino code

Arduino Code 1: Turn ON three LEDs one by one and OFF one by one.

```
// Define LED pins

int led1 = 9;

int led2 = 10;

int led3 = 11;

void setup() {

    // Set all LED pins as OUTPUT

    pinMode(led1, OUTPUT);

    pinMode(led2, OUTPUT);

    pinMode(led3, OUTPUT);}

void loop() {

    // Turn ON all LEDs one by one

    digitalWrite(led1, HIGH);

    delay(500); // wait 0.5 seconds

    digitalWrite(led2, HIGH);

    delay(500);

    digitalWrite(led3, HIGH);

    delay(1000); // wait 1 second
```

```
// Turn OFF all LEDs one by one
```

```
digitalWrite(led1, LOW);
```

```
delay(500);
```

```
digitalWrite(led2, LOW);
```

```
delay(500);
```

```
digitalWrite(led3, LOW);
```

```
delay(1000);} 
```

Arduino code 2: Four LEDs ON and OFF

```
int leds [] = {6, 7, 8, 9};
```

```
void setup() {
```

```
  for (int i = 0; i < 4; i++) {
```

```
    pinMode(leds[i], OUTPUT);  }}
```

```
void loop() {
```

```
  for (int i = 0; i < 4; i++) {
```

```
    digitalWrite(leds[i], HIGH);
```

```
    delay(300);  }
```

```
  for (int i = 0; i < 4; i++) {
```

```
    digitalWrite(leds[i], LOW);
```

```
    delay(300);  }}
```

Arduino Code 3 : eight LEDs with array

```
const int ledCount = 8;
```

```
int leds[ledCount] = {2, 3, 4, 5, 6, 7, 8, 9};
```

```

void setup() {

  for (int i = 0; i < ledCount; i++) {

    pinMode(leds[i], OUTPUT); }

  void loop() {

    for (int i = 0; i < ledCount; i++) {

      digitalWrite(leds[i], HIGH);

      delay(100);

      digitalWrite(leds[i], LOW); }

    for (int i = ledCount - 2; i > 0; i--) { // reverse

      digitalWrite(leds[i], HIGH);

      delay(100);

      digitalWrite(leds[i], LOW); }}

```

2.7 Result

The LEDs light up sequentially from left to right and then in reverse (right to left), successfully demonstrating the chaser effect.

2.8 Review Questions

1. What is the purpose of using resistors with LEDs?
2. How does delay() affect the LED chaser speed?
3. What modifications are required to make the sequence faster?
4. What is the difference between active HIGH and LOW LEDs?

Variations to Try

- Make the LEDs blink **in reverse order**
- Blink all LEDs **simultaneously**

Conclusion

This experiment introduces the core principles of microcontroller-based digital output. Students gain hands-on experience in hardware interfacing and efficient programming techniques like arrays and loops.

2.9 Practical exercises

1. As an electronic technician, you are requested to provide an electronic LEDs flashing product having four LEDs in the way that the loop consists of red LED turning ON/OFF six times for 200ms, blue LED turning ON/OFF four times for 400ms, yellow LED turning ON/OFF three times for 600ms and green LED turning ON/OFF two times for 800ms.

- a. Draw clearly the design circuit for above situation in proteus software
- b. Develop correctly the code for the design circuit
- c. Simulate properly the designed circuit.

2. A single way traffic light is designed in the way red lamps lights 7seconds to stop vehicles to pass, green lamp to allow vehicles to pass during 10 seconds and yellow lamp is using for driving slowly and the time allocation for this lamp is 3 seconds. Note that Off-time is 10th of a second. As electronic technician, you are requested to write code and simulate it in Proteus. Do the same for four-way traffic light.

3. A single way traffic light is designed in the way red lamps lights 7seconds to stop vehicles to pass and blinks 3 times to alert the drivers that the time to stop is going to end, green lamp to allow vehicles to pass during 10 seconds and blinks 3 times to alert also the drivers that the time to pass is going to finish and yellow lamp is using for driving slow and the time allocation for this lamp is 3 seconds. Note that Off-time is 10th of a second and period of blinking is 300ms. As electronic technician, you are requested to write code and simulate it in Proteus

4. Write the code and make the simulation using Proteus software to turn ON and OFF 8 LEDs depend on the following sequences:

- a. All LEDs turn ON (1second) and OFF (1second) five times.
- b. LEDs ON (500ms) and OFF (500ms) from left to right one by one.
- c. LEDs ON (300ms) and OFF (300ms) from right to left one by one.
- d. All LEDs turn ON (1second) and OFF (1second) five times from left one by one

5. As an electronic technician, you are requested to design and simulate the LEDs in the form of heart (use 12 LEDs of different available colors) in the way that All LEDs turn on -off five times, all LEDs turn on-off one by one from left to right, then all LEDs turns on-off one by one from right to left by jumping one LED and lastly each LED blinks three times from right to left.

Experiment Three: Interfacing Pushbutton Control to Manually Toggle an LED

3.1 Objective

To design and implement a simple embedded system circuit where a **pushbutton switch** is used to **manually toggle an LED** ON and OFF using a **microcontroller** (e.g., Arduino or ATmega328).

3.2 Required Materials

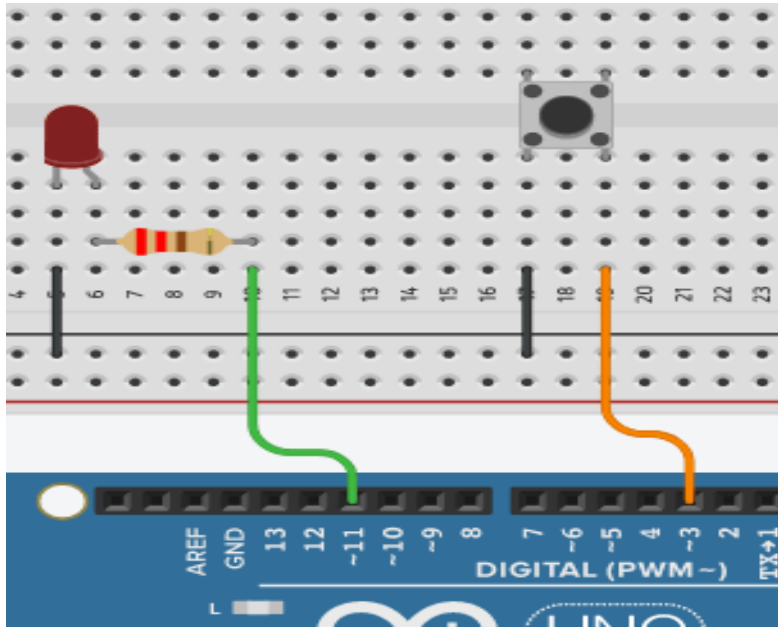
Component	Quantity
Microcontroller (e.g., Arduino Uno or ATmega328)	1
LED	1
Pushbutton	1
Resistors (220 Ω for LED, 10k Ω for pull-down)	2
Breadboard	1
Jumper Wires	As needed
USB Cable (for Arduino)	1
Power Supply (if needed)	1

3.3 Circuit Diagram

Connections:

- **LED:** Connect the anode (+) of the LED to **digital pin 13** (or any GPIO pin) via a **220 Ω resistor**, and the cathode (-) to **GND**.
- **Pushbutton:**
 - Connect one leg of the pushbutton to **digital pin 2**.
 - Connect the same leg to **GND** via a **10k Ω pull-down resistor**.
 - Connect the opposite leg to **5V**.

This sets up the button to read **LOW** when not pressed, and **HIGH** when pressed.



3.4 Theory

A **pushbutton** is a digital input device. When pressed, it completes the circuit and sends a signal to the microcontroller. In this lab, we use it to **toggle the state of an LED**. That means:

- First press → LED ON
- Second press → LED OFF
- Third press → LED ON again, and so on...

To detect a **toggle**, we check for **state change** (edge detection), not just level.

3.5 Source Code

```
const int buttonPin = 3;
```

```
const int ledPin = 11;
```

```
bool ledState = false;
```

```
bool lastButtonState = HIGH;
```

```
void setup() {
```

```
  pinMode(buttonPin, INPUT_PULLUP); // Enable internal pull-up resistor
```

```

pinMode(ledPin, OUTPUT);}

void loop() {

    bool buttonState = digitalRead(buttonPin);

    // Check for button press (active LOW) and detect edge
    if (buttonState == LOW && lastButtonState == HIGH) {

        ledState = !ledState; // Toggle LED state

        digitalWrite(ledPin, ledState);

        delay(200); // debounce delay

    }

    lastButtonState = buttonState;

}

```

3.6 Procedure

1. Assemble the components on the breadboard as per the circuit diagram.
2. Upload the Arduino code to the board using the Arduino IDE.
3. Observe the LED:
 - Press the button once → LED turns ON.
 - Press again → LED turns OFF.
4. Continue testing the toggle behavior.

3.7 Precautions

- Ensure correct resistor values to protect the LED.
- Use a pull-down resistor to prevent floating input states.
- Debounce the button properly to avoid multiple toggles from a single press.

3.8 Review Questions

1. What is the purpose of the pull-down resistor?

2. Why do we use debouncing?
3. What would happen if we didn't use debouncing?
4. How can this logic be extended to multiple buttons?
5. What is the difference between edge detection and level detection?

3.9 Practical exercises

1. Write and simulate the code to turn on and off all two LEDs (blue and yellow) three times when the switch is pressed and to turn On a LED (green) when the switch is released.

2.

Experiment Four: Use PWM to Control LED Brightness

4.1 Objective

To learn how to use Pulse Width Modulation (PWM) to control the brightness of an LED using a microcontroller (e.g., Arduino UNO).

4.2 Apparatus/Components Required

Component	Quantity
Arduino Uno	1
Breadboard	1
LED (any color)	1
Resistor (220Ω)	1
Jumper wires	As required
USB Cable	1
Computer with Arduino IDE	1

4.3 Theory

Pulse Width Modulation (PWM) is a technique where the amount of power delivered to a device is controlled by switching it on and off very quickly. The average voltage (and power) depends on the duty cycle of the signal.

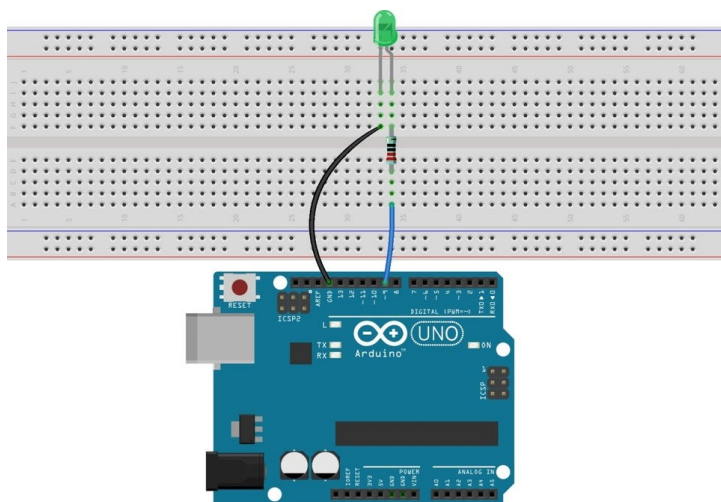
- **Duty Cycle:** The percentage of time the signal is "HIGH" in one cycle.

- In Arduino, PWM is generated using the `analogWrite(pin, value)` function, where value ranges from 0 (off) to 255 (fully on).

4.4 Circuit Diagram

Connection:

- One end of the resistor is connected to the digital pin 9.
- The other end goes to the anode (long leg) of the LED.
- The cathode (short leg) of the LED goes to **GND**.



fritzing

4.5 Arduino Code

```
int ledPin = 9; // PWM-capable pin

void setup() {
    pinMode(ledPin, OUTPUT);}

void loop() {
    // Increase brightness

    for (int brightness = 0; brightness <= 255; brightness++) {
        analogWrite(ledPin, brightness);
    }
}
```

```

    delay(10); }

// Decrease brightness
for (int brightness = 255; brightness >= 0; brightness--) {
    analogWrite (ledPin, brightness);

    delay(10); }}

```

4.6 Procedure

1. Connect the circuit as shown in the diagram.
2. Plug in the Arduino board to your computer via USB.
3. Open the **Arduino IDE**.
4. Write the code above into the sketch.
5. Click **Upload** to flash the code to your Arduino.
6. Observe the LED fading in and out smoothly.

4.7 Observation

Duty Cycle (%)	Brightness Level	Description
0	OFF	LED is off
25	Dim	Low brightness
50	Medium	Medium brightness
75	High	Brighter
100	Max	Fully bright (constant)

4.8 Result

The brightness of the LED varies smoothly depending on the PWM duty cycle. The LED appears brighter at higher duty cycles.

4.9 Conclusion

PWM is a simple yet powerful technique to simulate analog voltage levels using digital output. It's widely used in LED dimming, motor control, and signal modulation.

5.10 Safety Precautions

- Ensure LED polarity is correct.
- Don't exceed 5V or 20mA on any digital pin.
- Use current-limiting resistors to protect the LED.

Experiment Five: Arduino Serial Communication and Control LEDs

5.1 Objective

To understand how to use serial communication between a computer and Arduino to control LEDs using the Serial Monitor.

