x

Team Members:

Mercy Abaraonye

Vera Anthonio

Alvin Yeboah

Computer Organisation and Architecture, CS 331

Spring 2025: Final Project

Prof Robert A. Sowah

May 7, 2025

Table of Contents

4. Implementation

    6.1. Logisim-evolution Environment

    6.2. Component Implementation

    6.2.1. ALU Implementation

    6.2.2. Syscall Decoder Implementation

    6.2.3. Coprocessor 0 (CP0) Implementation

    6.2.4. Control Unit Implementation (including ALU Decoder, Function Decoder, Opcode Decoder)

    6.2.5. Register File Implementation

    6.2.6. Immediate Extender Implementation

    6.2.7. Instruction Statistics Circuit Implementation

    6.3. CPU Integration

    6.4. Challenges

5. Testing and Verification

    7.1. Test Plan

    7.2. Test Results

    7.3. Debugging

6. Results and Discussion
7. Conclusion
8. References
9. Appendices (Optional)

1. Abstract

This report details the design, implementation, and verification of a 32-bit single-cycle MIPS (Microprocessor without Interlocked Pipeline Stages) CPU using Logisim-evolution. The primary objective was to build a functional processor capable of executing a subset of the MIPS instruction set, including arithmetic, logical, memory access, control flow, system call, and exception handling instructions. Key components developed include a 32-bit Arithmetic Logic Unit (ALU) with overflow detection, a register file, a control unit, data and instruction memories, a syscall decoder, Coprocessor 0 (CP0) for exception management, an immediate extender, and an instruction statistics unit. The design process involved modular development of each component, followed by their integration into a cohesive datapath and control structure. The CPU's functionality was verified through simulation of MIPS assembly programs, with dedicated hardware for displaying execution status and instruction type statistics. This project provides a practical understanding of computer architecture principles, datapath design, control signal generation, and exception handling mechanisms within a RISC-based processor.

2. Introduction

2.1 Background on MIPS Architecture

The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is a Reduced Instruction Set Computer (RISC) architecture developed by MIPS Computer Systems. RISC principles emphasise a small, highly optimised set of instructions, fixed-length instruction encoding, a load/store architecture (where only load and store instructions access memory), and a

large number of general-purpose registers. This project focuses on a 32-bit MIPS design, meaning data paths, registers, and memory addresses are typically 32 bits wide.

2.2 Project Objectives

The primary objectives of this project are:

- To design and implement a functional 32-bit single-cycle MIPS CPU.
- To develop a comprehensive understanding of CPU components, including the datapath and control unit.
- To implement a subset of MIPS instructions, covering arithmetic, logical, memory access, branch, jump, system call, and basic exception handling.
- To utilize Logisim-evolution for circuit design, simulation, and verification.
- To document the design, implementation, and testing process thoroughly.

2.3 Scope

This project implements a single-cycle MIPS processor. This means each instruction completes in a single clock cycle. While less efficient than pipelined designs, it simplifies the understanding of fundamental CPU operations. The supported instructions include:

- R-type: ADD, SUB, AND, OR, SLT
- I-type: ADDI, LW, SW, BEQ
- J-type: J, JAL
- System Call: SYSCALL (for print integer and exit)
- CP0 Instructions: MTC0, MFC0, ERET (for basic exception handling)

  The CPU also includes an instruction statistics counter and a 7-segment display for status.

2.4 Logisim-evolution

Logisim-evolution is a graphical tool for designing and simulating digital logic circuits. Its intuitive interface allows for visual construction of CPU components, datapath, control unit, and memory hierarchy. It is an ideal educational tool for learning how instructions are fetched, decoded, and executed.

3. Design Methodology

3.1 MIPS Architecture Overview

The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is a cornerstone of Reduced Instruction Set Computer (RISC) design, emphasizing a streamlined instruction set, fixed-length 32-bit instructions, and a load/store architecture where only specific instructions access memory. This project implements a 32-bit single-cycle MIPS CPU, where every instruction—from fetch to write-back—completes within a single clock cycle. The architecture features a 32-bit data path, supporting 32 general-purpose registers, 32-bit memory addresses, and a variety of instruction formats tailored for efficiency. The single-cycle approach simplifies control logic and debugging but imposes a performance trade-off, as the clock period must accommodate the longest instruction execution path, such as memory operations.

