North South University

Department of Electrical & Computer Engineering

Project Report

Course ID & Name: CSE332 Computer Architecture & Organization

Section: 2

Semester: Spring 2025

Project Name: 18-Bit custom MIPS Architecture

Submitted to: Tnf

Report Submission Date: 12/4/25

Group Number: 12

Group Members ID & Name:

1. 2311657042	Saud
2. 2132239642	Farhan Faiyaz
3.	
4.	
5.	

Remarks:

Score

Introduction

In this project, we designed and built a custom 18-bit microprocessor from scratch. The processor supports a total of ten instructions, covering a combination of R-type (register), I-type (immediate), and one J-type (jump) instruction. Each instruction is executed within a single clock cycle, made possible by the integration of three key stages—Fetch, Decode, and Execute.

Our design is based on a custom Instruction Set Architecture (ISA), which was developed and submitted beforehand. Following the MIPS architecture, this microprocessor was built from the ground up to perform each instruction accurately, precisely, and efficiently according to the ISA. In the following sections, we will walk through the core subsystems that make up the microprocessor and explain the reasoning behind each design decision.

Register File

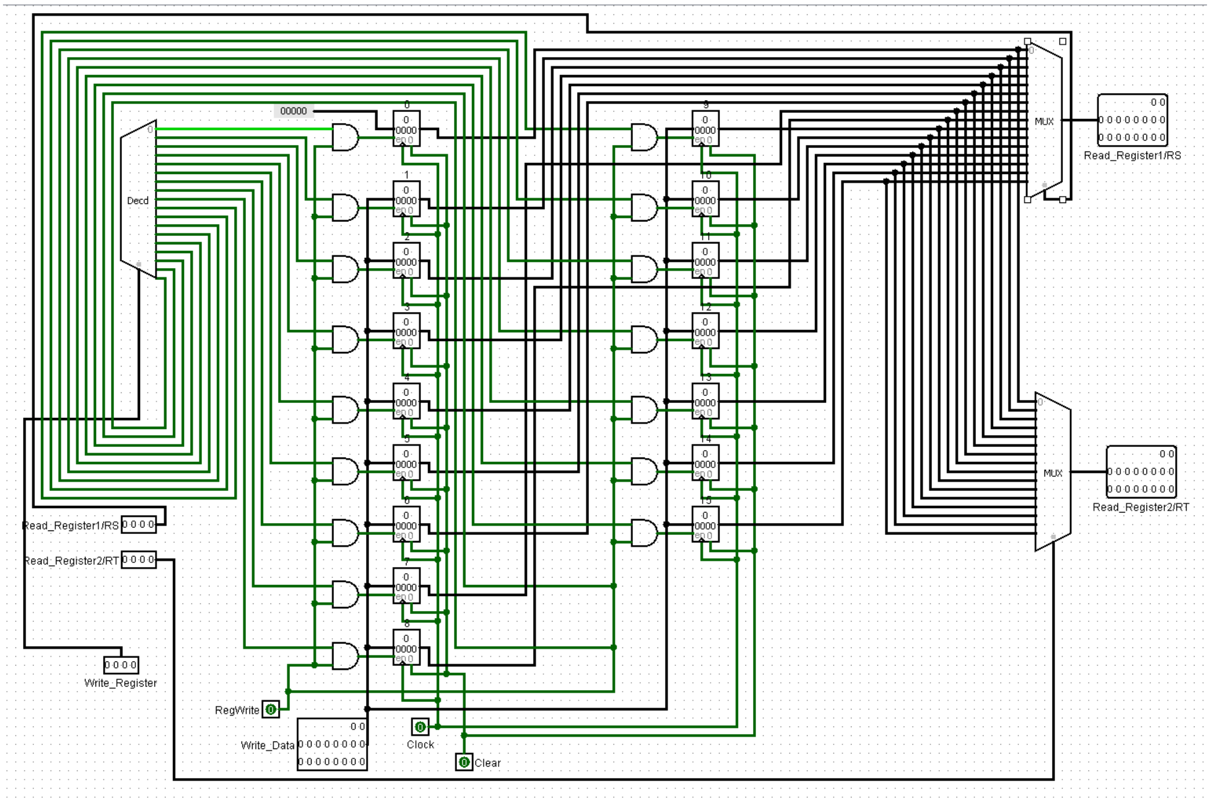
The register file serves as the internal data storage unit for the processor, playing a critical role in most operations executed by the Arithmetic Logic Unit (ALU). In a high-speed processor, fast and reliable access to register data is essential.

We implemented **16 general-purpose registers**, each capable of holding an 18-bit value. This number was chosen deliberately—it strikes a balance between hardware simplicity and sufficient storage capacity. Adding more registers would increase design complexity and resource usage, while fewer registers could limit flexibility in instruction execution.