5.2 Components Required

Component	Quantity
Arduino Uno	1
LEDs (Red, Yellow, Green, Blue)	4
220Ω Resistors	4
Breadboard	1
Jumper wires	As needed
USB cable	1
Laptop with Arduino IDE	1

5.3 Theory

Serial communication is the process of sending data **bit by bit** over a single communication line. In Arduino, the most commonly used method is **UART**

(Universal Asynchronous Receiver/Transmitter), which uses the **TX** (transmit) and **RX** (receive) pins.

Arduino serial communication refers to the process by which an Arduino board exchanges data with other devices, such as a computer, another microcontroller, or a peripheral, using a serial communication protocol. This involves sending and receiving data bit by bit over a single line.

Key aspects of Arduino Serial Communication:

Physical Components:

Serial Port: Arduino boards typically have at least one serial port, often referred to as **Serial**, which utilizes digital pins 0 (RX - Receive) and 1 (TX - Transmit). Some boards, like the Mega 2560 or Due, have additional serial ports (e.g., Serial1, Serial2).

USB Port: On USB-enabled boards (e.g., Uno, Leonardo), the USB port also facilitates serial communication with a computer, appearing as a virtual serial port.

Software Components:

Arduino IDE Serial Monitor: The Arduino Integrated Development Environment (IDE) includes a Serial Monitor tool, which allows users to send data to the Arduino and view data received from the Arduino, facilitating debugging and interaction.

Serial Library Functions: The Arduino programming language provides a built-in Serial library with functions for managing serial communication:

Serial.begin(baud rate): Initializes serial communication at a specified baud rate (data transfer speed).

Baud rate defines how fast data is transmitted (commonly 9600 bps).

Serial.print() / Serial.println(): Sends data as human-readable characters to the serial port.

Serial.print(value) and **Serial.println(value)**: Used to send data (text or numerical values) from the Arduino to the serial monitor or another device. **println** adds a newline character after the data.

Serial.write(): Sends raw binary data (bytes) to the serial port.

Serial.available(): Checks if there is incoming serial data to read.

Serial.read(): Reads incoming serial data.

How it works:

Initialization:

The **Serial.begin()** function sets up the serial communication parameters, including the baud rate, ensuring both devices communicate at the same speed.

Data Transmission:

When **Serial.print()** or **Serial.write()** is used, the Arduino converts the data into a stream of bits and sends them sequentially through the TX pin.

Data Reception:

When data is sent to the Arduino's RX pin, the Arduino receives the bits, reassembles them, and makes them available for reading using **Serial.read()**.

Debugging and Interaction:

The Serial Monitor in the Arduino IDE provides a visual interface to observe the data being sent and received, enabling debugging and real-time interaction with the Arduino.

Applications:

Serial communication is fundamental for various Arduino applications, including:

- Sending sensor readings to a computer for logging or analysis.
- Receiving commands from a computer or another device to control actuators or change program behavior.

- Interacting with other microcontrollers or serial-enabled modules (e.g., GPS modules, Bluetooth modules).

🔌 **Arduino Pins Used:**

- TX (Digital Pin 1): Transmit data
 - RX (Digital Pin 0): Receive data
- However, for computer communication via USB, we use the **built-in Serial** interface via the USB port.

5.4 Circuit Diagram Description

- Connect each LED (with its 220Ω resistor) to a separate digital pin on the Arduino (e.g., D10 to D13).
- The Serial Monitor will be used to type commands to turn ON/OFF each LED.

Note: No external serial module needed. Serial communication is done via USB.

5.5 Pin Connections

LED Pins → Arduino

LED Color	Arduino Pin	Resistor
Red	D10	220Ω
Yellow	D11	220Ω
Green	D12	220Ω
Blue	D13	220Ω

5.5.1 Send Message from Arduino to Serial Monitor

🧰 **Components:**

- Arduino Uno board
- USB cable

- Arduino IDE

 Code:

```
void setup() {  
  
    Serial.begin(9600);           // Initialize serial communication  
  
    Serial.println("Hello, RP-GISHARI COLLEGE") ;// Send message once  
  
}  
  
void loop() {  
  
    Serial.println("Arduino is running...");  
  
    delay(1000); // Delay 1 second  
  
}
```

5.5.2 Receive Data from Serial Monitor and Control LED

 Code 1:

```
String input;  
  
int redLED = 10;  
  
int yellowLED = 11;  
  
int greenLED = 12;  
  
int blueLED = 13;  
  
  
void setup() {  
  
    Serial.begin(9600); // Start serial communication  
  
    pinMode(redLED, OUTPUT);  
  
    pinMode(yellowLED, OUTPUT);  
  
    pinMode(greenLED, OUTPUT);  
  

```

```

pinMode(blueLED, OUTPUT);

Serial.println("Type RED ON / RED OFF / ALL OFF etc.");
}

void loop() {

  if (Serial.available()) {

    input = Serial.readStringUntil('\n');

    input.trim(); // Remove any newline or spaces

    if (input == "RED ON") {

      digitalWrite(redLED, HIGH);  }

    else if (input == "RED OFF") {

      digitalWrite(redLED, LOW);  }

    else if (input == "YELLOW ON") {

      digitalWrite(yellowLED, HIGH);  }

    else if (input == "YELLOW OFF") {

      digitalWrite(yellowLED, LOW);  }

    else if (input == "GREEN ON") {

      digitalWrite(greenLED, HIGH);  }

    else if (input == "GREEN OFF") {

      digitalWrite(greenLED, LOW);  }

    else if (input == "BLUE ON") {

      digitalWrite(blueLED, HIGH);  }

    else if (input == "BLUE OFF") {

      digitalWrite(blueLED, LOW);  }

```

```

else if (input == "ALL OFF") {

    digitalWrite(redLED, LOW);

    digitalWrite(yellowLED, LOW);

    digitalWrite(greenLED, LOW);

    digitalWrite(blueLED, LOW); }

else {

    Serial.println("Invalid command."); } }

}

```

Code 2:

Components:

- Arduino Uno
- 1 x LED
- 1 x 220Ω resistor
- Breadboard and jumper wires

Circuit:

- LED anode → Pin 8
- LED cathode → 220Ω resistor → GND

```

void setup() {

    pinMode(8, OUTPUT);

    Serial.begin(9600);

    Serial.println("Type ON or OFF:");

}

```

```
void loop() {  
  
  if (Serial.available()) {  
  
    String command = Serial.readStringUntil('\n'); // Read input  
    command.trim(); // Remove any extra space or newline  
  
    if (command == "ON") {  
      digitalWrite(8, HIGH);  
      Serial.println("LED is ON");  
    } else if (command == "OFF") {  
      digitalWrite(8, LOW);  
      Serial.println("LED is OFF");  
    } else {  
      Serial.println("Invalid Command");    } }  
}
```

5.6 Procedure

1. Connect LEDs with 220 Ω resistors to Arduino digital pins 10 to 13.
2. Upload the provided code to the Arduino board.
3. Open **Serial Monitor** from Arduino IDE (Baud Rate: 9600).
4. Type commands like RED ON, YELLOW OFF, GREEN ON, BLUE OFF, or ALL OFF.
5. Observe the LEDs turning ON or OFF based on typed command.

5.7 Observations

Serial Command	LED Action
RED ON	Red LED turns ON
YELLOW ON	Yellow LED turns ON
GREEN OFF	Green LED turns OFF
BLUE ON	Blue LED turns ON
ALL OFF	All LEDs turn OFF
Invalid Command	No action; message displayed

5.8 Review Questions

1. What is serial communication in Arduino?
2. Which function is used to read input from Serial Monitor?
3. What is the purpose of Serial.begin()?
4. How is data sent from Arduino to PC?
5. Can serial commands be used to control motors or other actuators?

5.9 Conclusion

This experiment helped students understand the basics of Arduino serial communication and its practical use in controlling external devices like LEDs. This concept is useful in debugging, user input processing, and IoT device control through serial interface.

5.10 Practical exercises

1. Provide and simulate the arduino sketch to switch the LED ON if 'a' is typed in serial monitor and switch the LED off if 'B' is typed in serial monitor. Initialize serial communication at 19200 bits per second.
2. Develop a system by using serial monitor to switch the LED ON if the password ELT is entered and the Serial monitor displays LED ON. LED off will be switched

OFF if the password ETT is entered in serial monitor and Serial monitor displays LED OFF.

Experiment Six: Interfacing Analog and Digital Sensors to Control LEDs Using Arduino

✓ SENSOR 1: LDR (Light Dependent Resistor)

6.1 Interfacing LDR Sensor to Control LED

6.1.1 Objective

To interface an LDR sensor with Arduino and control an LED based on ambient light intensity.

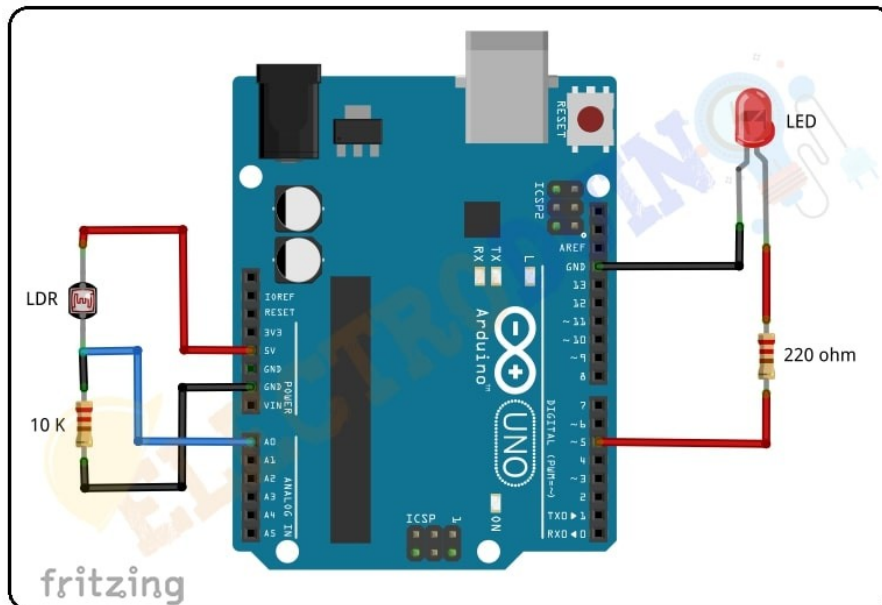
6.1.2 Components Required

Component	Quantity
Arduino Uno	1
LDR Sensor	1
10k Ω Resistor	1
LED (Any color)	1
220 Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop with IDE	1

6.1.3 Circuit Diagram Description

- LDR forms a voltage divider with a 10k Ω resistor

- The middle point is connected to Arduino analog pin A0
- LED connected to digital pin (e.g., D8) with 220Ω resistor



6.1.4 Arduino Code

```
int ldrPin = A0;
```

```
int ledPin = 8;
```

```
int threshold = 500;
```

```
void setup() {
```

```
  pinMode(ledPin, OUTPUT);
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
  int ldrValue = analogRead(ldrPin);
```

```
  Serial.println(ldrValue);
```

```
  if (ldrValue < threshold) {
```

```
    digitalWrite(ledPin, HIGH); // Turn ON LED when dark
```

```

}

else {

    digitalWrite(ledPin, LOW); // Turn OFF LED in bright light

}

delay(500);

}

```

6.1.5 Procedure

1. Connect the LDR and resistor in a voltage divider.
2. Connect middle point to A0, and LED to pin D8.
3. Upload the code and open Serial Monitor.
4. Observe LDR values change with light.
5. LED turns ON when light is low, OFF when light is high.

6.1.6 Observations

LDR Value	LED Status
< 500	ON
≥ 500	OFF

6.1.7 Review Questions

1. What is an LDR and how does it work?
2. Why use a voltage divider?
3. Can LDR detect darkness or brightness?
4. What is the analog input range of Arduino?
5. What factors affect LDR sensitivity?

6.1.8 Practical exercises

1. As we all know, there are places where the lamps are turned ON based on amount of light from daylight to night. Conduct an experiment with arduino and appropriate sensor to turn ON and OFF according to the amount of light detected by this sensor where the light of lamp varies with the amount of light detected. Assume that the minimum value from the sensor to turn ON the lamps is 400.

✓ SENSOR 2: LM35 (Temperature Sensor)

6.2 Interfacing LM35 Sensor to Control LED

6.2.1 Objective

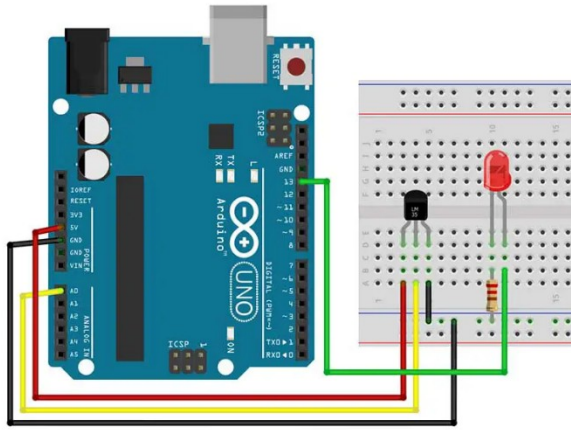
To control an LED based on temperature measured by the LM35 sensor using Arduino.