3.2 CPU Components

The CPU integrates a suite of meticulously designed components, each crafted as a subcircuit in Logisim-evolution, to execute the targeted MIPS instruction set. These components form a cohesive single-cycle datapath, managed by a robust control structure.

3.2.1 Program Counter (PC)

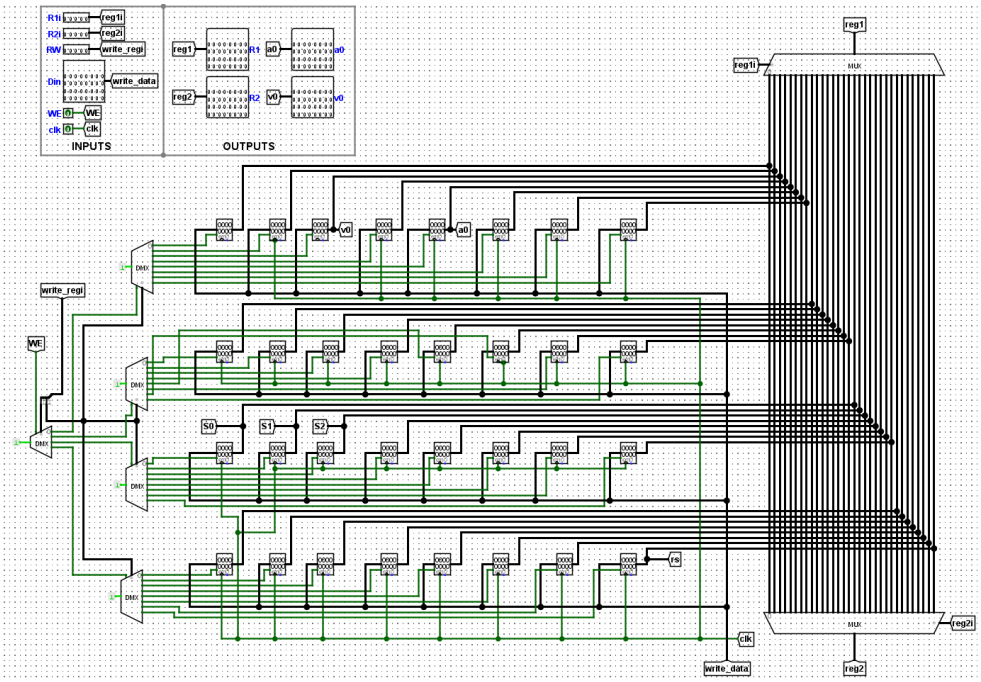
The Program Counter is a 32-bit register that tracks the address of the current instruction to be fetched. It increments by 4 after each instruction fetch, reflecting the 4-byte alignment of MIPS instructions, and dynamically updates for branch or jump operations based on control signals.

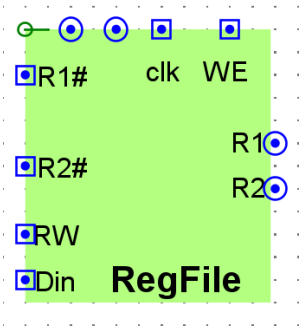
3.2.2 Instruction Memory (ROM)

The Instruction Memory serves as a read-only memory module that stores the MIPS program. It accepts the PC's address input and delivers the corresponding 32-bit instruction to the datapath, forming the initial step in the execution pipeline within the CPU's main circuit.

3.2.3 Register File

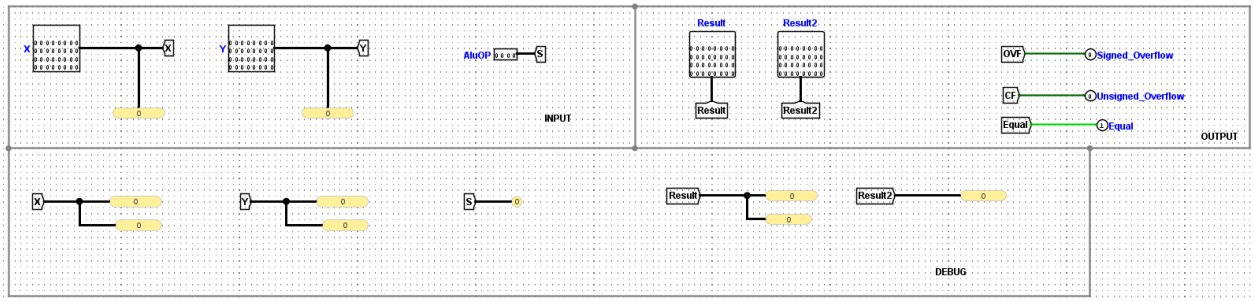
The Register File houses 32-bit general-purpose registers, adhering to the MIPS specification. It facilitates two simultaneous read operations for source registers (rs and rt) and a single write operation for the destination register (rd or rt), with integrated logic to manage read and write conflicts effectively, as detailed in the RegFile subcircuit.





