**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**


# 4-Bit Arithmetic Logic Unit

## Logic & Sequential Circuit Design - EC210

**SUBMITTED TO:**
**Prof. Dr. Muhammad Salman LE**
**Osama Shaukat**

**SUBMITTED BY:**

**NOMAN ALI - 502241**
**SHAZIL - 532263**
**TEHREEM ASIF –504221**
**MARYAM ZAFAR - 503495**

**Submission Date: 15th May 2025**

## Introduction

This project involves the design and implementation of an 8-function arithmetic and logical unit (ALU). The ALU is capable of performing four arithmetic operations (addition, subtraction, incrementing by 1, and decrementing by 1) and four logical operations (AND, OR, XOR, XNOR). The design is implemented both in simulation using Proteus and as a hardware prototype.

## Components Used

- Integrated Circuits (ICs)
    - AND Gate - 7408
    - OR Gate – 7432
    - NOT Gate – 7404
    - Multiplexer – 74151
    - Demultiplexer – 73157
    - 4-bit Binary Adder – 7483
    - NAND Gate – 7400
    - XOR Gate – 7486
    - BCD to 7-segment Decoder – 7447
    - Clock IC-74193

## Other Components

- Common anode 7-segment displays
- Resistors, capacitors, and connecting wires
- Breadboard for prototyping

## Project Steps

1. Design and Simulation in Proteus

- Circuit Design: Using Proteus, design the circuit schematics for the ALU, incorporating all required ICs and their interconnections.

2. Logic Section Simulation:

- Design the logic operations (AND, OR, XOR, XNOR) using the respective ICs (7408, 7432, 7486, 7400).
- Test each operation individually by providing 4-bit inputs and verifying the 4-bit outputs.

3. Arithmetic Section Simulation:

- Implement addition and subtraction using the 7483 IC.
- Design increment and decrement operations using additional logic gates.
- Simulate each arithmetic function to ensure correctness.

4. Multiplexer and Control Logic:

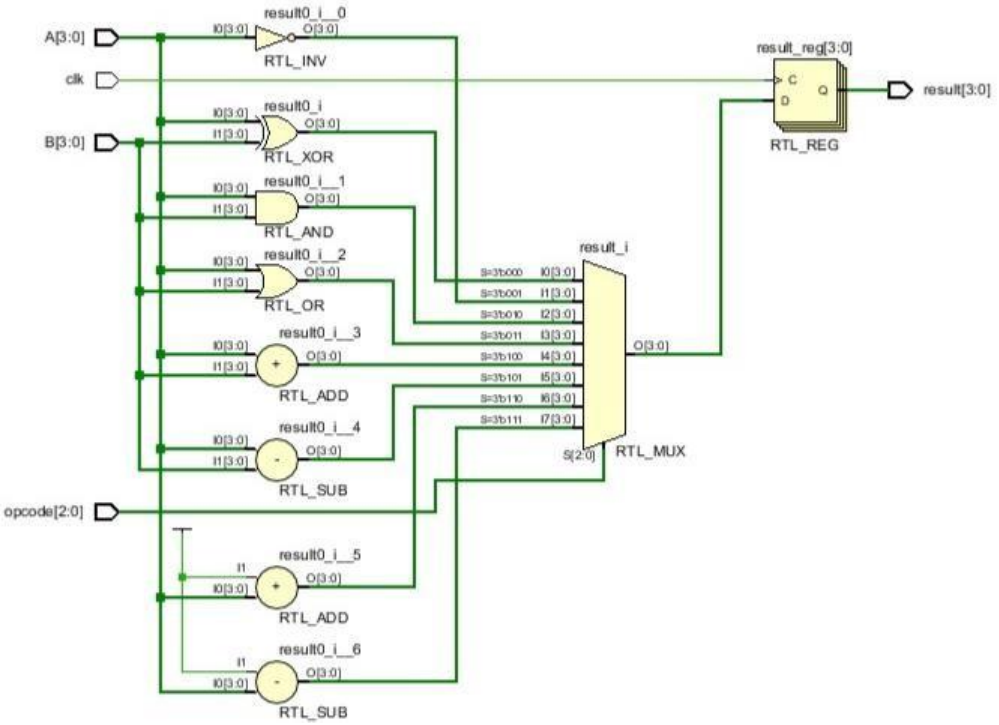
- Use the 74151 multiplexer (MUX) to select one of the eight operations based on the 3-bit operation code.
- Simulate the selection process by varying the operation code and verifying the output.