6.2.2 Components Required

Component	Quantity
Arduino Uno	1
LM35 Sensor	1
LED	1
220Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop with IDE	1

6.2.3 Circuit Diagram Description

- LM35 output connected to A0
- LED connected to digital pin (e.g., D13)
- LM35 is powered from 5V and GND



6.2.4 Arduino Code

```
int lm35Pin = A0;

int ledPin = 13;

float temp;

void setup() {

  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);

}

void loop() {

  int sensorValue = analogRead(lm35Pin);

  temp = (sensorValue * 5.0 * 100.0) / 1024.0;

  Serial.print("Temperature: ");

  Serial.println(temp);

  if (temp > 30.0) {

    digitalWrite(ledPin, HIGH); // Turn on LED if temp > 30°C

  }

}
```

```
else {  
  
    digitalWrite(ledPin, LOW); }  
  
delay(1000);}
```

6.2.5 Procedure

1. Connect LM35 as per circuit.
2. Upload code and monitor temperature on Serial Monitor.
3. LED turns ON above 30°C and OFF otherwise.

6.2.6 Observations

Temperature	LED Status
> 30°C	ON
≤ 30°C	OFF

6.2.7 Review Questions

1. What is the output scale of LM35?
2. Why is analogRead used?
3. Can LM35 be used outdoors?
4. How do you convert analog to temperature?
5. What is the resolution of analog pins?

✓ SENSOR 3: Potentiometer (Analog Input)

6.3 Interfacing Potentiometer with Arduino to Control LED

6.3.1 Objective

To interface a potentiometer with Arduino and control LED brightness using analog input (PWM).

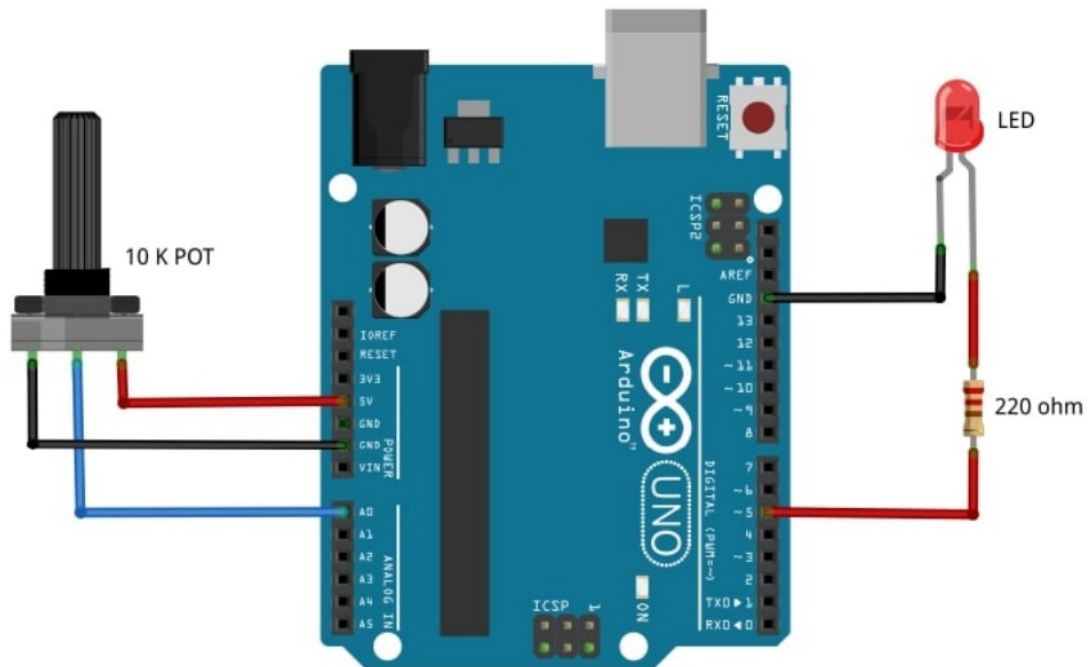
6.3.2 Components Required

Component	Quantity
-----------	----------

Arduino Uno	1
Potentiometer	1
LED	1
220Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop with IDE	1

6.3.3 Circuit Diagram Description

- Potentiometer middle pin → A0
- One outer pin to 5V, other to GND
- LED on PWM pin (e.g., D5) with resistor



6. 3.4 Arduino Code

```
int potPin = A0;
```

```
int ledPin = 5; //pwm pin
```

```
int potValue;
```

```
void setup() {
```

```
  pinMode(ledPin, OUTPUT);
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
  potValue = analogRead(potPin);
```

```
  int brightness = map(potValue, 0, 1023, 0, 255);
```

```
  analogWrite(ledPin, brightness);
```

```
  Serial.println(brightness);
```

```
delay(100);  
  
}
```

6.3.5 Procedure

1. Wire the potentiometer and LED as per diagram.
2. Upload code and open Serial Monitor.
3. Rotate the knob and observe LED brightness changes.

6.3.6 Observations

Potentiometer Value	LED Brightness (PWM)
0 - 1023	0 - 255

6.3.7 Review Questions

1. What is the purpose of a potentiometer?
2. What is PWM and where is it used?
3. Why do we use map() function?
4. Which pins support PWM on Arduino Uno?
5. What is analogRead() range?

6.3.8 Practical exercises

1. Conduct a project with Arduino to control a brightness of LED with potentiometer and using LCD and Serial Monitor to indicate the value of potentiometer that corresponding with LED brightness.

✓ SENSOR 4: DHT11 (Temperature & Humidity Sensor)

6.4 Interfacing DHT11 Sensor with Arduino to Control LED

6.4.1 Objective

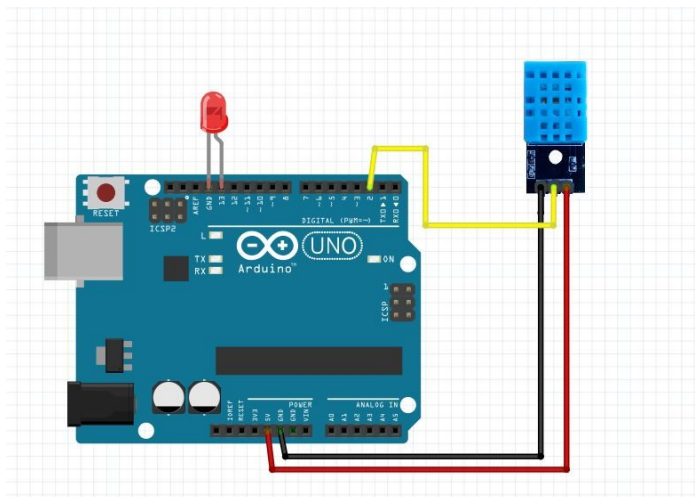
To interface a DHT11 temperature & humidity sensor with Arduino and turn ON an LED if temperature exceeds a certain level.

6.4.2 Components Required

Component	Quantity
Arduino Uno	1
DHT11 Sensor	1
LED	1
220Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop	1

6.4.3 Circuit Diagram Description

- Connect DHT11 output to pin D2
- Power it from 5V and GND
- LED connected to D8 with resistor



6. 4.4 Arduino Code

```
#include <DHT.h>

#define DHTPIN 2

#define DHTTYPE DHT11

DHT dht (DHTPIN, DHTTYPE);

int ledPin = 8;

void setup() {
  Serial.begin(9600);
  dht.begin();
  pinMode(ledPin, OUTPUT);
}

void loop() {
  float temp = dht.readTemperature();
  Serial.println(temp);
  if (temp > 28) {
    digitalWrite(ledPin, HIGH); // LED ON
  }
  else {
    digitalWrite(ledPin, LOW); // LED OFF
  }
  delay(2000);
}
```

}

6.4.5 Procedure

1. Connect DHT11 as per the schematic.
2. Upload the code.
3. Open Serial Monitor to read temperature.
4. Observe LED turns ON if temperature > 28°C.

6.4.6 Observations

Temperature	LED Status
> 28°C	ON
≤ 28°C	OFF

6.4.7 Review Questions

1. What values does DHT11 return?
2. Why do we use a library for DHT11?
3. What is the refresh rate of DHT11?
4. Can DHT11 detect humidity alone?
5. What are common applications?

6.4.8 Practical exercises

1. Conduct a project with Arduino using the temperature sensor, LEDs ,LCD and the Serial monitor. The program should continuously read the temperature value from the temperature sensor and if the temperature value is:
 - a. Less than or equal to 28°C then display the message “Very Cold” on the screens and set the light color for the LED to Blue.
 - b. Greater than 28 and less than or equal to 30 °C then display the message “Cold” on the screens and set the light color for the LED to Green.
 - c. Greater than 30 and less than or equal to 35°C then display the message “Warm” on the screens and set the light color for the LED to Yellow.
 - d. Greater than 35 °C then display the message “Very Hot” on the screen, set the light color for the LED to Red .

Note: The value of temperature should be also displayed in LCD and in Serial Monitor as for example 35°C.

2. As Electronician, you are request to design and implement a system to control a DC motor of Fan by either using temperature or computer, if the word “ON“ is typed in serial monitor or temperature value is greater than 38.2°C, the DC motor of Fan will turn on automatically with a flashing red LED and if the word “OFF” is typed or temperature value is less than or equal 36.7°C, the fan will turn off automatically. Display the value of the temperature in Celsius in serial monitor and LCD.

✓ **SENSOR 5: DHT22 (Digital Temperature & Humidity Sensor – High Accuracy)**

6.5 Interfacing DHT22 Sensor with Arduino to Control LED

6.5.1 Arduino Code

```
#include <DHT.h>

#define DHTPIN 2

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

int ledPin = 8;

void setup() {

  Serial.begin(9600);

  dht.begin();

  pinMode(ledPin, OUTPUT);

}

void loop() {

  float temperature = dht.readTemperature();

  float humidity = dht.readHumidity();
```

```

Serial.print("Temp: "); Serial.print(temperature);

Serial.print(" °C, Hum: "); Serial.println(humidity);

if (temperature > 30 || humidity > 70) {

    digitalWrite(ledPin, HIGH); // LED ON

} else {

    digitalWrite(ledPin, LOW); // LED OFF

}

delay(2000);}

```

6.5.2 Observations

Temp (°C) / Humidity (%)	LED Status
Temp > 30 or Hum > 70	ON
Otherwise	OFF

6.5.3 Review Questions

1. How is DHT22 better than DHT11?
2. What is the response time of DHT22?
3. Can you use both temp and humidity for control?
4. What's the accuracy of DHT22?
5. Why do we use delay in the loop?

✓ SENSOR 6: Ultrasonic Sensor (HC-SR04)

6.6 Interfacing Ultrasonic Sensor to Control LED

6.6.1 Objective

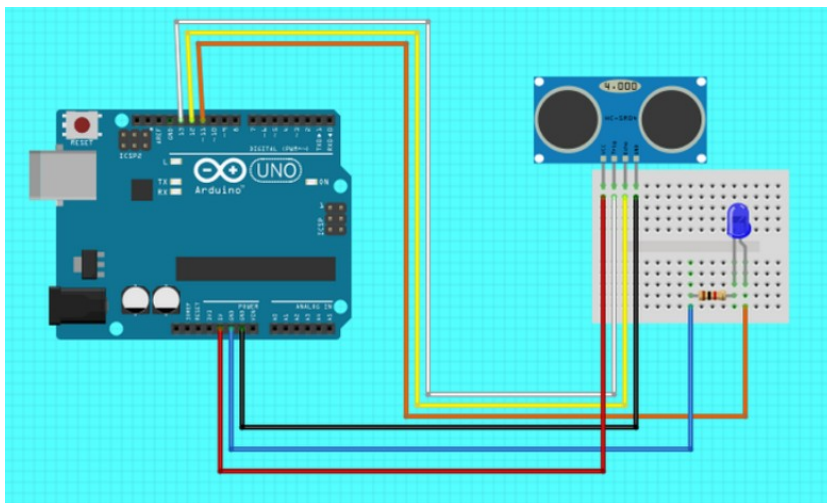
To control an LED based on the distance measured by the HC-SR04 ultrasonic sensor.