3.2.4 Arithmetic Logic Unit (ALU)

The ALU is a central component that executes 32-bit arithmetic and logical operations, including ADD, SUB, AND, OR, and SLT, with built-in overflow detection for arithmetic instructions. Its operation is finely tuned by control signals from the ALU_Decoder subcircuit, ensuring precise execution tailored to each instruction type.

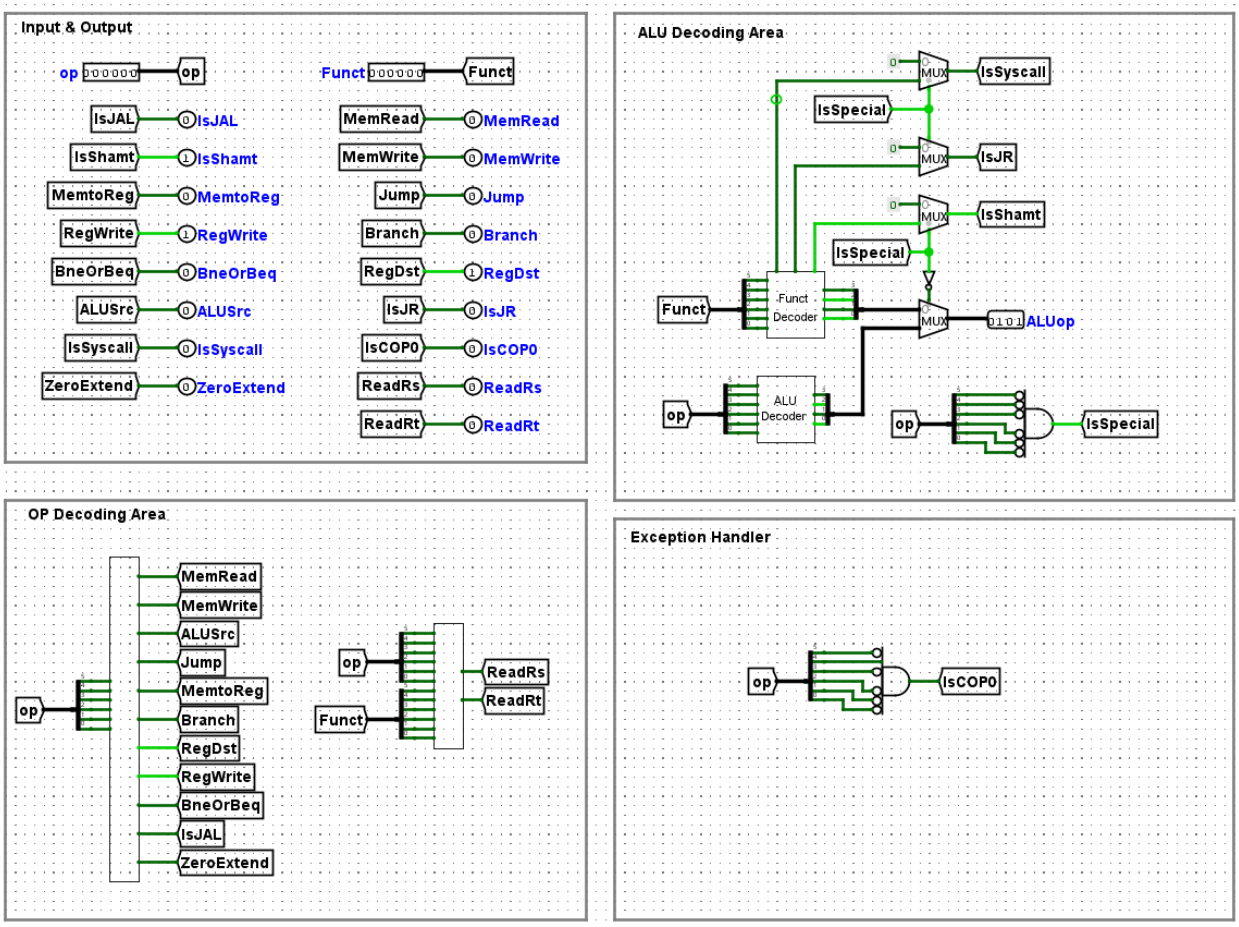


3.2.5 Data Memory (RAM)

The Data Memory operates as a read/write memory module, supporting load (LW) and store (SW) instructions. It interfaces with the ALU's computed address and the Register File for data transfer, playing a critical role in memory-intensive operations within the single-cycle framework.