5. BCD to 7-segment Display:

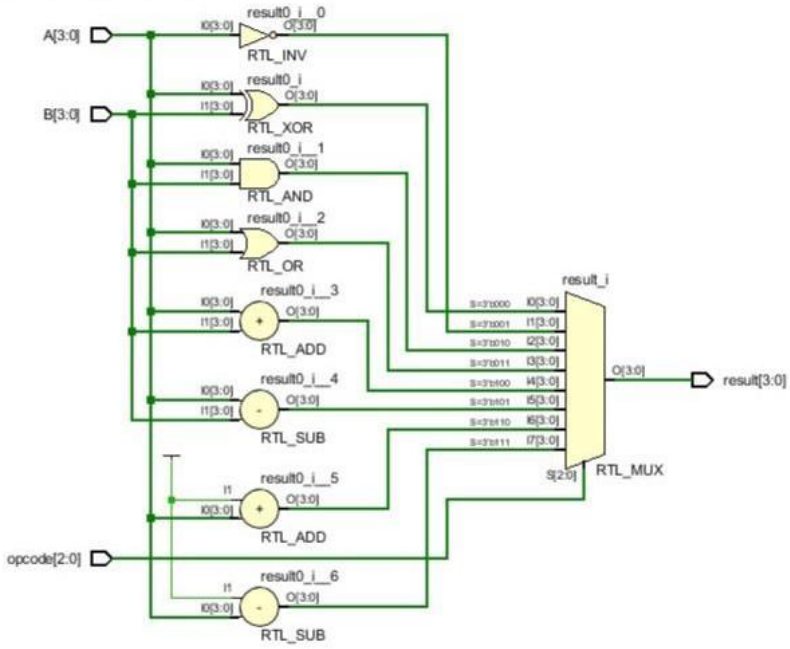
- Connect the outputs to 7447 BCD to 7-segment decoders.
- Simulate the display of results on the common anode 7-segment displays.

## SCHEMATIC:

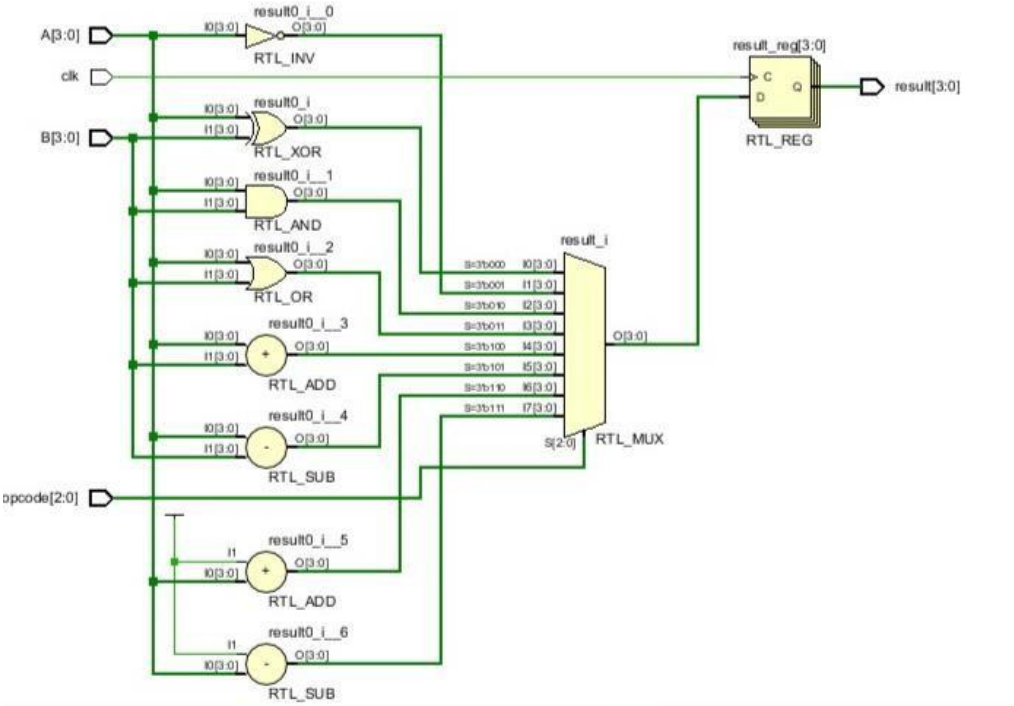
### Level 1:



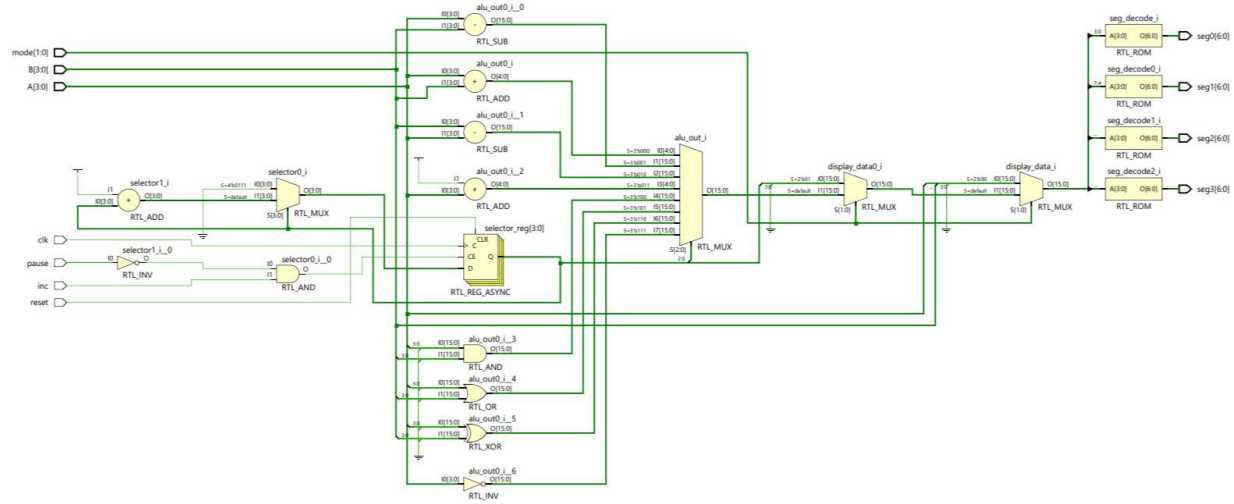

### LEVEL 2:



## LEVEL 3:




## FINAL SCHEMATIC:

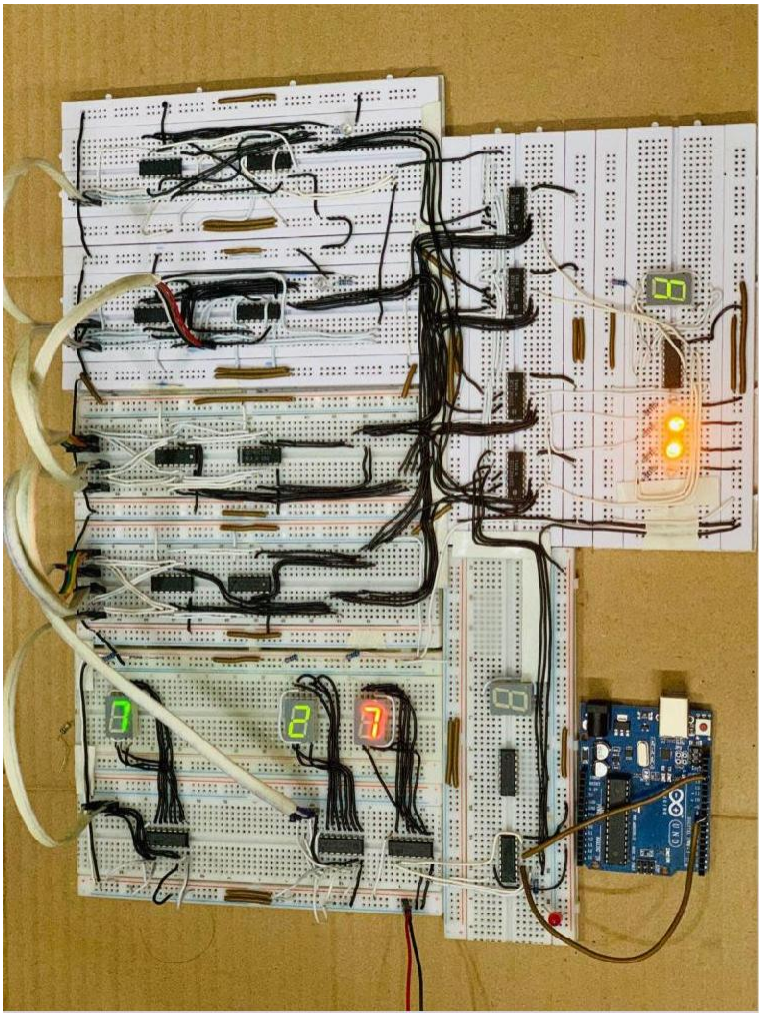


## Hardware Implementation

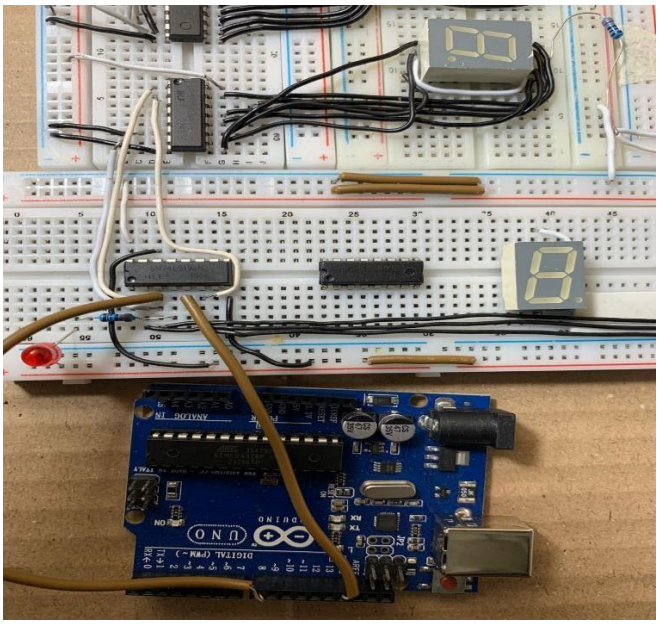
1. Component Assembly:
   - Place all ICs on the breadboard and connect them as per the Proteus design.
   - Ensure proper power supply connections to each IC.
2. Input and Operation Code Setup:
   - Connect 4-bit binary switches for the two input numbers.
   - Use a 3-bit binary switch for the operation code.
3. Output Display:
   - Connect the 4-bit output to the 7447 BCD decoders and then to the 7-segment displays.
4. Verification and Testing:

Verify each operation (AND, OR, XOR, XNOR, addition, subtraction, increment, decrement) by providing appropriate inputs and checking the outputs on the 7-segment displays