6.6.2 Components Required

Component	Quantity
Arduino Uno	1
HC-SR04 Sensor	1
LED	1
220 Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop	1

6.6.3 Circuit Diagram Description

- Trigger pin $\rightarrow$ D9, Echo pin $\rightarrow$ D8
- LED $\rightarrow$ D7 with 220 Ω resistor



6.6.4 Arduino Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Set the LCD address to 0x27 or 0x3F (depending on your module)
LiquidCrystal_I2C lcd(0x27, 16, 2);
const int trigPin = 9;

const int echoPin = 8;

const int ledPin = 7;

void setup() {

  lcd.init();    // Initialize LCD
  lcd.backlight(); // Turn on backlight
  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);

}

void loop() {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);

  int distance = duration * 0.034 / 2;
```

```

Serial.println(distance);

lcd.setCursor(0, 0);
lcd.print("Dist:");
lcd.print(distance);
lcd.print(" cm");
if (distance < 10) {
digitalWrite(ledPin, HIGH); // Object too close

} else {

digitalWrite(ledPin, LOW);

}

delay(500);
}

```

6.6.5 Procedure

1. Connect HC-SR04 to Arduino as per circuit diagram.
2. Upload the code.
3. Monitor distance in Serial Monitor.
4. Move object close (<10cm) to turn LED ON.

6.6.6 Observations

Distance (cm)	LED Status
< 10	ON
≥ 10	OFF

6.6.7 Review Questions

1. How does HC-SR04 work?
2. What is pulseIn()?
3. How is distance calculated from time?

4. What is the range of the sensor?
5. Can it detect transparent objects?

6.6.8 Practical exercise

1. Conduct a project with Arduino to indicate the range of detected object using appropriate sensor and LEDs and using LCD and Serial Monitor to display the value of range. Blue LED indicates that the detected object is above 70cm, green LED indicates that the detected object is between 70cm and 50cm, otherwise Red LED indicates that the detected object is not located on the mentioned range.
2. As an electronic technician, you are requested to implement a project of recoding and displaying with LCD distance value from sensor and use that distance to control a red LED by flashing and Buzzer ON if object is detected at a distance which is less than or equal to 100cm. Display also "OBJECT DETECTED" or "NO OBJECT DETECTED".

✓ SENSOR 7: Infrared (IR) Obstacle Sensor

6.7 Interfacing IR Obstacle Sensor to Control LED

6.7.1 Objective

To control an LED when an object is detected using IR obstacle sensor.

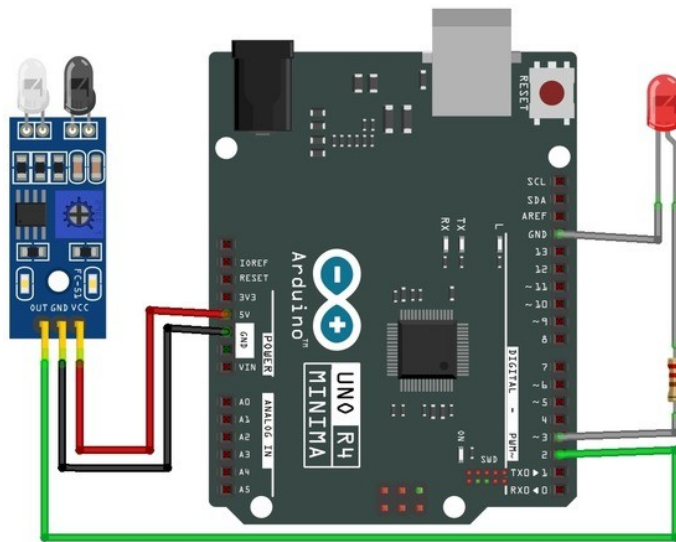
6.7.2 Components Required

Component	Quantity
Arduino Uno	1
IR Sensor	1
LED	1
220Ω Resistor	1
Breadboard	1

Jumper Wires	As needed
USB Cable	1
Laptop	1

6.7.3 Circuit Diagram Description

- IR output → D2
- LED → D3 with 220Ω resistor



6.7.4 Arduino Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

int irPin = 2;
int ledPin = 3;

void setup() {
  pinMode(irPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  lcd.init();    // Initialize LCD
  lcd.Backlight (); // Turn on backlight
}
```

```

void loop() {
  int state = digitalRead(irPin);
  Serial.println(state);
  if (state == LOW) {
    digitalWrite (ledPin, HIGH); // Object detected
    lcd.setCursor(0, 0);

    lcd.print ("Object detected");

  }
  else {
    digitalWrite (ledPin, LOW); // No object
    lcd.setCursor(0, 0);

    lcd.print ("No Object detected");

  }
  delay (100);
}

```

6.7.5 Procedure

1. Connect IR sensor and LED.
2. Upload the code.
3. Bring object close to IR sensor → LED turns ON.

6.7.6 Observations

IR Output	LED Status
LOW	ON
HIGH	OFF

6.7.7 Review Questions

1. What is the principle of IR sensors?
2. How does reflectivity affect detection?
3. Can IR detect in sunlight?
4. Why is output LOW when object is nearby?
5. Difference between active and passive sensors?

6.7.8 Practical exercises

As an electronic technician, you are requested to provide a system to count persons through the stadium gate with a flashing green LED. An LCD prints the number of persons and prints also “PERSON IS ENTERED!!!!” if there is a person and clear LCD if no person entering through the stadium gate.

✓ SENSOR 8: PIR Motion Sensor

6.8. Interfacing PIR Motion Sensor to Control LED

6.8.1 Objective

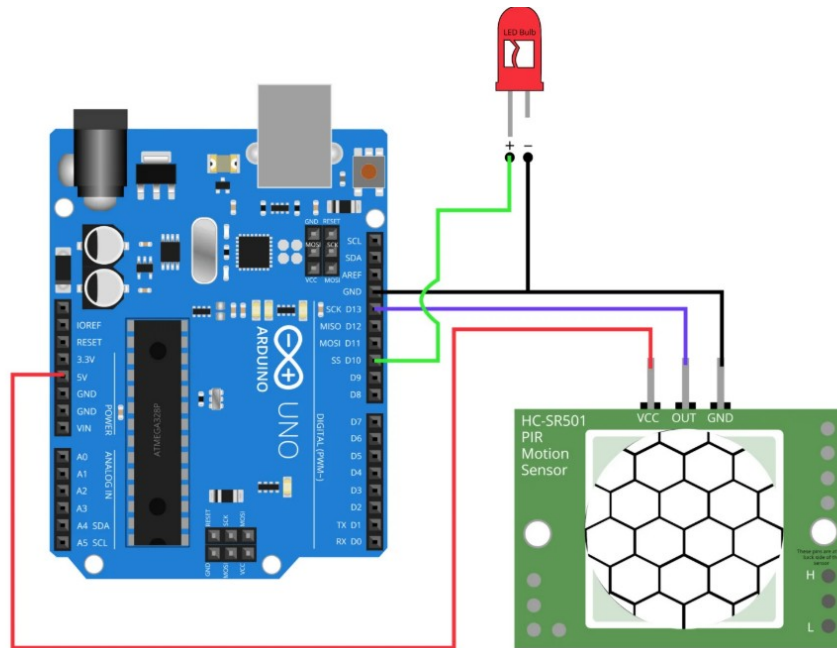
To detect motion using a PIR sensor and control an LED accordingly.

6.8.2 Components Required

Component	Quantity
Arduino Uno	1
PIR Sensor	1
LED	1
220Ω Resistor	1
Breadboard	1
Jumper Wires	As needed
USB Cable	1
Laptop	1

6.8.3 Circuit Diagram Description

- PIR output → D2
- LED → D7 via 220Ω resistor



6.8.4 Arduino Code

```
int pirPin = 2;
int ledPin = 7;
void setup() {
  pinMode(pirPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int motion = digitalRead(pirPin);
  Serial.println(motion);
  if (motion == HIGH) {
    digitalWrite(ledPin, HIGH); // Motion detected
  } else {
    digitalWrite(ledPin, LOW); // No motion
  }
  delay(100);
}
```

6.8.5 Procedure

1. Connect PIR sensor and LED.
2. Upload the code and observe LED behavior.
3. Move in front of sensor → LED turns ON.

6.8.6 Observations

Motion Detected	LED Status
YES (HIGH)	ON
NO (LOW)	OFF

6.8.7 Review Questions

1. How does a PIR sensor work?
2. What does passive mean in PIR?
3. What is the detection range?
4. Why is there delay/reset time?

6.7.8 Practical exercises

As an electronic technician, you are requested to provide a system to count persons through the stadium gate with a flashing green LED. An LCD prints the number of persons and prints also “PERSON IS ENTERED!!!!” if there are persons and clear LCD if no person entering through the stadium gate.

Experiment Seven: Interfacing Arduino Keypad to Control LEDs

7.1 Objective

To design and implement an interface between a **4x4 or 4x3 keypad** and **Arduino Uno** to **control multiple LEDs** by pressing specific keys.

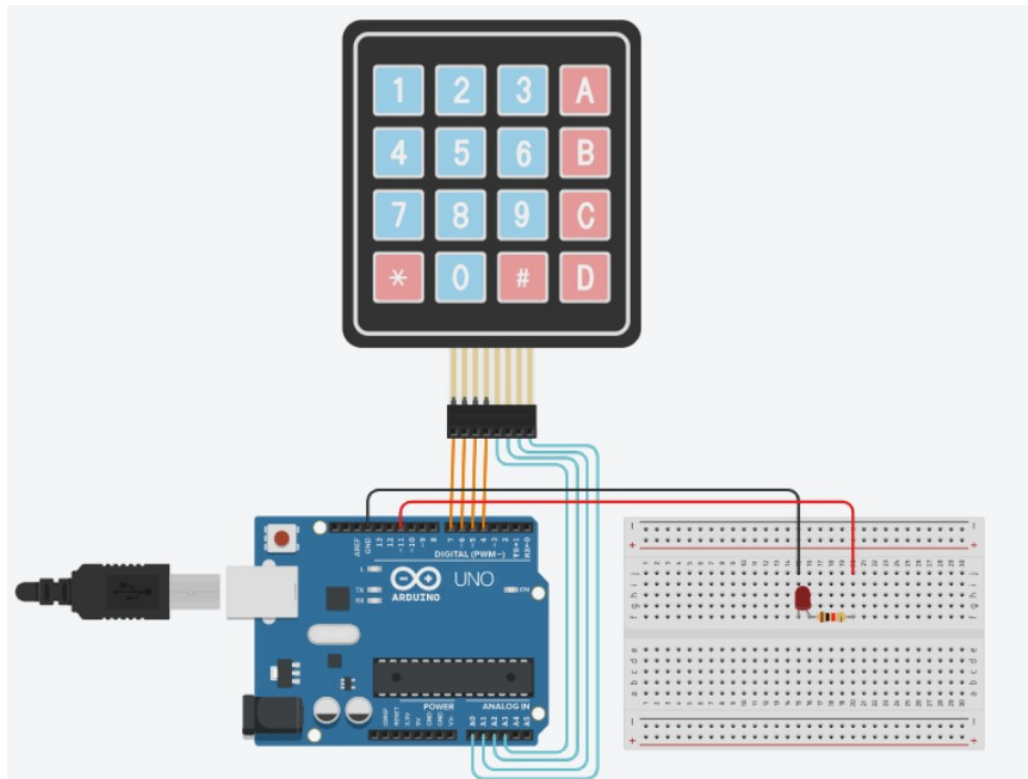
7.2 Components Required

Component	Quantity
Arduino Uno	1
4x4 Keypad Module	1
LEDs (Red/Green/Yellow)	4

220Ω Resistors	4
Breadboard	1
Jumper wires	As needed
USB cable	1
Laptop with Arduino IDE	1

7.3 Circuit Diagram Description

- Connect the **rows and columns of the keypad** to Arduino digital pins.
- Connect **each LED (with resistor)** to a separate digital pin.
- The Arduino will **detect key presses** and **switch LEDs ON/OFF** accordingly.



🔌 Pin Connections

☐ Keypad Pins → Arduino

Keypad Pin	Arduino Digital Pin
Row 1	D9
Row 2	D8
Row 3	D7
Row 4	D6
Col 1	D5
Col 2	D4
Col 3	D3

💡 **LED Pins → Arduino**

LED Color	Arduino Pin	Resistor
Red	D10	220Ω
Yellow	D11	220Ω
Green	D12	220Ω
Blue	D13	220Ω

7.4 Arduino Code

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; // Connect to the row pinouts
byte colPins[COLS] = {5, 4, 3};    // Connect to the column pinouts
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
int led1 = 10;
int led2 = 11;
int led3 = 12;
int led4 = 13;
void setup(){
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}
```

```

}
void loop(){
  char key = keypad.getKey();

  if(key) {
    Serial.println(key);
    switch(key) {
      case '1': digitalWrite(led1, HIGH); break;
      case '2': digitalWrite(led2, HIGH); break;
      case '3': digitalWrite(led3, HIGH); break;
      case '4': digitalWrite(led4, HIGH); break;
      case '5': digitalWrite(led1, LOW); break;
      case '6': digitalWrite(led2, LOW); break;
      case '7': digitalWrite(led3, LOW); break;
      case '8': digitalWrite(led4, LOW); break;
      case '0':
        digitalWrite(led1, LOW);
        digitalWrite(led2, LOW);
        digitalWrite(led3, LOW);
        digitalWrite(led4, LOW);
        break; }}}

```

7. 5 Procedure

1. Connect the **keypad** to the Arduino according to the pin mapping.
2. Connect **each LED and resistor** to digital output pins.
3. Upload the code to the Arduino board.
4. Open **Serial Monitor** to observe keypress values.
5. Press keys 1 to 4 to **turn ON** LEDs.
6. Press keys 5 to 8 to **turn OFF** corresponding LEDs.
7. Press 0 to **turn OFF all LEDs** at once.

7.6 Observations

Key Pressed	LED Action
'1'	Red ON
'2'	Yellow ON

'3'	Green ON
'4'	Blue ON
'5' to '8'	Corresponding OFF
'0'	All LEDs OFF

7.7 Review Questions

1. What is the role of a keypad in embedded systems?
2. How does the Arduino detect which key is pressed?
3. What are rows and columns in keypad interfacing?
4. What is the use of the Keypad library in Arduino?
5. Can you interface a keypad with other output devices besides LEDs?

7.8 Conclusion

This experiment demonstrates how to interface a keypad with Arduino and control external devices like LEDs. This concept is fundamental in designing **embedded user interfaces** such as door locks, access panels, or control systems.

7.9 Practical exercise

Conduct a project with arduino and keypad to turn ON/OFF LEDs by pressing the key as follow:

- Red LED will be ON by pressing key 3 and OFF by the pressing * key
- Blue LED will be ON by pressing key 5 and OFF by the pressing # key
- Yellow LED will be ON by pressing key 2 and OFF by the pressing key 6

Experiment Eight: Interfacing 16x2 LCD with Arduino to display messages and control LEDs

8.1 Objective

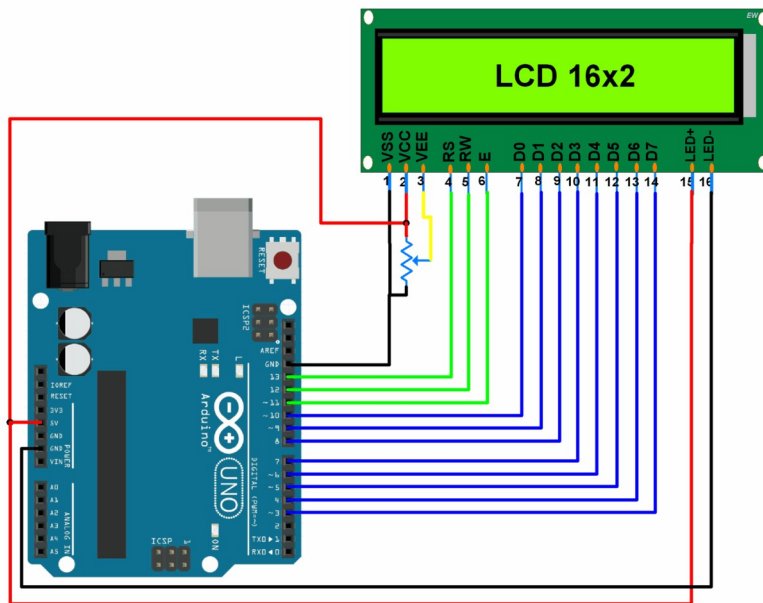
To use an LCD to display control options and switch ON/OFF an LED via input or internal conditions.

8.2 Components Required

Component	Quantity
Arduino Uno	1
16x2 LCD Display	1
Potentiometer (10k)	1
LEDs (Red/Green)	2
220Ω Resistors	2
Breadboard	1
Jumper wires	As needed

8.3 Circuit Diagram Description

- LCD RS → D12, E → D11, Data4 to Data7 → D5-D2
- Connect potentiometer for contrast adjustment
- LED1 → D8, LED2 → D9



8.4 Arduino Code

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