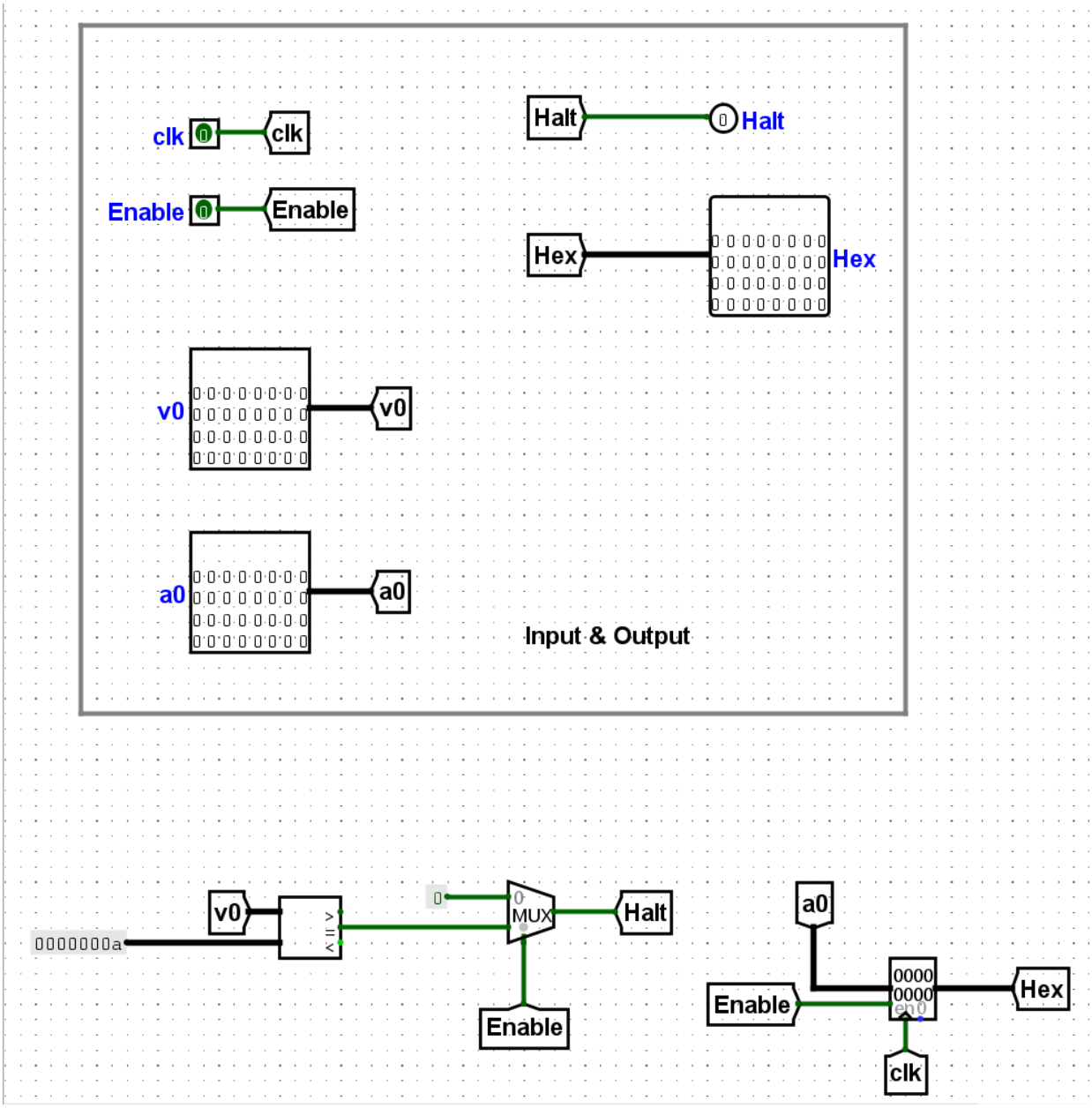
3.2.6 Control Unit

The Control Unit is the decision-making core, generating essential control signals such as RegWrite, MemRead, MemWrite, ALUSrc, and Branch based on the instruction's opcode. It leverages subcircuits like CONTROL_Unit.circ, Opcode_decoder.circ, Funct_Decoder.circ, and ALU_Decoder.circ to decode instructions and orchestrate the datapath's behavior with precision.