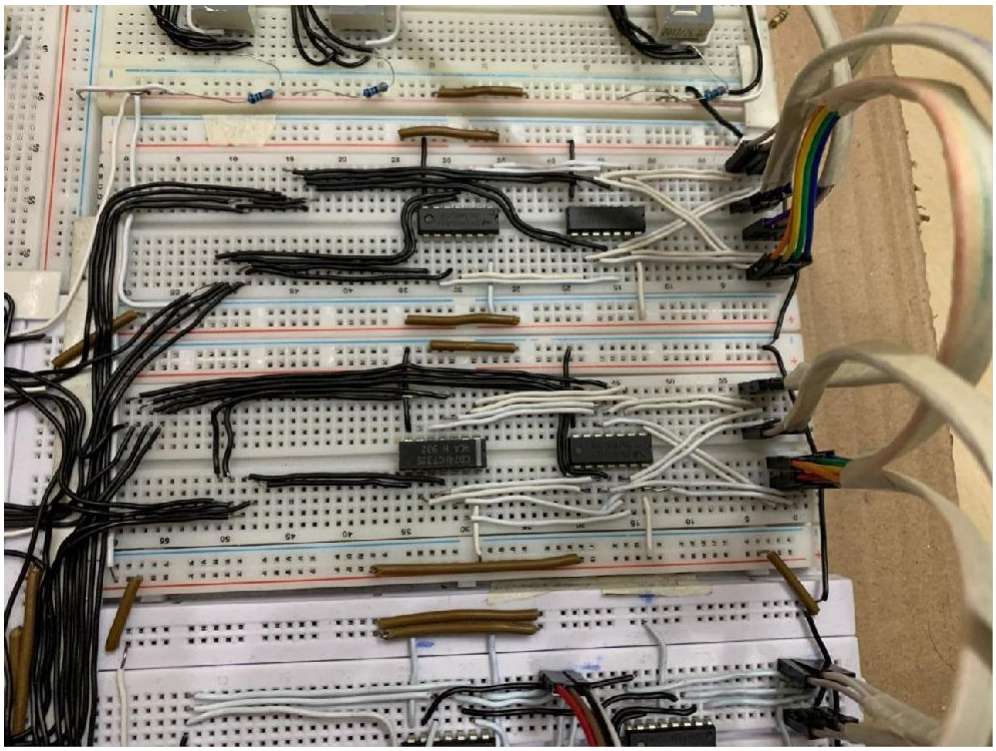
## FINAL CIRCUIT:



## CLOCK:

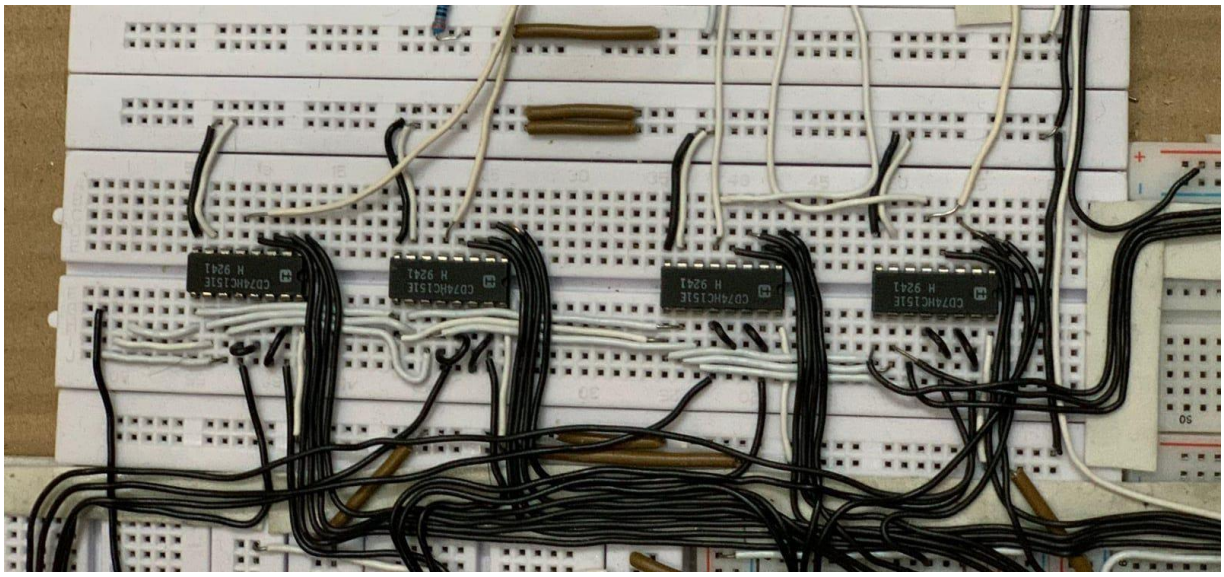


## LOGICAL OPERATIONS:

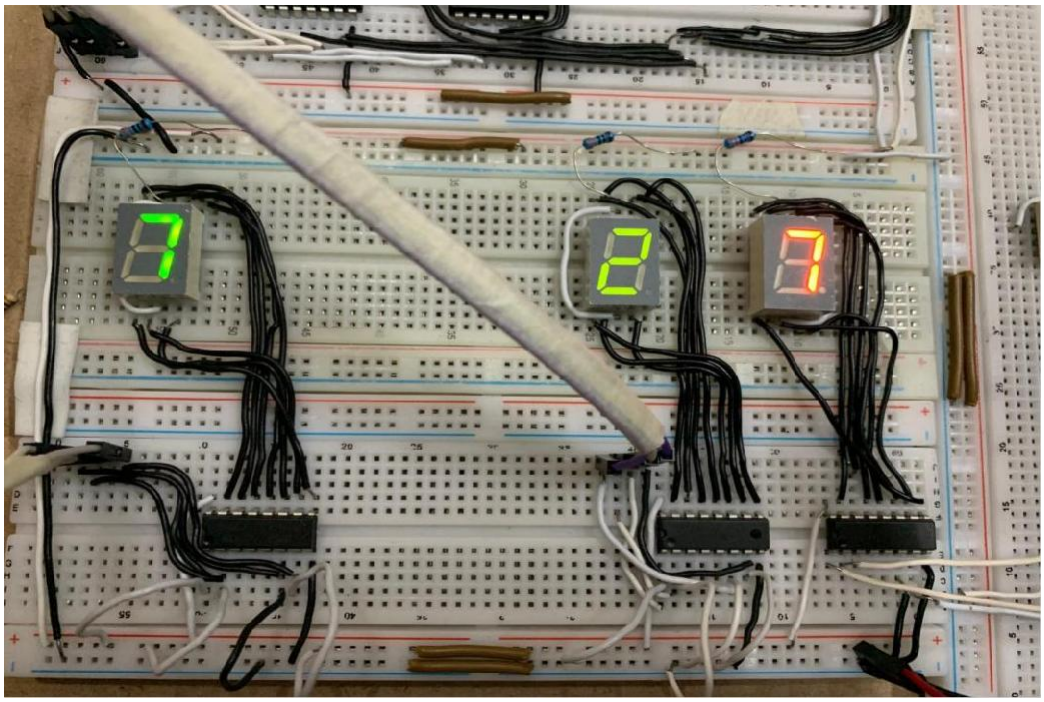