```
int led1 = 8;
```

```
int led2 = 9;
```

```
void setup() {
```

```
  lcd.begin(16, 2);
```

```
  pinMode(led1, OUTPUT);
```

```
  pinMode(led2, OUTPUT);
```

```
  lcd.print("LEDs ON");
```

```
  digitalWrite(led1, HIGH);
```

```
  digitalWrite(led2, HIGH);
```

```
  delay(2000);
```

```
  lcd.clear();
```

```
  lcd.print("LEDs OFF");
```

```
  digitalWrite(led1, LOW);
```

```
  digitalWrite(led2, LOW);
```

```
}  
void loop () {}
```

8.5 Procedure

1. Build circuit and connect LCD.
2. Connect LEDs with resistors.
3. Upload code.
4. Observe LCD messages and LED control.

8.6 Observations

Message	LED Status
"LEDs ON"	Both ON
"LEDs OFF"	Both OFF

8.7 Review Questions

- How many pins does a 16x2 LCD use?
- How is contrast adjusted?
- Can LCDs take input?

Experiment Nine : Interfacing Arduino with I2C LCD Display

9.1 Objective

To interface a 16x2 I2C LCD with Arduino Uno and display text or sensor values using I2C communication.

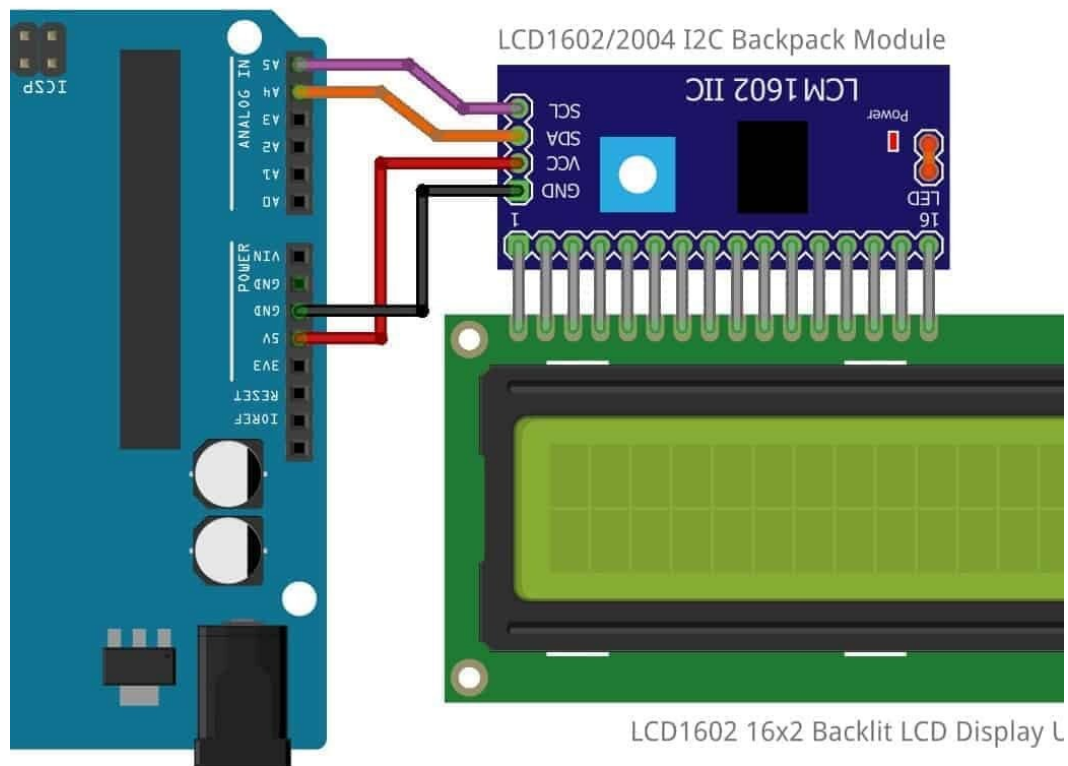
9.2 Components Required

Component	Quantity
Arduino Uno	1
16x2 LCD with I2C module	1
Breadboard (optional)	1

Jumper wires	As needed
USB cable	1
Laptop with Arduino IDE	1

9.3 Circuit Diagram Description

- The I2C LCD uses only 2 pins: **SDA** and **SCL**
- SDA connects to Arduino **A4**, and SCL connects to **A5**
- The LCD is powered using **5V** and **GND**





Library

Required

Install LiquidCrystal_I2C library from the Arduino Library Manager.

9.4 Arduino Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Set the LCD address to 0x27 or 0x3F (depending on your module)
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup() {
  lcd.begin();    // Initialize LCD
  lcd.backlight(); // Turn on backlight
  lcd.setCursor(0, 0);
  lcd.print("Hello TVET!");
  lcd.setCursor(0, 1);
  lcd.print("LCD I2C Ready!");
}
void loop() {
  // Nothing to do here
}
```

✦ **Note:** *If nothing appears on LCD, try changing 0x27 to 0x3F.*

✦ **You can also run the I2C Scanner sketch to find the correct address.**

9.5 Procedure

1. Connect the I2C LCD to Arduino as per the pin mapping.
2. Install the required library (LiquidCrystal_I2C) from Library Manager.
3. Upload the code to the Arduino Uno.
4. Observe the text “Hello TVET!” and “LCD I2C Ready!” displayed on the screen.
5. Adjust contrast if needed using the potentiometer on the I2C module.

9.6 Observations

Operation	LCD Display
Code uploaded	"Hello TVET!" (Row 1)
	"LCD I2C Ready!" (Row 2)

9.7 Review Questions

1. What is the role of I2C protocol in LCD interfacing?
2. What is the difference between parallel and I2C LCDs?
3. What is the I2C address and how do you find it?
4. Why are only two pins used in I2C communication?
5. Can multiple I2C devices be connected to Arduino? How?

9.8 Conclusion

This experiment demonstrated the use of I2C communication to interface a 16x2 LCD with Arduino using only two data pins, simplifying wiring and saving digital I/O resources.

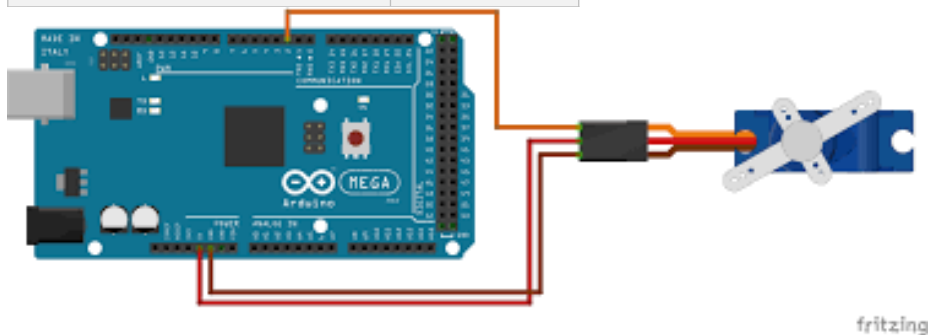
Experiment Ten: Control LED by servo position using Arduino

10.1 Objective

To turn ON an LED when servo reaches specific angles

10.2 Components Required

Component	Quantity
Arduino Uno	1
Servo Motor (SG90)	1
LED	1
Resistor (220Ω)	1
Breadboard, Jumpers	As needed



10.3 Pin Connections

- Servo signal → D9
- LED → D8

10.4 Arduino Code

```
#include <Servo.h>
```

```
Servo myservo;
```

```
int led = 8;
```

```
void setup() {
```

```
  myservo.attach(9);
```

```
  pinMode(led, OUTPUT);
```

```

}

void loop () {
  myservo. write (0);
  digitalWrite (led, LOW);
  delay (1000);
  myservo. write (90);
  digitalWrite (led, HIGH);
  delay (1000);
}

```

10.5 Practical exercise

Conduct a project with Arduino, servo motor and temperature sensor where the servo motor rotate to angle between 20 to 220 degrees with respect to the value from temperature sensor from DHT sensor.

Experiment Eleven: Interfacing Bluetooth Module (HC-05) with Arduino to Control LEDs

11.1 Objective

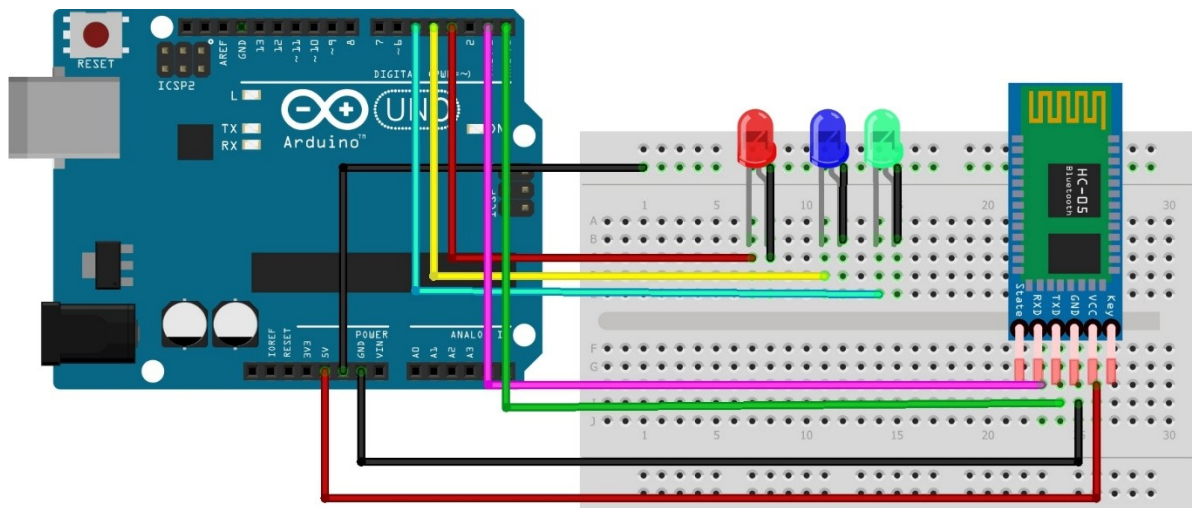
To design and implement wireless control of LEDs using a smartphone and the HC-05 Bluetooth module.

11.2 Components Required

Component	Quantity
Arduino Uno	1
Bluetooth Module HC-05	1
LEDs (Red/Green/Blue)	3
220Ω Resistors	3
Breadboard	1
Jumper wires	As needed

11.3 Circuit Diagram Description

- HC-05 VCC → Arduino 5V
- HC-05 GND → Arduino GND
- HC-05 TX → Arduino RX (Pin 0)
- HC-05 RX → Arduino TX (Pin 1) *(Use voltage divider or level shifter)*
- LEDs connected to Pins D10, D11, and D12 via 220Ω resistors



11.4 Arduino Code

```
char data = 0;

void setup() {

  Serial.begin(9600);

  pinMode(10, OUTPUT);

  pinMode(11, OUTPUT);
```

```

    pinMode(12, OUTPUT);
}

void loop() {
    if(Serial.available()) {
        data = Serial.read();

        if(data == 'R') {
digitalWrite(10, HIGH); }

            if(data == 'r') {
digitalWrite(10, LOW); }

            if(data == 'G') {
digitalWrite(11, HIGH); }

            if(data == 'g') {
digitalWrite(11, LOW); }

            if(data == 'B') {
digitalWrite(12, HIGH); }

            if(data == 'b') {
digitalWrite(12, LOW); }

        }
    }
}

```

11.5 Procedure

1. Connect the HC-05 to Arduino as per the diagram.
2. Connect LEDs to digital pins.
3. Upload the code.
4. Pair HC-05 with your phone (default password: 1234/0000).

