Computer Architecture
Assignment 07

—

Ritik Tiwari

B21CS098

MIPS Control Unit Design and Analysis Report

1. Introduction

- Assignment Overview

  The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture represents a fundamental RISC (Reduced Instruction Set Computing) design that has significantly influenced modern processor architecture. This assignment focuses on implementing a MIPS control unit using Logisim, specifically addressing:

  1) Design and simulation of a basic MIPS control unit
  2) Generation and management of critical control signals
  3) Analysis of fault conditions through MSB modifications
  4) Verification of control unit behavior across different instruction types

- Objectives

  The primary objectives of this implementation include:

  1) Development of a functional MIPS control unit capable of:
     a) Processing 32-bit MIPS instructions
     b) Generating appropriate control signals
     c) Determining ALU operations when required
     d) Handling R-type, I-type, and J-type instructions

  2) Analysis of control unit behavior under various conditions:

     a) Normal operation scenarios
     b) Modified MSB conditions
     c) Fault detection and impact assessment

- Technical Context

  Control Unit Rule

  The control unit serves as the instruction decoder and command center of the processor, responsible for:

  1) Interpreting incoming instructions
  2) Generating control signals that orchestrate data movement
  3) Coordinating ALU operations
  4) Managing memory access operations
  5) Ensuring proper instruction execution sequence

- Scopes and Limitations

  This implementation encompasses:

  1) Basic MIPS instruction set support
  2) Core control signal generation
  3) ALU operation determination
  4) MSB modification analysis

2. Design Implementation

- Circuit Components

  Description of main circuit blocks:
  1) **Instruction decoder:** The instruction decoder plays a key role in translating the binary machine instructions into actionable commands that the CPU can execute, forming a bridge between program code and actual hardware operations.
  2) **Control signal generator:** A control signal generator is a component in a CPU that produces specific control signals based on the decoded instruction to direct the operation of various hardware units, such as the ALU, registers, and memory. These

signals ensure that each unit performs the correct action at the right time during instruction execution.

3) **ALU operation checker:** It checks whether what alu operation to be performed.

4) **Input/Output interface:**This makes the what input instruction to be provide in zeros and ones to get the output on the output section.

- Control Signals Implementation

Here are all the control Signal which is required in this assignment.

1) **RegDst**: Determines the destination register for writing data; selects whether the destination register is specified in the instruction field (rt) or (rd).

2) **ALUSrc**: Chooses the second operand for the ALU; selects between a register value or an immediate value from the instruction.

3) **MemToReg**: Controls whether the data written to the register file comes from memory (data load) or the ALU's output.

4) **RegWrite**: Enables writing to the register file, allowing the specified register to be updated with new data.

5) **MemRead**: Activates reading from memory, used during load instructions to fetch data from a specified memory address.

6) **MemWrite**: Enables writing to memory, used during store instructions to update the content at a specified memory address.

7) **Branch**: Indicates if the instruction is a branch type; used to determine if the program counter should be updated to a target address.

8) **ALUOp** (2 bits): Specifies the operation that the ALU should perform (e.g., addition, subtraction, AND, OR); used to select between arithmetic or logic operations.

9) **ALU Required**: Indicates whether the ALU should be used for the current instruction, determining if an arithmetic or logic operation is necessary.

- Instruction Handling
    1) Different Instrcution involved in the MIPS Processor.
        a) R-type instruction processing
            1) **Definition**: R-type (Register-type) instructions are used for arithmetic and logical operations, where both operands and the destination are specified in the register fields.
            2) **Processing**: The instruction includes fields for the source registers (`rs` and `rt`), the destination register (`rd`), the shift amount (`shamt`), and the function code (`funct`). The ALU performs the operation defined by the function code using the values from the source registers, and the result is stored in the destination register.

        b) I-type instruction processing
            1) **Definition**: I-type (Immediate-type) instructions are used for operations involving immediate values, memory access (load/store), and branching.
            2) **Processing**: The instruction includes fields for one source register (`rs`), a destination or target register (`rt`), and a 16-bit immediate value. The ALU is used for computing addresses (in load/store instructions) or performing arithmetic/logic operations with the immediate value. For branching, it calculates the target address.

        c) J-type instruction processing
            1) **Definition**: J-type (Jump-type) instructions are used for unconditional jump operations.
            2) **Processing**: The instruction contains a 26-bit address field that specifies the target address for the jump. The program

counter (PC) is updated to this new address, allowing for direct control of program flow.