## ARTHEMETIC OPERATIONS:

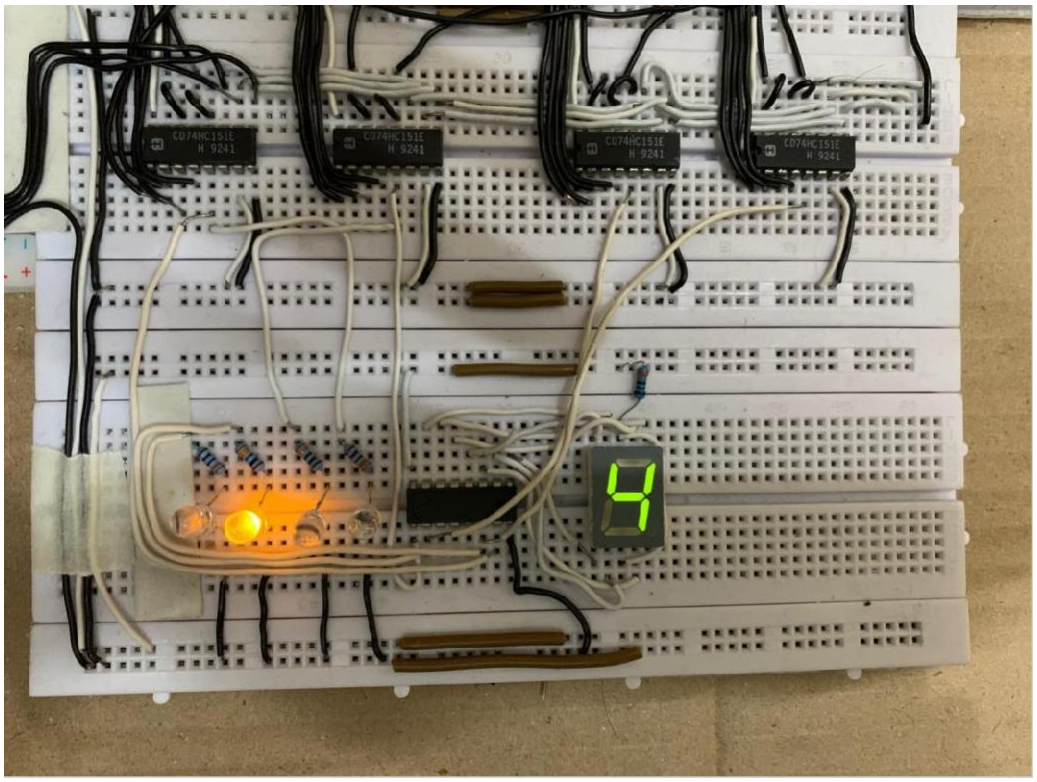
## MULTIPLEXER :