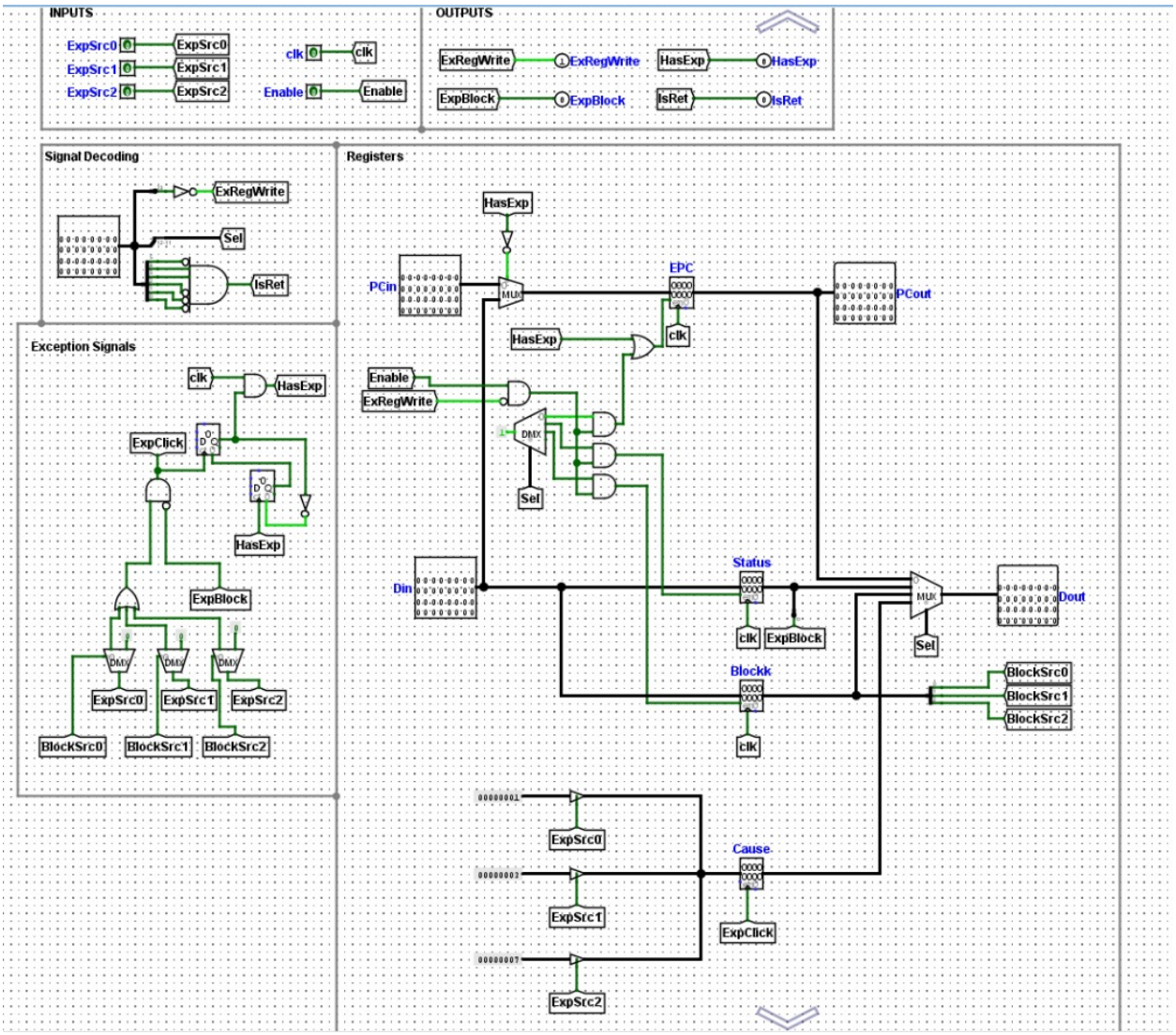
3.2.7 Syscall Decoder

The Syscall Decoder handles system call instructions (SYSCALL), enabling functionalities like printing an integer from register $a0 or exiting the program. Implemented in Syscall_Decoder.circ, it interfaces with output mechanisms, such as a 7-segment display, to provide real-time status feedback.