5. Use a Bluetooth terminal app to send characters: R, G, B to turn ON and r, g, b to turn OFF.

11.6 Observations

Key Sent	LED Action
R	Red ON
r	Red OFF
G	Green ON
g	Green OFF
B	Blue ON
b	Blue OFF

11.7 Review Questions

1. What is the function of HC-05?
2. Why do we use Serial communication in this experiment?
3. Can Bluetooth be used to read sensor data too?
4. How is HC-05 different from HC-06?
5. How can you secure Bluetooth communication?

11.8 Conclusion

Bluetooth interfacing with Arduino enables basic wireless control and can be extended to home automation and robotics.

11.9 Practical exercise

1. A Bluetooth controlled smart lights using arduino uno/mega is an innovative experiment that combines the power of Bluetooth technology and the flexibility of arduino microcontrollers to create a smart lighting system that can be controlled wirelessly using a smartphone or tablet. With this experiment, you can turn on/off your lights with just a few taps on your mobile device. Conduct an experiment to control five LEDs(green, blue, yellow ,white and red) in the following ways:

- A. Green LED blinks six times with 0.2second ON and 0.2second OFF time.
- B. Blue LED blinks five times with 0.4second ON and 0.1second OFF time.
- C. Yellow LED four nine times with 0.4second ON and 0.3second OFF time.
- D. Red LED blinks three times with 0.3second ON and 0.3second OFF time.
- E. White LED blinks two times with 0.5 second ON time and 0.5 second OFF time.

Experiment Twelve: Interfacing Stepper motor with Arduino

12.1 Objective

To interface a **stepper motor** with an **Arduino Uno** and control its **direction and speed of rotation** using digital output signals.

12.2 Components Required

Component	Quantity
Arduino Uno	1
Stepper Motor (28BYJ-48)	1
ULN2003 Driver Module	1
External Power Supply (5V–12V)	1
Breadboard	1
Jumper Wires	As required

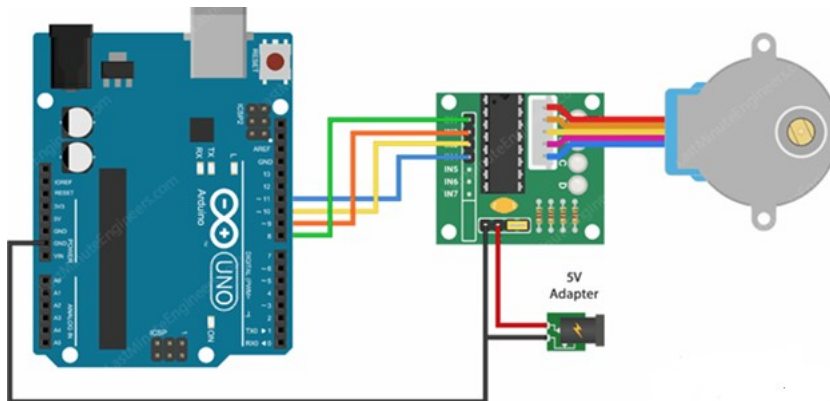
12.3 Circuit Diagram Description

Stepper Motor and Driver Connections

- Stepper motor connected to **ULN2003 driver board**
- ULN2003 **IN1** → Arduino **D8**
- ULN2003 **IN2** → Arduino **D9**

- ULN2003 **IN3** → Arduino **D10**
- ULN2003 **IN4** → Arduino **D11**
- ULN2003 **VCC** → External **5V**
- ULN2003 **GND** → Arduino **GND**

(Ensure common ground between Arduino and external power supply)



12.4 Arduino Code

Code1:

```
#include <Stepper.h>

const int stepsPerRevolution = 2048;

const int rpm = 5;

Stepper stepper1 = Stepper(stepsPerRevolution, 8, 10, 9, 11);

void setup() {
    stepper1.setSpeed(rpm);
}

void loop() {
    stepper1.step(stepsPerRevolution);
    delay(100);
    stepper1.step(-stepsPerRevolution);
    delay(100);
}
```

Code2: Stepper motor control using serial monitor

```
char command;
int pins[] = {8, 9, 10, 11};

int seq[4][4] = {
  {1,0,0,0},
  {0,1,0,0},
  {0,0,1,0},
  {0,0,0,1}
};

void setup() {
  Serial.begin(9600);
  for(int i=0;i<4;i++) pinMode(pins[i], OUTPUT);
  Serial.println("Enter C for Clockwise, A for Anticlockwise");
}

void loop() {
  if (Serial.available()) {
    command = Serial.read();
    if (command == 'C') rotateCW();
    if (command == 'A') rotateCCW();
  }
}

void rotateCW() {
  for(int i=0;i<50;i++)
    for(int j=0;j<4;j++) stepMotor(j);
}

void rotateCCW() {
  for(int i=0;i<50;i++)
    for(int j=3;j>=0;j--) stepMotor(j);
}

void stepMotor(int step) {
  for(int i=0;i<4;i++)
    digitalWrite(pins[i], seq[step][i]);
  delay(10);
}
```

Code3: Stepper motor control using keypad

```

#include <Keypad.h>

const byte rows = 4, cols = 4;
char keys[rows][cols] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

byte rowPins[rows] = {2,3,4,5};
byte colPins[cols] = {6,7,12,13};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins,
rows, cols);

int motorPins[] = {8,9,10,11};
int seq[4][4] = {
    {1,0,0,0},
    {0,1,0,0},
    {0,0,1,0},
    {0,0,0,1}
};

void setup() {
    for(int i=0;i<4;i++) pinMode(motorPins[i], OUTPUT);
}

void loop() {
    char key = keypad.getKey();
    if(key == '1') rotateCW();
    if(key == '2') rotateCCW();
}

void rotateCW() {
    for(int i=0;i<50;i++)
        for(int j=0;j<4;j++) step(j);
}

void rotateCCW() {
    for(int i=0;i<50;i++)
        for(int j=3;j>=0;j--) step(j);
}

void step(int s) {
    for(int i=0;i<4;i++)
        digitalWrite(motorPins[i], seq[s][i]);
}

```

```
    delay(10);  
}
```

12.5 Procedure

1. Connect the stepper motor to the ULN2003 driver.
2. Interface the driver module with Arduino as per the circuit description.
3. Provide external power to the motor.
4. Connect Arduino to the computer.
5. Upload the program to the Arduino.
6. Observe the stepper motor rotation.
7. Note direction change and speed variations.

12.6 Observations

Action	Observation
Clockwise stepping	Motor rotates smoothly
Anticlockwise stepping	Direction reverses
Increase delay	Motor speed decreases
Decrease delay	Motor speed increases

12.7 Review Questions

1. What is a stepper motor?
2. Why is a driver circuit required for a stepper motor?
3. What is the function of ULN2003?
4. Differentiate between stepper motor and DC motor.
5. How is motor speed controlled in this experiment?
6. How can you secure Bluetooth communication?

12.8 Conclusion

In this experiment, a stepper motor was successfully interfaced with an Arduino Uno using a ULN2003 driver. Precise control of motor direction and speed was achieved through programmed stepping sequences, demonstrating the suitability of stepper motors for applications requiring accurate positioning such as robotics and CNC systems.

Experiment Thirteen: Interfacing Seven Segment Display with Arduino to Indicate LED Control

13.1 Objective

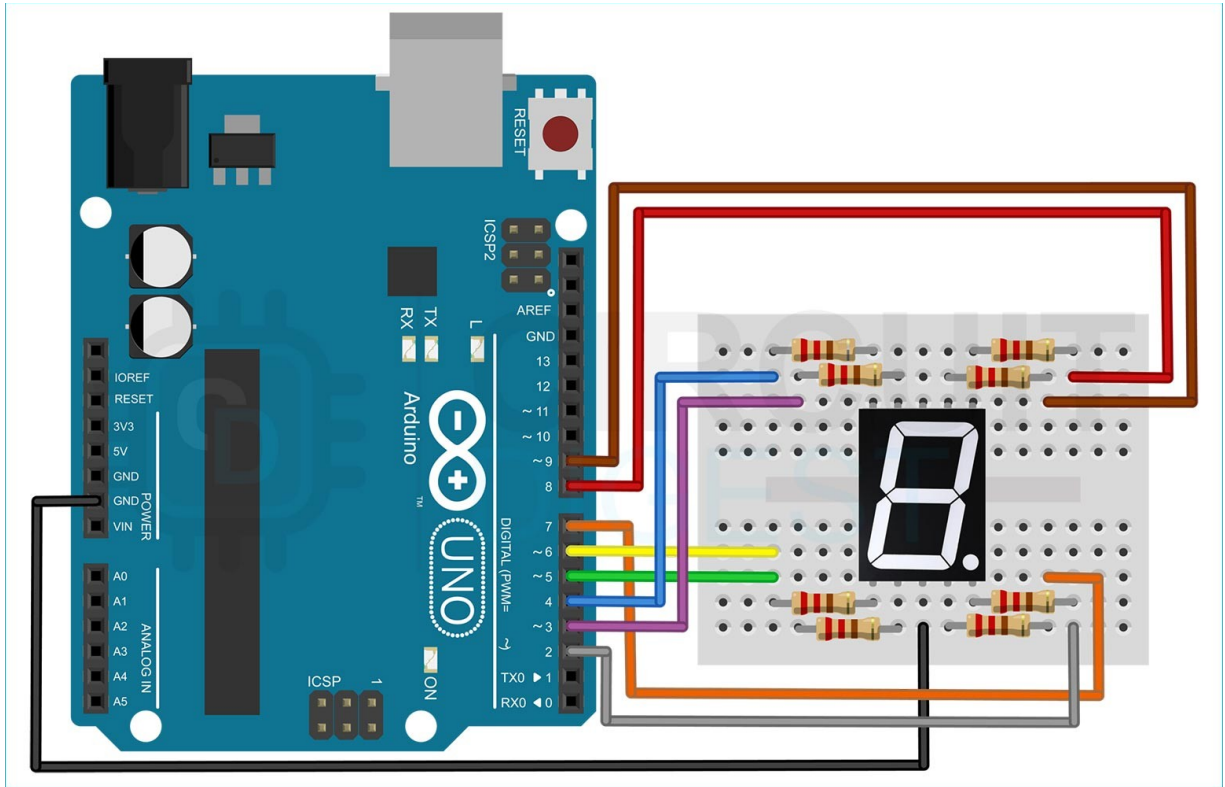
To control LEDs via Arduino and display their status using a seven-segment display.

13.2 Components Required

Component	Quantity
Arduino Uno	1
Common Cathode 7-Segment Display	1
LEDs (Red/Green)	2
220Ω Resistors	5
Breadboard	1
Jumper wires	As needed

13.3 Circuit Diagram Description

- Connect segments (a-g) of 7-segment to D2–D8
- Connect common cathode to GND
- Connect LEDs to D10 and D11
- Use code to show “1” for Red ON, “2” for Green ON, “0” for both OFF



<https://circuitdigest.com/microcontroller-projects/interfacing-seven-segment-display-with-Arduino>

13.4 Arduino Code

Code 1:

```
int seg[] = {2, 3, 4, 5, 6, 7, 8};
byte digits [3][7] = {
    {1,1,1,1,1,1,0}, // 0
    {0,1,1,0,0,0,0}, // 1
    {1,1,0,1,1,0,1} // 2
};
void displayDigit(byte num) {
    for(int i=0;i<7;i++)
        digitalWrite(seg[i],
            digits[num][i]);
}
void setup() {
```

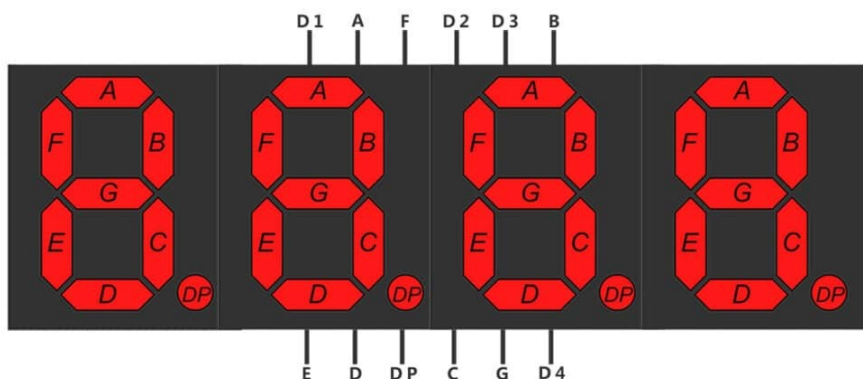
```

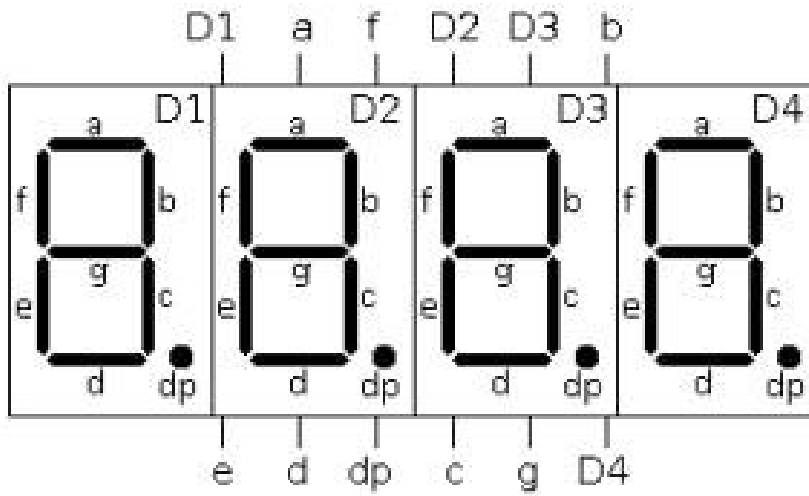
for(int i=0;i<7;i++) {
pinMode(seg[i], OUTPUT);
  pinMode(10, OUTPUT); // Red
  pinMode(11, OUTPUT); // Green
}
void loop() {
  digitalWrite(10, HIGH);
  displayDigit(1);
  delay(2000);
  digitalWrite(10, LOW);
  digitalWrite(11, HIGH);
  displayDigit(2);
  delay(2000);
  digitalWrite(11, LOW);
  displayDigit(0);
  delay(2000);
}