The register file is driven by four main inputs:

- **ReadRegister1**: Specifies the first register to read from
- **ReadRegister2**: Specifies the second register to read from
- **WriteRegister**: Specifies the register where data should be written
- **WriteData**: Contains the 18-bit value to be stored in the WriteRegister

Additionally, a 4x16 **decoder** and two 1x16 **multiplexers** help manage register selection and data flow. The decoder enables one of the sixteen registers based on the write address, while the multiplexers select which registers to read. This configuration ensures streamlined data access and efficient execution. Below is a screenshot of the register file architecture:



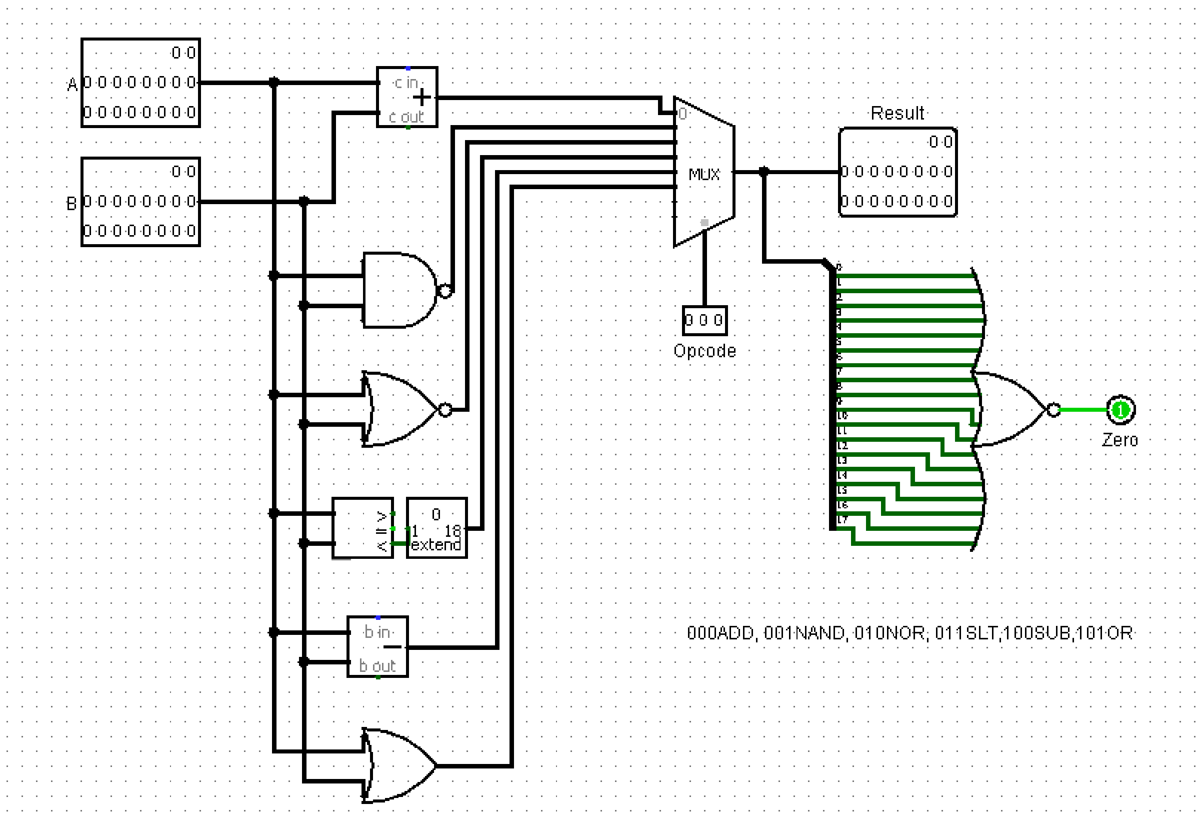
18-bit Arithmetic Logic Unit (ALU)

The ALU is the computational heart of the microprocessor. It is responsible for carrying out arithmetic and logic operations essential for executing instructions. In our design, the ALU supports six key operations, which are sufficient to cover the functionality of all ten instructions in our ISA.

The operations supported by the ALU include:

- **Addition**
- **NAND**
- **NOR**
- **Set-on-Less-Than (SLT)**
- **Subtraction**
- **OR**

This unit accepts two 18-bit inputs and produces an 18-bit result. Additionally, it outputs a **zero flag** used specifically for conditional branch instructions. To allow for selection between operations, a **1x8 multiplexer** is employed. The select lines for this multiplexer are derived from the **ALU Control Unit**, which interprets instruction type and function code to determine the correct operation. The diagram below illustrates the ALU configuration:



Memory Units

Our microprocessor follows the MIPS architecture, which separates instruction memory from data memory to allow simultaneous access and better performance.