**INPUT:**




**OUTPUT:**



## Logical Section Explanation

**AND Operation**

- IC Used: 7408
- Operation: Each bit of the two 4-bit inputs is ANDed together.
- Example: 1010 AND 1100 = 1000

**OR Operation**

- IC Used: 7432
- Operation: Each bit of the two 4-bit inputs is ORed together.

Example: 1010 OR 1100 = 1110 **XOR Operation**

- IC Used: 7486
- Operation: Each bit of the two 4-bit inputs is XORed together.
- Example: 1010 XOR 1100 = 0110

**XNOR Operation**

- IC Used: 7400 (NAND gates configured as XNOR)
- Operation: Each bit of the two 4-bit inputs is XNORed together.
- Example: 1010 XNOR 1100 = 1001

# Arithmetic Section Explanation

**Addition**

- IC Used: 7483
- Operation: Adds two 4-bit numbers to produce a 4-bit sum and a carry out.
- Example: 1010 + 1100 = 0110 (with carry)

**Subtraction**

- IC Used: 7483 (configured for subtraction using 2's complement)
- Operation: Subtracts one 4-bit number from another.
- Example: 1010 - 1100 = 1110 (with borrow)

**Increment by 1**

- IC Used: 7483 (one input set to 0001)
- Operation: Adds 1 to the 4-bit number.
- Example: 1010 + 0001 = 1011

**Decrement by 1**

- IC Used: 7483 (one input set to 1111, effectively subtracting 1)
- Operation: Subtracts 1 from the 4-bit number.
- Example: 1010 - 0001 = 1001

## Op Code Mapping

- XOR → 000
- NOT → 001
- AND → 010
- OR → 011
- Add → 100
- Sub → 101
- Increment → 110
- Decrement → 111

## Selection Mechanism

The selection of the operation is controlled by a 3-bit Op Code input. This input is processed by a multiplexer (74151), which routes the inputs to the appropriate function blocks based on the Op Code. The 74151 IC is a 8-to-1 multiplexer that can select one of eight inputs to pass through to the output based on a 3-bit selector.

## Implementation Steps

1. **Multiplexer Setup**:

    - The 74151 multiplexer is configured with its inputs connected to the outputs of the different operation blocks (addition, subtraction, increment, decrement, AND, OR, XOR, XNOR).
    - The 3-bit Op Code is connected to the selection inputs of the multiplexer.

2. **Connecting the Operation Blocks**:

- **XOR (000)**: The result from the XOR operation (using 7486) is connected to the 0th input of the multiplexer.
- **NOT (001)**: The result from the NOT operation (using 7404) is connected to the 1st input of the multiplexer.
- **AND (010)**: The result from the AND operation (using 7408) is connected to the 2nd input of the multiplexer.
- **OR (011)**: The result from the OR operation (using 7432) is connected to the 3rd input of the multiplexer.
- **Add (100)**: The sum output from the 7483 4-bit adder is connected to the 4th input of the multiplexer.
- **Sub (101)**: The result from the subtraction block (configured using the 7483 adder in 2's complement) is connected to the 5th input of the multiplexer.
- **Increment (110)**: The output from the increment block (addition with a fixed input of 0001) is connected to the 6th input of the multiplexer.
- **Decrement (111)**: The output from the decrement block (addition with a fixed input of 1111) is connected to the 7th input of the multiplexer.

3. **Control Logic**:
    - The 3-bit Op Code is provided by external switches or a microcontroller, which allows the user to select the desired operation.
    - The multiplexer reads the Op Code and directs the appropriate operation's output to the ALU's final output.