2) ALU Operation Determination Logic:

The ALU operation determination logic uses the `ALUOp` and `funct` fields to decide which specific operation the ALU should perform. For R-type instructions, the function code (`funct`) specifies the operation, while for I-type instructions, the `ALUOp` field indicates operations such as addition (for load/store) or subtraction (for branch comparisons). The logic ensures that the appropriate arithmetic or logic operation is executed based on the instruction type and its parameters.

3. Testing and Verification

a) Test Cases

Minimum 3 test cases for each instruction type:

1) **R-type instructions**
   -> add $t2, $t0, $t1 (Binary Output: 00000001000010010101000000100000)

<div align="center">

32 bit ALU Output

0000 0000 0000 0000 0000 0000 0000 0000

</div>

-> and $t4, $t0, $t1 (Binary Ouput:
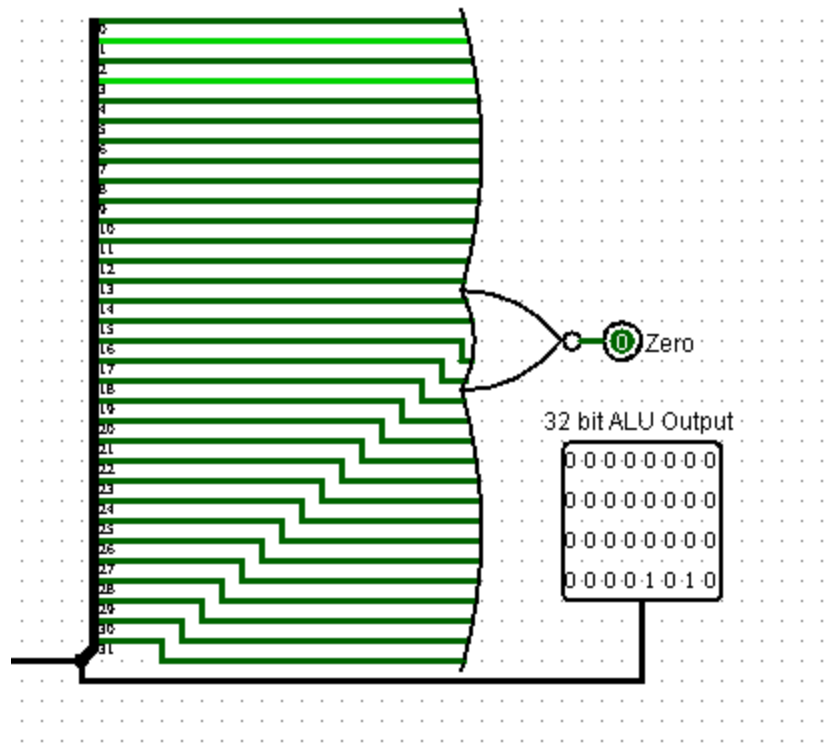00000001000010010110000000100100)

