Arithmetical Logical Unit Design

LazR ('3')
Alexxia

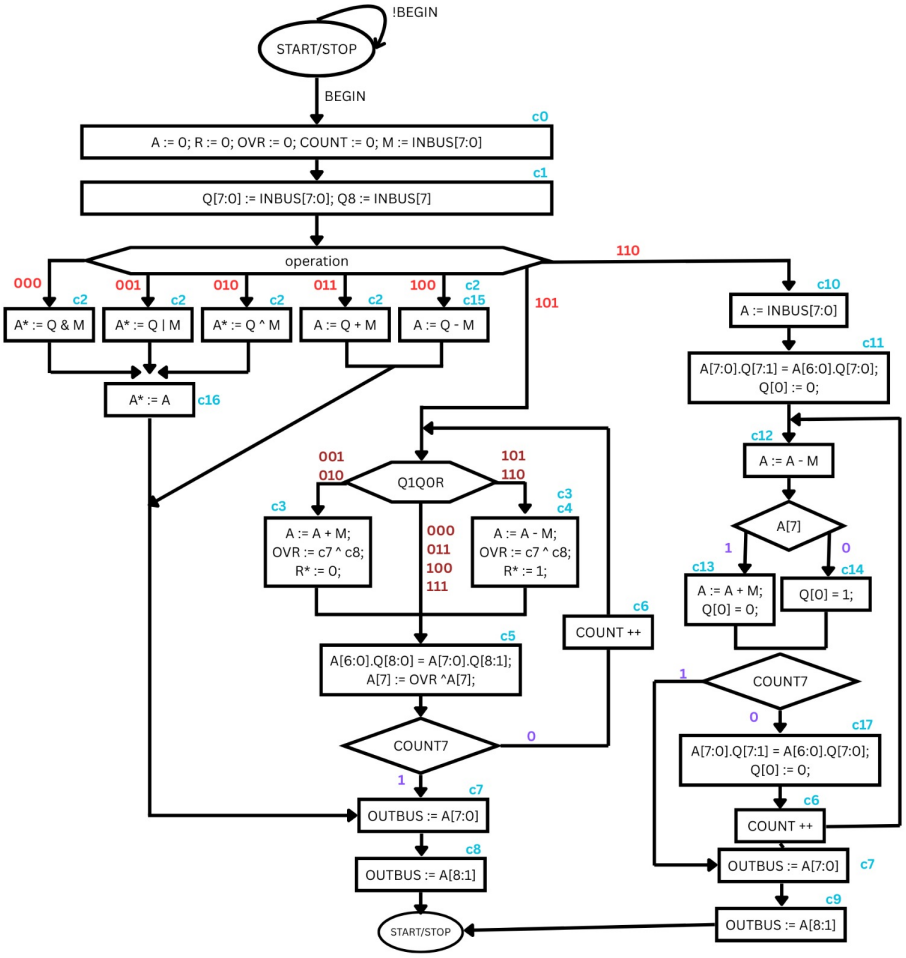
18th of April, 2025

Contents

1 Abstract

This project presents the design and implementation of an Arithmetic Logic Unit (ALU) capable of performing both logical and arithmetic operations on two 8-bit operands. The ALU supports basic logical operations including AND, OR, and XOR, as well as fundamental arithmetic operations such as addition, subtraction, multiplication, and division. The goal of this project is to show how these basic operations can be combined into a simple digital system. The project was developed by Maria - Alexia Crișan and Lazăr - Iulian Boariu as part of the Calculatoare Numerice course.

2 Finite State Diagram



The diagram illustrates the overall behavior of the ALU, detailing the steps involved in processing data and selecting the appropriate operation based on the provided operation code.

The design begins with the initialization of registers and variables, including the two 8-bit operands (Q and M), a remainder register (R), overflow flag (OVR), and a counter. The control logic then decodes the operation code to determine which operation to execute—logical (AND, OR, XOR) or arithmetic (addition, subtraction, multiplication, division). Each operation follows a specific path in the flowchart, with conditional branches guiding the transitions between processing stages.

Multiplication is implemented using the Modified Booth algorithm, which improves performance by reducing the number of partial products, and is triggered by specific control signals. Similarly, division is handled using the Restoring Division method, as shown on the right side of the diagram. This method involves conditional subtraction and shifting, iteratively refining the quotient and remainder.

The process continues until the counter reaches its limit, after which the final result is written to the output bus. The flowchart provides a structured overview of how the ALU manages complex operations in a controlled and sequential manner.

3 Algorithms Used

Modified Booth Multiplication

The Modified Booth algorithm is an efficient technique used for signed binary multiplication. It works by encoding the multiplier in groups of three bits, allowing the algorithm to reduce the number of partial products that need to be generated and added. Instead of performing a simple shift-and-add for every bit of the multiplier, Booth encoding identifies patterns and replaces them with fewer arithmetic operations, such as subtracting or addition. This method increases speed and reduces the hardware complexity of the multiplier, making it especially useful for larger word sizes or performance-critical designs.

Restoring Division

Restoring division is a classic algorithm used for performing binary division by sequentially subtracting the divisor from the partial remainder. At each step, the dividend is shifted left, and the divisor is subtracted from the current remainder. If the result is positive, the subtraction was valid, and the quotient bit is set to 1. If the result is negative, the original remainder is restored by adding the divisor back, and the quotient bit is set to 0.