- **Instruction Memory (ROM):**
  A Read-Only Memory (ROM) is used to store the processor's instructions. The 4-bit addressable width allows storage for 16 different instructions. This limited size was sufficient for our custom instruction set and allowed a smaller, more manageable control path. The instructions are hardcoded into the ROM as there is no need for writing, ensuring stability and predictability during execution.
- **Data Memory (RAM):**
  The RAM in our processor is **18 bits wide** to match the architecture's overall data width. Since the microprocessor operates on 18-bit instructions and data, the RAM must also handle 18-bit values during **load (LD)** and **store (SW)** operations. This ensures consistency across the datapath, allowing full 18-bit data to be stored to or retrieved from memory without truncation or extra processing. By aligning the RAM width with the processor's word size, we maintain smooth and efficient data handling throughout the system.

These memory units are clearly distinguished and integrated within the datapath structure of the processor.

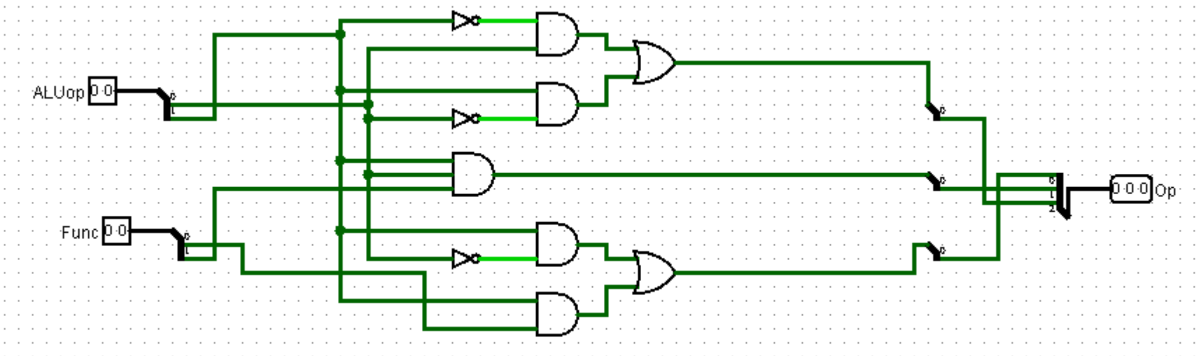
ALU Control Unit
The ALU Control Unit acts as a translator between instruction type and the ALU's internal operation selector. It takes as input:

- Two bits from the **Function Field**, which comes from the instruction itself
- Two bits from the **Main Control Unit**, which identifies the instruction category

These four bits are used to determine the exact operation the ALU should perform. Based on a defined truth table, a combinational logic circuit was constructed to generate the ALU's selection signals accurately.

Instruction	A1	A0	F1	F0	Q2	Q1	Q0
LW/SW	0	0	X	X	0	0	0
BEQ/SUBi	0	1	X	X	1	0	0
ORi	1	0	X	X	1	0	1
ADD	1	1	0	0	0	0	0
NAND	1	1	0	1	0	0	1
NOR	1	1	1	0	0	1	0
SLT	1	1	1	1	0	1	1

A combinational circuit is created from this truth table. Here is that image:



This setup ensures that the ALU only performs the intended operation for a given instruction, minimizing errors and maintaining the one-cycle execution goal.