32 bit ALU Output
0000 0000 0000 0000 0000 0000 0000 0000

-> slt $t6, $t0, $t1 (Binary Output:
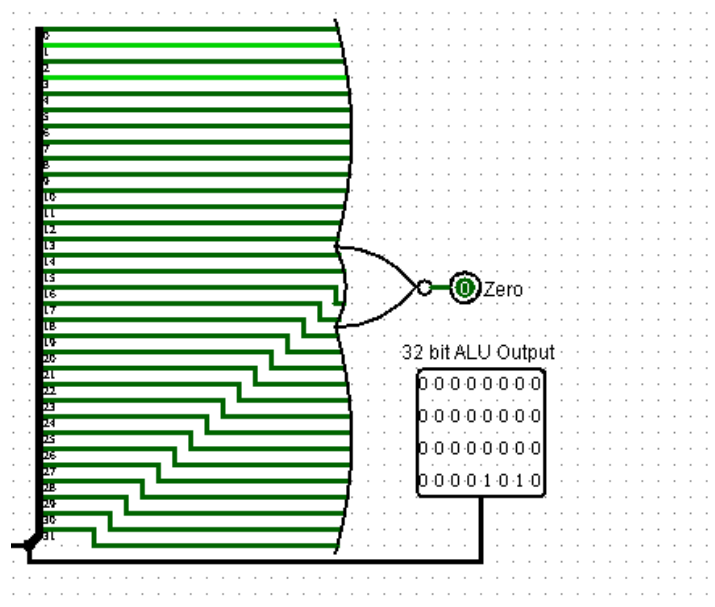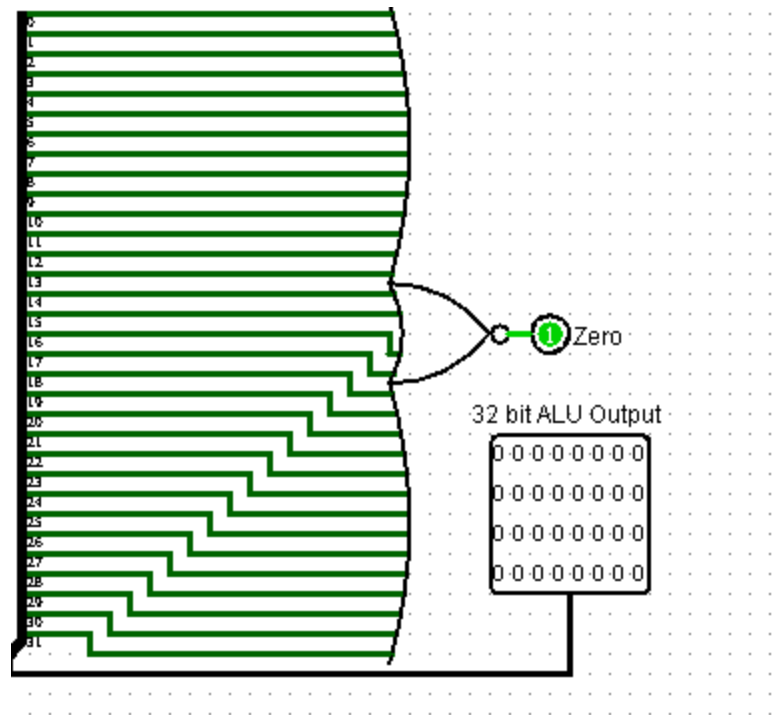00000001000010010111000000101010)
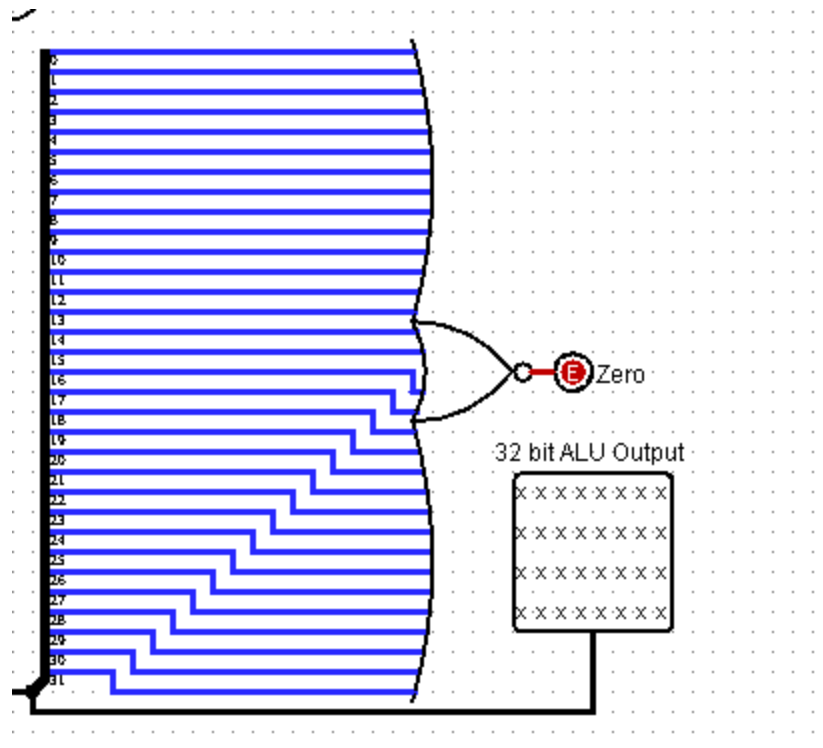
2) I-type instructions