4 ALU Design

5 Control Unit

The control unit is a key component of the ALU system, responsible for coordinating the execution of operations by generating the necessary control signals at each step. In this project, the control logic is implemented using a sequence counter, which acts as a simple state machine. Each state corresponds to a specific stage in the execution process, such as loading operands, performing the selected operation, or storing the result. By advancing through its states in a predefined order, the sequence counter ensures that all operations are carried out in the correct sequence and with proper timing. This approach simplifies the control logic and makes the overall system easier to manage and debug.

To implement the control logic, a Modulo-5-sequence counter was designed, which cycles through five distinct states corresponding to the stages of the ALU operation. This module serves as the core timing mechanism of the control unit. The system is composed of several interconnected submodules: Decoder-1-out-of-5, a Modulo-5-counter, and an SR flip-flop. Each component plays a specific role in generating and controlling the state transitions.

1. **Modulo-5-counter**

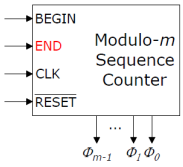
This module generates a repetitive sequence of five binary values (from 0 to 4) that represent the current phase. It increments its output on each clock cycle, looping back to zero after reaching four. The modulo-5 behavior ensures that the control unit progresses through exactly five phases before restarting the sequence.

2. **Decoder-1-out-of-5**

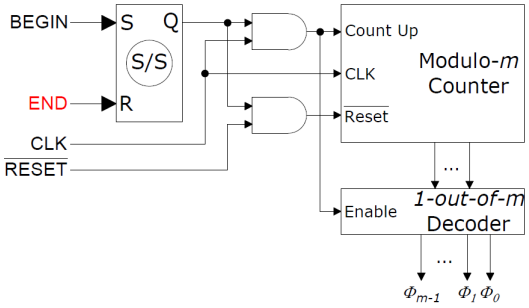
The decoder takes the 3-bit output from the modulo-five counter and activates exactly one of its five outputs at a time. This ensures that only one phase signal is active during each step, allowing precise control over which control lines are enabled. Essentially, it translates the binary count into a one-hot encoding format, which is easier to use for controlling sequential logic.

3. **SR Flip-Flop**

The Start/Stop latch controls whether the sequence counter is active or halted. It enables or disables the advancement through the counting sequence based on external conditions or control inputs. When the latch is set, the modulo-5 counter is allowed to advance on each clock cycle, progressing through the defined control states. When reset, the counter is held in place, effectively pausing the control unit. This mechanism provides a simple and effective way to synchronize the ALU's operation with the system's global control logic.



The picture illustrates the internal structure and signal connections within the Modulo-5-sequence counter used in the control unit. It shows how the Modulo-5-counter, 1-out-of-5 Decoder and the Start/Stop latch are connected to manage state transitions.



Each operation supported by the ALU has a specific set of control signals that are activated during certain steps of the control sequence. These signals

coordinate actions such as loading operands, enabling the ALU, performing addition or subtraction, and storing the result. The list below outlines the control signals associated with each operation and indicates during which phase (based on the output of the Modulo-5 sequence counter) each signal is active:

Control Signals by Operation

1. AND / OR / XOR (Logical Operations)

Operation codes: AND(000), AND(001), XOR(010)

Cycle 0:

- Phase 0: c0 -> Load operand M
- Phase 1: c1 -> Load operand Q
- Phase 2: c2 -> Enable logic unit
- Phase 3: c16 -> Store result in A-star based of off op

Cycle 9:

- Phase 0: c7 -> Load result A
- Phase 1: c8 -> Load result Q

———> Activate END signal

2. ADD / SUB (Arithmetic Operations)

Operation codes: ADD(011), SUB(100)

Cycle 0:

- Phase 0: c0 -> Load operand M
- Phase 1: c1 -> Load operand Q
- Phase 2: c2 -> Triggers addition, the result is stored in A

- Phase 2: c15 -> Triggers subtraction, the result is stored in A

Cycle 9:

- Phase 0: c7 -> Load result A
- Phase 1: c8 -> Load result Q

———> Activate END signal

3. MULTIPLICATION (Modified Booth)

Operation codes: MULTIPLICATION(101)

Cycle 0:

- Phase 0: c0 -> Load operand M and initialize A, COUNT, R, OVR
- Phase 1: c1 -> Load operand Q and Q8

Cycle 1 - 8:

- Phase 0: c3 -> Triggers addition for Q1Q0R in 001, 010, 101, 110, R* = 0, OVR is calculated
- Phase 0: c4 -> Additional signal for subtraction for Q1Q0R in 101, 110, R* = 1, OVR is calculated
- Phase 1: c5 -> Triggers right shift
- Phase 2: c6 -> COUNT is updated

Cycle 9:

- Phase 0: c7 -> Load result A
- Phase 0: c5 -> Triggers final right shift
- Phase 1: c8 -> Load result Q

———> Activate END signal

4. DIVISION (Restoring Division)

Operation codes: DIVISION(110)

Cycle 0:

- Phase 0: c0 -> Load operand M
- Phase 1: c1 -> Load operand Q
- Phase 2: c10 -> Load operand A (first half of the dividend)
- Phase 3: c11 -> Triggers initial left shift

Cycle 1 - 8:

- Phase 0: c12 -> Triggers subtraction
- Phase 1: c13 -> Triggers addition when A[7] = 1, updates Q[0] = 0
- Phase 1: c14 -> Updates Q[0] = 1
- Phase 2: c6 -> COUNT is updated
- Phase 3: c17 -> Triggers left shift

Cycle 9:

- Phase 0: c7 -> Load result A
- Phase 0: c9 -> Load result Q

———> Activate END signal