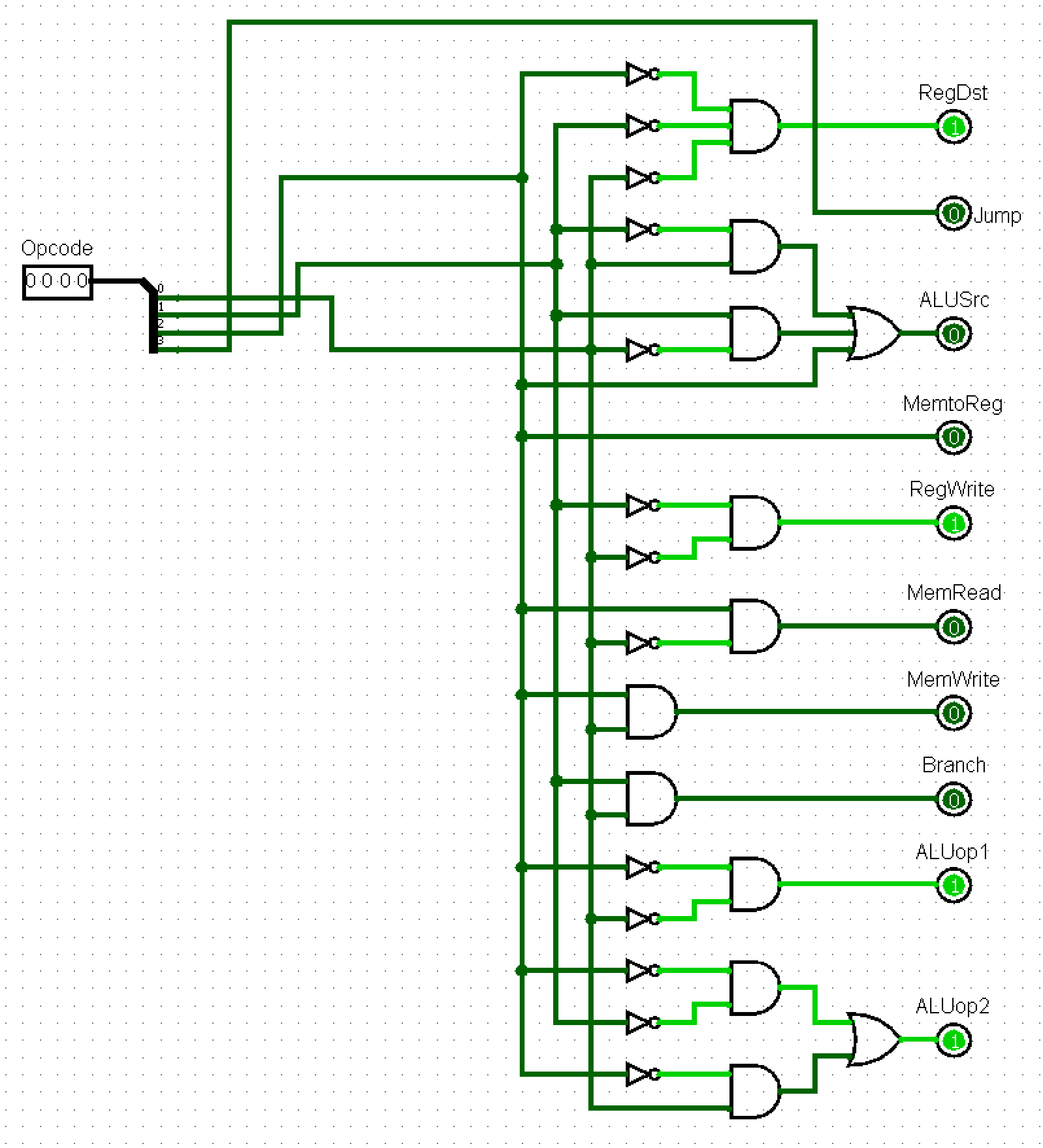
Main Control Unit
The Main Control Unit serves as the command center of the processor. It decodes the **opcode**, which consists of the four most significant bits of the 18-bit instruction, and generates the necessary control signals to guide the flow of data through the processor's subsystems.

Each instruction type—whether R-type, I-type, or J-type—triggers a unique set of control signals. These signals control multiplexers, memory read/write operations, register writes, ALU operations, and branching.

The control logic was constructed from a truth table, which maps each opcode to its corresponding control signal outputs.

Input	R-format	LW	SW	BEQ	SUBi	ORi
Op3	0	0	0	0	0	0
Op2	0	1	1	0	0	0
Op1	0	0	0	1	0	1
Op0	0	0	1	1	1	0
RegDst	1	0	X	X	1	1
ALUsrc	0	1	1	0	1	1
MemtoReg	0	1	X	X	X	X
RegWrite	1	1	0	0	1	1
MemRead	0	1	0	0	0	0
MemWrite	0	0	1	0	0	0
Branch	0	0	0	1	0	0
ALUop1	1	0	0	0	0	1
ALUop0	1	0	0	1	1	0

A combinational circuit is created from this truth table. Here is that image:



The Jump instruction is the only control output that needs the MSB of the opcode, i.e. no truth table was needed for it.

By modularizing the control logic in this way, it becomes easier to troubleshoot the processor in the future.

Conclusion

In conclusion, we successfully developed an 18-bit custom microprocessor from the ground up, closely following the MIPS architecture in both structure and operation. Every instruction—from arithmetic operations to data handling and conditional branching—was executed within a single clock cycle.

While the current implementation supports only ten instructions and a limited ROM capacity (16 instructions), the architecture is scalable. With additional memory and an expanded control unit, the processor could support more instructions and more complex programs in the future.

This project not only demonstrated a working CPU design but also reinforced our understanding of digital logic, hardware organization, and system integration. The microprocessor performed all tasks as intended, validating our instruction set and design choices.