-> lw $t0, num1 (Binary Ouput: 10001100000010000000000000001010)

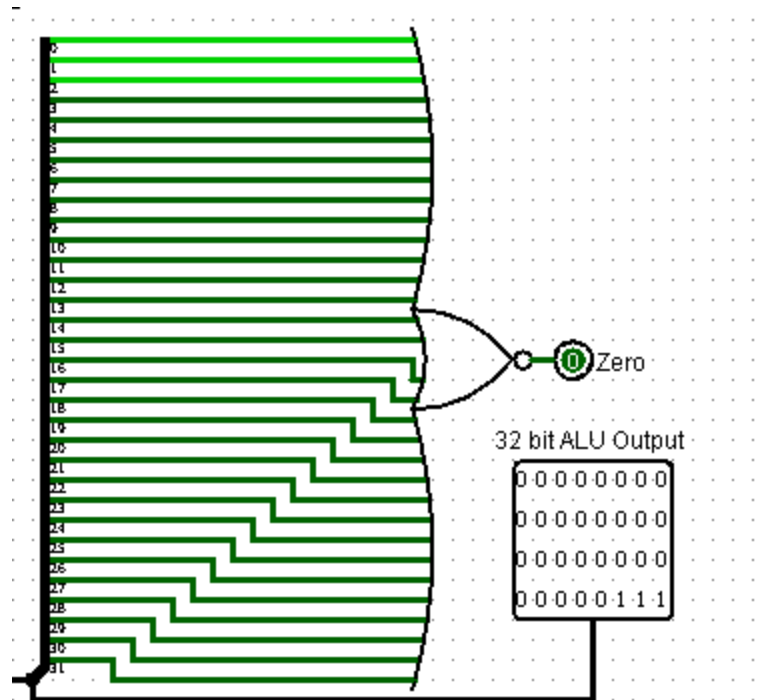-> addi $t2, $t0, 10 (Binary Output: 00100001000010100000000000001010)