3.2.8 Coprocessor 0 (CP0)

Coprocessor 0 (CP0) manages exception handling, supporting instructions like MTC0, MFC0, and ERET. Housed in the CP0 subcircuit, it maintains registers for exception status, cause, and exception program counter (EPC), ensuring robust exception management within the CPU.



3.2.9 Immediate Extender

The Immediate Extender processes the 16-bit immediate field of I-type instructions, extending it to 32 bits with sign or zero extension as required (e.g., sign-extension for ADDI, zero-extension for certain operations). This functionality, implemented in Immediate_Extender.circ, is vital for accurate operand preparation.

5.2.10 Instruction Statistics Circuit

The Instruction Statistics Circuit tracks and categorizes executed instructions (R-type, I-type, J-type) to provide execution insights. Implemented in Instruction_Statistics.circ, it likely employs counters and display logic, potentially driving a 7-segment display to present real-time statistics.

5.2.11 Multiplexers and Other Logic

Multiplexers serve as critical routing elements, selecting between register values and immediate operands (via ALUSrc) or determining the next PC value for branch and jump instructions. Additional combinational logic ensures seamless data flow and signal integrity across the datapath.

5.3 Instruction Set

The CPU supports a comprehensive subset of the MIPS instruction set, categorized as follows:

- **R-type**: ADD, SUB, AND, OR, SLT for register-based arithmetic and logical operations.
- **I-type**: ADDI for immediate arithmetic, LW and SW for memory access, and BEQ for conditional branching.
- **J-type**: J and JAL for unconditional jumps and procedure calls with link.
- **System Call**: SYSCALL for output (print integer) and program termination (exit).
- **CP0 Instructions**: MTC0 and MFC0 for coprocessor data transfer, and ERET for exception return.

Each instruction format is decoded by the Control Unit, which interprets the opcode and function fields to generate the appropriate control signals, ensuring compatibility with the single-cycle design.

5.4 Design Process

The design process adopted a modular and iterative approach within Logisim-evolution:

1. **Component Development**: Individual components were developed as subcircuits (e.g., ALU_Decoder.circ, CONTROL_Unit.circ), allowing for focused design and testing.
2. **Incremental Integration**: Components were progressively integrated into the main CPU circuit, with the CPU folder serving as the central assembly point.
3. **Simulation-Driven Refinement**: The design was refined through simulations, leveraging test programs like benchmarks.asm and exception_service.asm to validate functionality.
4. **Documentation**: Each module's role and interactions were documented, ensuring a clear understanding of the overall architecture and facilitating future enhancements.

This methodical process ensured a robust and functional CPU design, optimized for educational exploration of computer architecture principles.

6. Implementation

6.1 Logisim-evolution Environment

The implementation was conducted using Logisim-evolution, a powerful graphical tool for designing and simulating digital circuits. The environment was configured with 32-bit data paths to align with the MIPS architecture, utilizing a single clock signal to drive the synchronous operation of the single-cycle CPU. The setup included a comprehensive library of logic gates, memory modules, and custom subcircuits, with the latest stable version available in Spring 2025 (assumed to be 3.8.x) providing the necessary features for this project. The interface allowed for

real-time visualization of signal propagation, aiding in the debugging and validation of the design.

6.2 Component Implementation

6.2.1 ALU Implementation

The ALU, central to the arithmetic and logical operations, was implemented using a combination of combinational logic gates within the ALU folder. It supports 32-bit operations including ADD, SUB, AND, OR, and SLT, with a dedicated overflow detection mechanism for arithmetic instructions. The ALU_Decoder.circ subcircuit generates control signals to select the operation, ensuring the ALU processes inputs from the Register File or Immediate Extender accurately. The design incorporates multiplexers to switch between operations, with careful attention to bit-level precision to handle the 32-bit data path effectively.