```

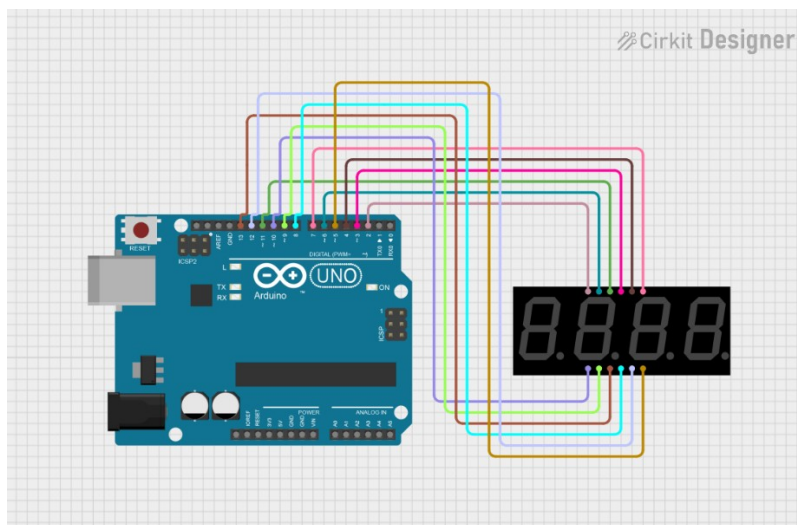
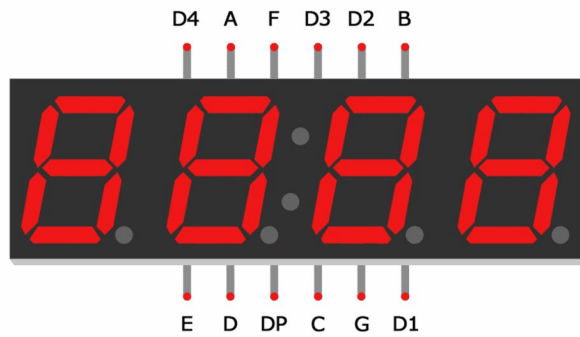
Code 2:

4-digit Seven segment display (common CATHODE)





4-digit Seven segment display (common Anode)



```

#include "SevSeg.h"
SevSeg sevseg; //Instantiate a seven segment controller object

void setup() {
  byte numDigits = 4;
  byte digitPins[] = {2, 3, 4, 5};
  byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13};
  bool resistorsOnSegments = false; // 'false' means resistors are on digit pins
  byte hardwareConfig = COMMON_ANODE;
  bool updateWithDelays = false; // Default 'false' is Recommended
  bool leadingZeros = false; // Use 'true' if you'd like to keep the leading zeros
  bool disableDecPoint = false; // Use 'true' if your decimal point doesn't exist or
  isn't connected

  sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins,
  resistorsOnSegments,
  updateWithDelays, leadingZeros, disableDecPoint);
  sevseg.setBrightness(90);
}

void loop() {
  static unsigned long timer = millis();
  static int deciSeconds = 0;

  if (millis() - timer >= 100) {
    timer += 100;
    deciSeconds++; // 100 milliSeconds is equal to 1 deciSecond

    if (deciSeconds == 10000) { // Reset to 0 after counting for 1000 seconds.
      deciSeconds=0;
    }
    sevseg.setNumber(deciSeconds, 1);
  }
  sevseg.refreshDisplay(); // Must run repeatedly
}

```

Code 3: Arduino Program (Common Cathode – Display 1234)

```

#include "SevSeg.h"
SevSeg sevseg;

void setup() {
  byte numDigits = 4;

```

```

    byte digitPins[] = {2, 3, 4, 5};
    byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13};
    bool resistorsOnSegments = 0;
    sevseg.begin(COMMON_ANODE,      numDigits,      digitPins,      segmentPins,
    resistorsOnSegments);
    sevseg.setBrightness(60);
}

```

```

void loop() {
    sevseg.setChars("1234"); // set the characters/numbers to display.
    sevseg.refreshDisplay();}

```

code 4:

```

int segPins[] = {6,7,8,9,10,11,12}; // a b c d e f g
int digitPins[] = {2,3,4,5}; // D1 D2 D3 D4

// Segment patterns for common anode (LOW = ON)
byte numbers[10][7] = {
    {0,0,0,0,0,0,1}, // 0
    {1,0,0,1,1,1,1}, // 1
    {0,0,1,0,0,1,0}, // 2
    {0,0,0,0,1,1,0}, // 3
    {1,0,0,1,1,0,0}, // 4
    {0,1,0,0,1,0,0}, // 5
    {0,1,0,0,0,0,0}, // 6
    {0,0,0,1,1,1,1}, // 7
    {0,0,0,0,0,0,0}, // 8
    {0,0,0,0,1,0,0} // 9
};

```

```

void setup() {
    for(int i=0;i<7;i++)
        pinMode(segPins[i], OUTPUT);

    for(int i=0;i<4;i++)
        pinMode(digitPins[i], OUTPUT);
}

void loop() {
    for(int count = 0; count <= 205; count++) {
        unsigned long startTime = millis();
        while(millis() - startTime < 100) { // display each number for 1 second
            displayNumber(count);
        }
    }
}

// ----- Display Functions -----

void displayNumber(int num) {
    int d[4];
    d[0] = num / 1000;
    d[1] = (num / 100) % 10;
    d[2] = (num / 10) % 10;
    d[3] = num % 10;

    for(int i=0;i<4;i++) {
        digitalWrite(digitPins[i], HIGH); // enable digit (CA)
        showDigit(d[i]);
        delay(5);
        digitalWrite(digitPins[i], LOW); // disable digit
    }
}

```

```

    }
}

void showDigit(int n) {
    for(int i=0;i<7;i++) {
        digitalWrite(segPins[i], numbers[n][i]);
    }
}

```

13.5 Procedure

1. Wire the seven segment and LEDs as per instructions.
2. Upload the code.
3. Observe LED ON/OFF state with segment display changing accordingly.

13.6 Observations

Display	LED State
1	Red ON
2	Green ON
0	All LEDs OFF

13.7 Review Questions

1. What is the difference between common cathode and anode displays?
2. How are digits displayed using code?
3. Can we connect multiple 7-segments?
4. How do you display characters like A, b, C?

13.8 Practical exercises

Q1.

Implement a project for chronometer that can be used by a trainer in different sport to indicate the time that a person doing sport is using. The chronometer is designed to display the maximum time which is 1minute 30 seconds.

Q2. Count from 200 to 0

Experiment Fourteen: Interfacing LED Matrix/Dot matrix with arduino

14.1 Objective

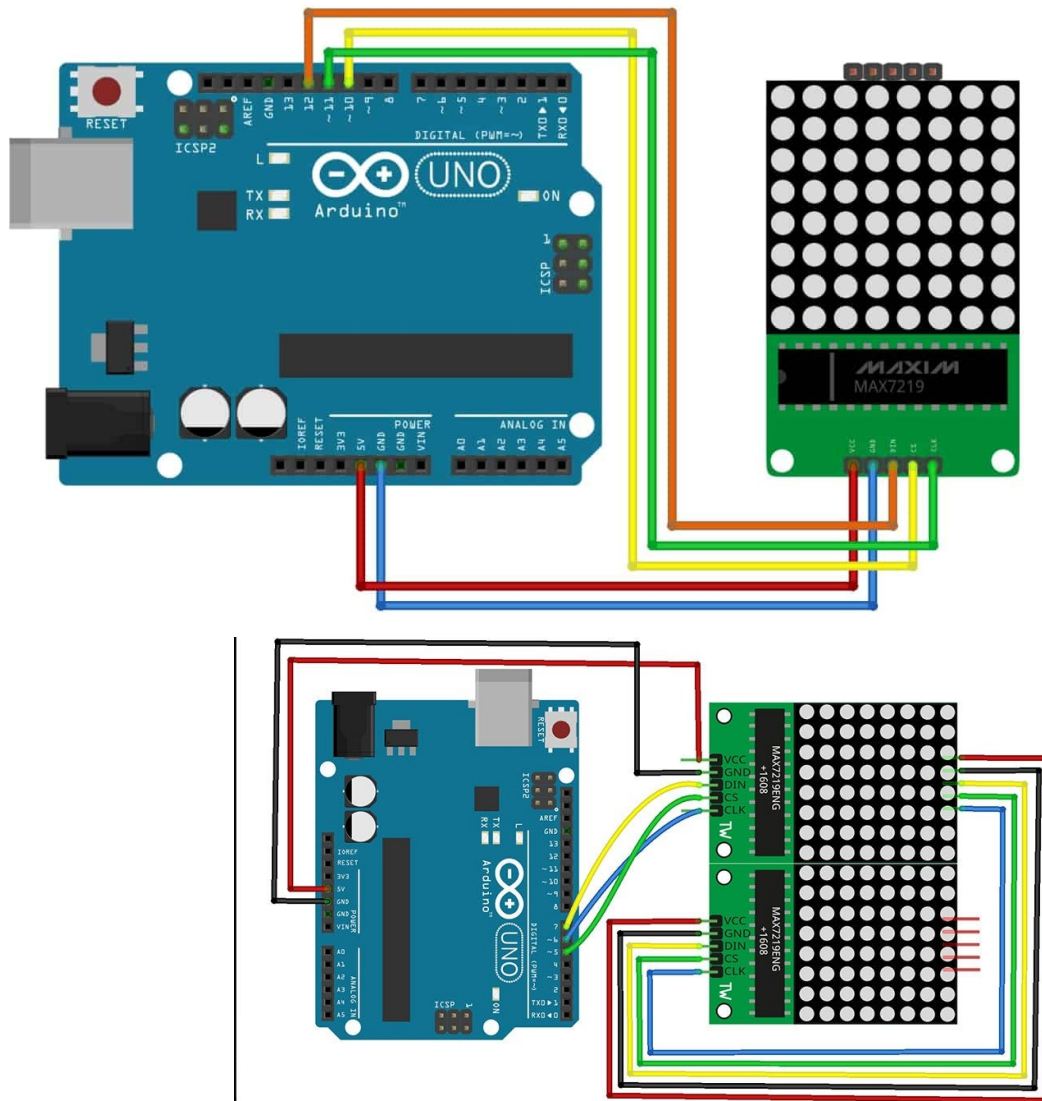
To interface an 8×8 LED dot matrix display with an Arduino and display alphabetical letters from A to I sequentially.

14.2 Components Required

- Arduino Uno / Nano
- **8×8 LED Matrix with MAX7219 driver**
- Jumper wires
- USB cable

14.3 Circuit Diagram Description

MAX7219 Pin	Arduino Uno
VCC	5V
GND	GND
DIN	D11
CS	D10
CLK	D13



Install Required Library

In **Arduino IDE**:

1. Go to **Sketch** → **Include Library** → **Manage Libraries**
2. Search for **“LedControl”**
3. Install **LedControl by Eberhard Fahle**

14.4 Arduino Code

Code 1:

```
#include <LedControl.h>

// DIN, CLK, CS
LedControl lc = LedControl(11, 13, 10, 1);
```

```

// Alphabet patterns (8x8)
byte letters[10][8] = {
  {0x18,0x24,0x42,0x7E,0x42,0x42,0x42,0x00}, // A
  {0x7C,0x42,0x42,0x7C,0x42,0x42,0x7C,0x00}, // B
  {0x3C,0x42,0x40,0x40,0x40,0x42,0x3C,0x00}, // C
  {0x78,0x44,0x42,0x42,0x42,0x44,0x78,0x00}, // D
  {0x7E,0x40,0x40,0x7C,0x40,0x40,0x7E,0x00}, // E
  {0x7E,0x40,0x40,0x7C,0x40,0x40,0x40,0x00}, // F
  {0x3C,0x42,0x40,0x4E,0x42,0x42,0x3C,0x00}, // G
  {0x42,0x42,0x42,0x7E,0x42,0x42,0x42,0x00}, // H
  {0x3C,0x18,0x18,0x18,0x18,0x18,0x3C,0x00}, // I
  {0x1E,0x04,0x04,0x04,0x44,0x44,0x38,0x00} // J
};

void setup() {
  lc.shutdown(0, false);
  lc.setIntensity(0, 8);
  lc.clearDisplay(0);
}

void loop() {
  for (int i = 0; i < 10; i++) {
    for (int row = 0; row < 8; row++) {
      lc.setRow(0, row, letters[i][row]);
    }
    delay (800); // display time per letter
    lc.clearDisplay(0);
  }
}

```

14.5 Observations

Letter Displayed	Observation
A	Letter A is clearly visible on the matrix
B	Letter B is displayed after a short delay
C	Letter C appears correctly
D	Letter D is displayed
E	Letter E is displayed
F	Letter F is displayed
G	Letter G is displayed
H	Letter H is displayed

I	Letter I is displayed
Sequence	Letters appear in correct alphabetical order

14.6 Review Questions

1. What is an LED dot matrix?
2. Why is the MAX7219 driver IC used?
3. What is multiplexing in LED displays?
4. How many rows and columns are present in an 8×8 LED matrix?
5. How is an alphabet character represented in an LED matrix?
6. What is the function of the delay () function in this experiment?
7. How can brightness of the LED matrix be controlled?