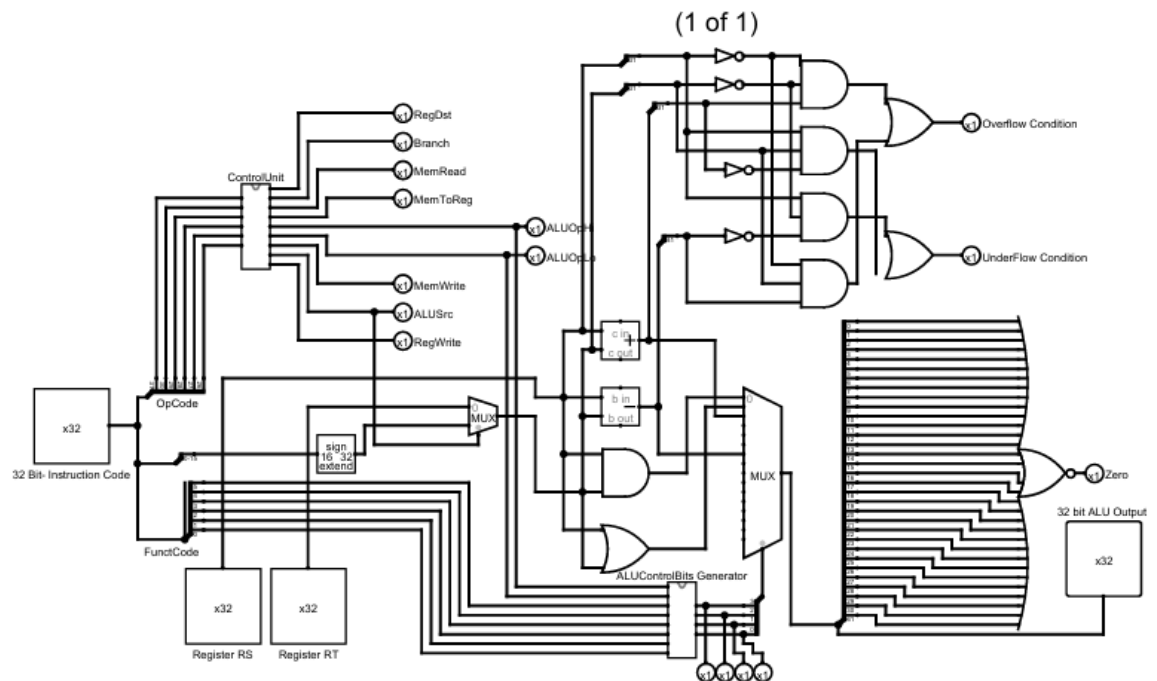-> beq $t0, $t1, equal_case (Binary Output:
00010001001010000000000000000010)

3) **J-type instructions**

-> j end (Binary Ouput: 00001000000000000000000000000111)

b) Screenshots of circuit operation

5. Conclusion

a) Summary of achievements

1) Successfully implemented a functional MIPS control unit capable of processing R-type, I-type, and J-type instructions using Logisim, demonstrating practical understanding of processor architecture.
2) Developed an efficient control signal generation system that accurately produces nine distinct control signals (RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, and ALUOp) based on instruction type.
3) Implemented an organized circuit design using subcircuits and proper labeling, making the system modular and maintainable.
4) Successfully integrated fault analysis capabilities into the design, allowing for comprehensive testing of different failure scenarios.

b) Key findings from MSB modification analysis

1) R-type Instructions Resilience:
   a) Demonstrated high resilience to first MSB modifications.
   b) Showed vulnerability to second MSB changes.
   c) Completely failed under combined MSB modifications.
   d) Most stable under stuck-at-0 conditions.
2) Control Signal Dependencies:
   a) ALUOp signals highly dependent on MSB integrity.
   b) Memory control signals showed predictable failure patterns.
   c) Branch control signals demonstrated high sensitivity to modifications.
   d) Register selection signals showed variable reliability.
3) Program Flow Impact:
   a) Jump instructions critically affected by second MSB modifications.
   b) Branch operations showed inconsistent behavior under fault conditions.
   c) Program counter updates failed in specific stuck-at combinations.
   d) Control flow integrity compromised in most fault scenarios.

c) Learning outcomes

1) Practical Architecture Understanding
   a) Gained deep insight into MIPS architecture functionality.
   b) Developed practical knowledge of control unit operations.
   c) Understanding of instruction decode mechanisms.
   d) Appreciation of hardware-level instruction processing.
2) Digital Design Skills
   a) Enhanced proficiency in Logisim tool usage.
   b) Improved digital circuit design capabilities.
   c) Developed debugging and testing strategies.
   d) Learned effective circuit organization techniques.
3) System Design Insights
   a) Understanding of control signal dependencies.
   b) Knowledge of failure mode analysis.

c) Appreciation of system reliability factors.
d) Insight into architectural trade-offs.