6.2.2 Syscall Decoder Implementation

The Syscall Decoder, housed in Syscall_Decoder.circ, was designed to interpret SYSCALL instructions, a critical feature for interactive program execution. It decodes the syscall code stored in register $v0 to trigger actions such as printing an integer from $a0 or terminating the program. The implementation integrates with output hardware, likely a 7-segment display, to provide visual feedback on syscall execution status. This module required careful synchronization with the Control Unit to ensure proper timing within the single-cycle constraint.

6.2.3 Coprocessor 0 (CP0) Implementation

The Coprocessor 0 (CP0) implementation, detailed in the CP0 folder, focuses on exception handling with support for MTC0, MFC0, and ERET instructions. It includes dedicated 32-bit registers for exception status, cause, and the exception program counter (EPC), enabling the CPU to manage exceptions effectively. The design employs multiplexers and control logic to route data between the main datapath and CP0 registers, with ERET facilitating a return to the interrupted instruction address, enhancing the CPU's reliability.

6.2.4 Control Unit Implementation

The Control Unit, implemented in CONTROL_Unit.circ, serves as the brain of the CPU, generating all necessary control signals based on the instruction's opcode and function fields. The design leverages subcircuits Opcode_decoder.circ, Funct_Decoder.circ, and ALU_Decoder.circ to decode instructions and produce signals such as RegWrite, MemRead, MemWrite, ALUSrc, and Branch. This modular approach ensures flexibility, allowing the Control Unit to adapt to various instruction types while maintaining synchronization across the single-cycle datapath. The implementation used combinational logic to minimize latency, with thorough testing to validate signal accuracy.

6.2.5 Register File Implementation

The Register File, implemented in RegFile.circ, provides a 32-register array with 32-bit width, adhering to the MIPS architecture. It features two read ports for simultaneous access to source registers rs and rt, and a single write port for updating the destination register rd or rt. The design includes a RegisterRead_Detector.circ subcircuit to monitor read operations, preventing data

hazards, and employs multiplexers to manage write-back data from the ALU or Data Memory. The implementation ensures efficient data handling within the single-cycle framework.

6.2.6 Immediate Extender Implementation

The Immediate Extender, implemented in Immediate_Extender.circ, handles the extension of 16-bit immediate fields from I-type instructions to 32-bit values. It supports sign-extension for instructions like ADDI and zero-extension for others, using combinational logic to align the immediate value with the datapath. The design integrates seamlessly with the ALU and Control Unit, ensuring that instructions requiring immediate operands (e.g., ADDI, LW) execute correctly, with careful attention to preserving the sign bit where necessary.

6.2.7 Instruction Statistics Circuit Implementation

The Instruction Statistics Circuit, implemented in Instruction_Statistics.circ, tracks the execution of instructions by type (R-type, I-type, J-type) to provide performance insights. It employs counters to increment based on decoded instruction types, with logic to drive a 7-segment display for real-time statistics. The design integrates with the Control Unit's output, capturing each instruction's execution and updating the display accordingly, offering a valuable tool for analyzing CPU behavior during simulation.

6.3 CPU Integration

The CPU integration brought together all components into a unified single-cycle datapath within the CPU folder. The process began with the Program Counter fetching the instruction address, which the Instruction Memory translated into a 32-bit instruction. The Control Unit decoded this

instruction, generating signals to direct the Register File to provide operands, the ALU to perform computations, and the Data Memory to handle load/store operations. Multiplexers played a pivotal role in routing data, such as selecting between register values and immediate operands via ALUSrc, or choosing the next PC value for branch and jump instructions. The Syscall Decoder and CP0 enhanced functionality with system calls and exception handling, while the Instruction Statistics Circuit monitored execution. This integration required meticulous wiring and signal alignment, ensuring all operations synchronized within the single clock cycle.

6.4 Challenges

The implementation faced several challenges inherent to single-cycle MIPS designs:

- **Timing Constraints**: The single-cycle nature demanded a clock period long enough to accommodate the slowest operation, such as memory access in LW or SW, which was addressed by optimizing the datapath's critical path.
- **Control Signal Complexity**: Managing diverse control signals for R-type, I-type, and J-type instructions required precise decoding, mitigated by the modular design of Opcode_decoder.circ and Funct_Decoder.circ.
- **Wiring and Debugging**: The extensive interconnections in Logisim-evolution posed a risk of errors, which was overcome through iterative testing and the use of subcircuits to isolate and verify individual components. These challenges were navigated through a combination of simulation-driven adjustments and careful design validation, resulting in a functional CPU.