14.7 Conclusion

Interfacing an LED dot matrix with Arduino using the MAX7219 driver is simple and efficient. The experiment demonstrates how **bitmap patterns** can be used to display characters. Sequential display of alphabets helps in understanding **display control, timing, and embedded programming concepts**.

14.8 Practical exercises

1. Perform the following:
 - ✓ Modify the program to display alphabets from **A to Z**.
 - ✓ Change the delay time to control the display speed.
 - ✓ Display only **vowel letters** on the LED matrix.
 - ✓ Create a **scrolling text effect** for alphabets.
 - ✓ Display **numbers (0–9)** instead of alphabets.
2. Arduino sketch that reproduces this exact style on an 8×8 LED dot-matrix using MAX7219, with walking animation (moving legs & arms, Pedestrian Walking Man (Traffic Light Style)).
3. Interface the DHT22 Temperature & Humidity Sensor with Arduino & display the sensor value on 8×32 Dot Matrix LED Display.
4. With an 8×8 LED dot-matrix, displaying Dotted Heart

Experiment Fifteen: Interfacing RFID with Arduino to Control LEDs

15.1 Objective

To read RFID card UID using Arduino and use it to turn ON/OFF LEDs.

15.2 Components Required

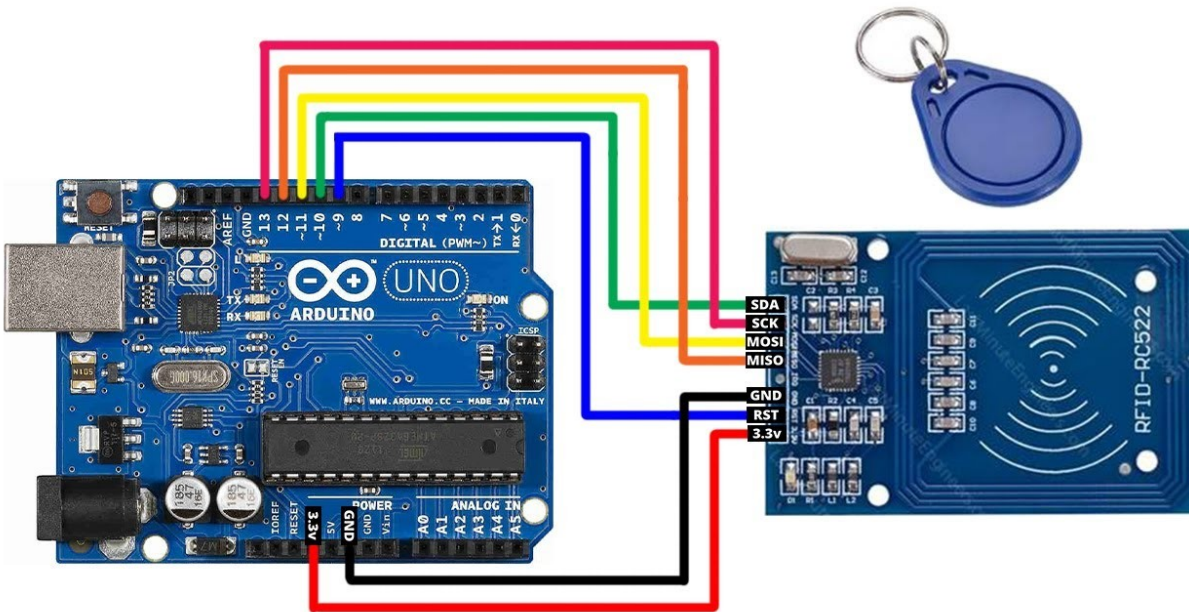
Component	Quantity
Arduino Uno	1
RC522 RFID Reader	1
RFID Card/Tag	1
LEDs	2
220Ω Resistors	2
Jumper wires	As needed

15.3 Circuit Diagram Description

- Connect RFID to Arduino using SPI: SDA → D10, SCK → D13, MOSI → D11, MISO → D12, RST → D9 (for arduino Uno)
- For other types of arduino boards, refer to the following table showing typical pin layout used.
- Connect LEDs to D6 and D7

* Typical pin layout used:

* -----						
* MFRC522	Arduino	Arduino	Arduino	Arduino	Arduino	
* Reader/PCD	Uno/101	Mega	Nano v3	Leonardo/Micro	Pro Micro	
* Signal Pin	Pin	Pin	Pin	Pin	Pin	
* -----						
* RST/Reset RST	9	5	D9	RESET/ICSP-5	RST	
* SPI SS SDA(SS)	10	53	D10	10	10	
* SPI MOSI MOSI	11 / ICSP-4	51	D11	ICSP-4	16	
* SPI MISO MISO	12 / ICSP-1	50	D12	ICSP-1	14	
* SPI SCK SCK	13 / ICSP-3	52	D13	ICSP-3	15	
*						



<https://arduinointro.com/articles/projects/how-to-use-rfid-rc522-with-arduino-a-complete-beginners-guide>

15.4 Arduino Code

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
MFRC522 rfid(SS_PIN, RST_PIN);
byte validCard[4] = {0xDE, 0xAD, 0xBE, 0xEF}; // Replace with your tag UID
int redLED = 6;
int greenLED = 7;
void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
}
```

```

void loop() {
  if(!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) return;

  boolean match = true;
  for (int i = 0; i < 4; i++) {
    if(rfid.uid.uidByte[i] != validCard[i]) {
      match = false; break;
    }
  }
  if(match) {
    digitalWrite(greenLED, HIGH);
    digitalWrite(redLED, LOW);
  } else {
    digitalWrite(redLED, HIGH);
    digitalWrite(greenLED, LOW);
  }
  delay(2000);
  digitalWrite(redLED, LOW);
  digitalWrite(greenLED, LOW);
  rfid.PICC_HaltA();
}

```

7. Procedure

1. Connect RFID and LEDs to Arduino.
2. Upload code and open Serial Monitor.
3. Tap RFID tag. LEDs turn ON/OFF based on UID.

15.5 Observations

Card UID Match	LED ON
Yes	Green
No	Red

15.6 Review Questions

1. What is UID in RFID?
2. What protocol does RFID use?
3. Can RFID be used for attendance systems?
4. What's the range of RC522?

15.7 Conclusion

RFID is a secure way to control access and trigger outputs like LED indicators.

15.8 Practical exercises

1. Conduct a project with arduino and RFID to turn green ON LED if the proper card is used and print in LCD “ACCESS **APPROVED**” and if the card is wrong, buzzer will be ON and LCD displays “**ACCESS DENIED**”.

2. Project Scenario: Arduino-Based RFID Access and Power Management System for Library Automation at RP-Gishari

An **Arduino-based RFID system** can be applied in a **smart access control system** for a college campus. In this scenario, RFID tags are issued to students, faculty, and staff as part of their ID cards. At entry points to restricted areas such as laboratories, libraries, or administrative offices, RFID readers connected to Arduino boards are installed. When a user scans their RFID tag near the reader, the Arduino processes the tag's unique ID and checks it against a preloaded database. If the tag is authorized, the system triggers a servo motor to unlock the door, logs the entry time, and displays a welcome message on an LCD screen. Unauthorized attempts are logged for security monitoring, and access is denied. This setup ensures efficient, automated, and secure access management while providing real-time monitoring capabilities.

As an Electrical and Electronics Engineering student, you are tasked with designing and implementing an **Arduino-based RFID system** aimed at automating access control and power conservation in the **RP-Gishari College Library**. The system will serve self-studying students and ensure efficient use of electrical energy. The main objective is to enhance security and energy efficiency during students' independent study sessions.

The system requirements include:

- **RFID Authentication:** Only authorized students with valid RFID cards can access the library.
- **Automated Door Control:** Upon scanning a valid RFID card, the door opens and remains open for **10 seconds**, after which it automatically closes.
- **Lighting Control:** The library lamps should automatically turn on when access is granted, providing light only when the space is occupied.
- **Invalid Access Alert:** If an unauthorized card is used:
 - ✓ A **buzzer** should sound.
 - ✓ A **Red LED** should turn on as a visual warning.

Experiment Sixteen: Interfacing GSM Module to Control LEDs via SMS

16.1 Objective

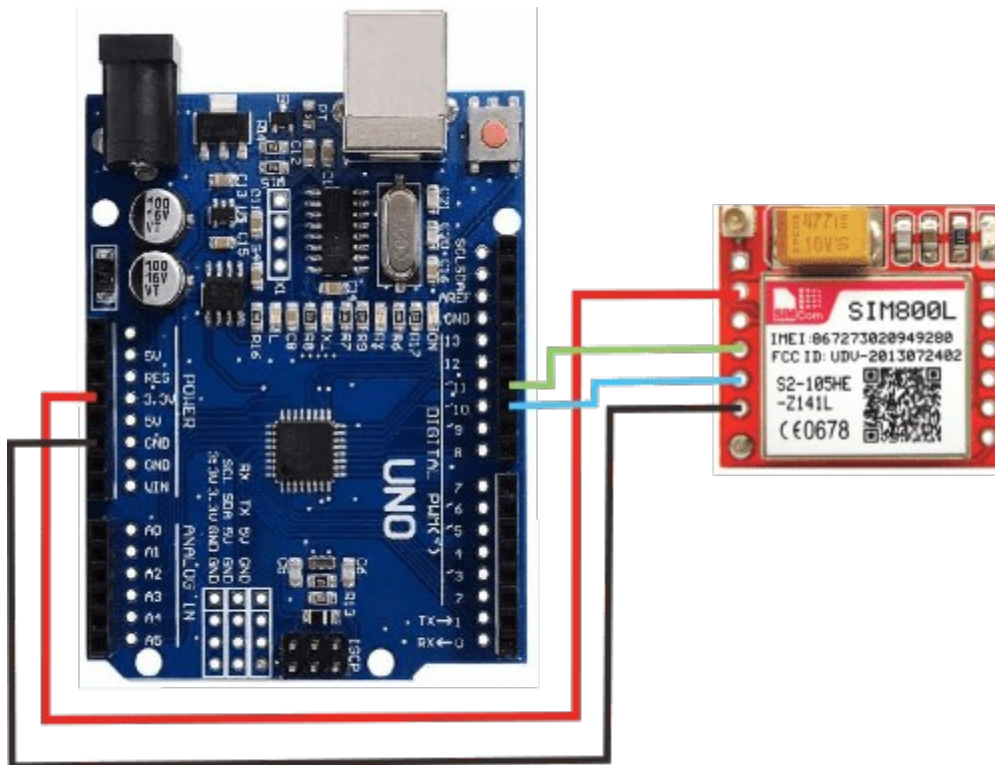
To send SMS commands from a phone to control LEDs connected to Arduino via a GSM module.

16.2 Components Required

Component	Quantity
Arduino Uno	1
GSM SIM800L	1
SIM Card	1
LEDs	2
Resistors 220 Ω	2
Level shifter or voltage divider	Optional
Jumper wires	As needed

16.3 Circuit Diagram Description

- Connect GSM TX to Arduino RX (D2), GSM RX to Arduino TX (D3) via software serial
- Power GSM with external supply (3.7–4.2V)
- LEDs to D10 and D11



<https://miliohm.com/sim800l-arduino-tutorial/>

16.3 Arduino Code

```
#include <SoftwareSerial.h>
SoftwareSerial gsm(2, 3);
String text = " ";
int led1 = 10;
int led2 = 11;
void setup() {
  gsm.begin(9600);
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
```

```

    pinMode(led2, OUTPUT);
}

void loop() {
    if(gsm.available()) {
        text = gsm.readString();
        Serial.println(text);
        if(text.indexOf("ON1") >= 0){
            digitalWrite(led1, HIGH);}
        if(text.indexOf("OFF1") >= 0) {
            digitalWrite(led1, LOW);}
        if(text.indexOf("ON2") >= 0){
            digitalWrite(led2, HIGH);}
        if(text.indexOf("OFF2") >= 0) {
            digitalWrite(led2, LOW);}
    }
}

```

16.4 Procedure

1. Insert SIM card and power GSM.
2. Upload code to Arduino.
3. Send SMS with content “ON1” or “OFF1” to control LED1, and so on.

16.5 Observations

SMS Text	LED State
ON1	LED1 ON
OFF1	LED1 OFF
ON2	LED2 ON
OFF2	LED2 OFF

16.6 Review Questions

1. What is the function of a GSM module?
2. Why do we use SoftwareSerial?
3. Can GSM modules receive voice?

4. How to secure GSM-based systems?

16.7 Conclusion

This experiment demonstrates remote control of devices using GSM, useful for rural automation and security systems.

“Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do.” — Late Pelé, Brazilian pro footballer.

..... The end.....