4. **Output Display**:

    - The selected operation's result is then sent to the BCD to 7-segment decoders (7447) to display the output on common anode 7-segment displays.
    - Both inputs and the selected operation code are also displayed to provide a complete view of the operation being performed.

# CONTRIBUTION:

## Project Modules and Member Contributions

### 1. Input/Output Module – Handled by Noman

**Description:**
This module manages user input and output:

- **Inputs:** DIP switches for operand A, operand B, and OP-code selection
- **Outputs:** LEDs or 7-segment display for showing the result
- **OP-code Control:** Directs the multiplexer to select the correct operation output

**Contribution:**
Noman connected and organized all input and output components. He ensured that the user could select operations and operands easily, and verified that the correct outputs were displayed. His work laid the foundation for smooth user interaction with the ALU.

### 2. Clock, Multiplexer, and Final Assembly – Handled by Shazil

**Description:**
This module integrates and controls the full ALU system:

- **Multiplexer:** Selects the output of the correct operation based on the OP-code
- **Clock Circuit:** Generates timing/control signals if required
- **System Assembly:** Full connection and integration of logic, arithmetic, and I/O modules

**Contribution:**
Shazil handled the core integration of all components. He wired the multiplexer selection logic, built the clock circuitry where needed, and assembled the entire ALU system. He also conducted full system testing and debugging to ensure proper functionality.

### 3. Arithmetic Module – Handled by Tehreem

**Description:**
Handles all arithmetic operations using the 7483 4-bit binary adder:

- **Add (100)** – Standard binary addition
- **Sub (101)** – Implemented using 2's complement logic with the adder
- **Increment (110)** – Addition with binary `0001`
- **Decrement (111)** – Addition with binary `1111`

**Contribution:**
Tehreem designed and tested all arithmetic circuits, including proper configuration of 2's complement

subtraction. She ensured that each operation correctly routed its output to the multiplexer for final selection.

### 4. Logical Module – Handled by Maryam

**Description:**
This module performs all logic-based operations using standard logic ICs:

- **XOR (000)** – Using 7486
- **NOT (001)** – Using 7404
- **AND (010)** – Using 7408
- **OR (011)** – Using 7432

**Contribution:**
Maryam constructed and tested all logic operation circuits. She ensured that the correct outputs were produced for each logic operation and connected them properly to the multiplexer input lines.

## Teamwork Summary

Each team member took charge of a specific part of the ALU project, enabling efficient parallel development and clear ownership. Noman set up user interfacing, Shazil integrated all components and ensured correct operation through clock and MUX control, Tehreem built reliable arithmetic logic, and Maryam implemented essential logic operations. This well-structured teamwork resulted in a fully functional and tested ALU system.

## BONUS POINT:

In our 4-bit ALU implementation, we adopted a more efficient and integrated approach compared to conventional designs where separate circuits are used for addition, subtraction, increment, and decrement operations. Instead, we combined the adder and subtractor functionalities into a single 4-bit circuit using a control-driven logic mechanism. The core of this logic lies in the use of XOR gates connected to each bit of the second input (B) of the adder.
Input A is directly fed to the adder, while input B is passed through the XOR gates. A control signal determines the operation mode: when the control input is ‘0’, the XOR gates pass B unchanged, enabling a standard 4-bit addition. When the control input is ‘1’, the XOR gates invert B (1’s complement), and with the addition of a carry-in ‘1’, the adder performs a 2’s complement addition—effectively executing a 4-bit subtraction.

Although this control signal can be manually set using a wire connected to ground (for logic 0) or VCC (for logic 1), in our design, we implemented an automated approach. The control signal is generated dynamically through a counter and clock mechanism, allowing the ALU to automatically switch between operations at each level without manual intervention. This same

logic is extended to combine increment and decrement operations into a single circuit, further enhancing the efficiency, compactness, and automation of the overall 4-bit ALU design.

## Conclusion

The design and implementation of the 8-function ALU was successfully carried out through Proteus simulation and hardware prototyping. Each function was tested and verified to perform correctly, with results displayed on 7-segment displays using 7447 BCD decoders. The project demonstrated effective use of various logic and arithmetic ICs to achieve the desired operations