CS224: Assignment 8 Report
Single-Cycle MIPS Processor Implementation

Group 30
Aaryan Bondekar, 230101001
Aditya Anand 230101005
Dedeepya Siri, 230101011
Bommagani Rohitha, 230101027

April 22, 2025

Single-Cycle MIPS Processor Implementation

1 Introduction

This report documents our design and implementation of a single-cycle MIPS processor in Verilog as specified in Assignment 8. The implementation supports a subset of the core MIPS instruction set, allowing for the execution of meaningful programs. We've implemented the processor at the gate and RTL level, avoiding behavioral implementations where possible. Our processor is modular, with each component implemented separately and connected to form the complete datapath.

The supported instruction set includes:

1. Memory reference instructions: load word (lw) and store word (sw)
2. Arithmetic-logical instructions: add, sub, and, or, and set-less-than (slt)
3. Control transfer instructions: branch equal (beq) and jump (j)
4. Subroutine support: jump and link (jal)

The design follows the architecture outlined in Patterson & Hennessy's "Computer Organization & Design: The HW/SW interface."

2 Architecture Overview

The single-cycle MIPS processor executes each instruction in one clock cycle, with the following main components:

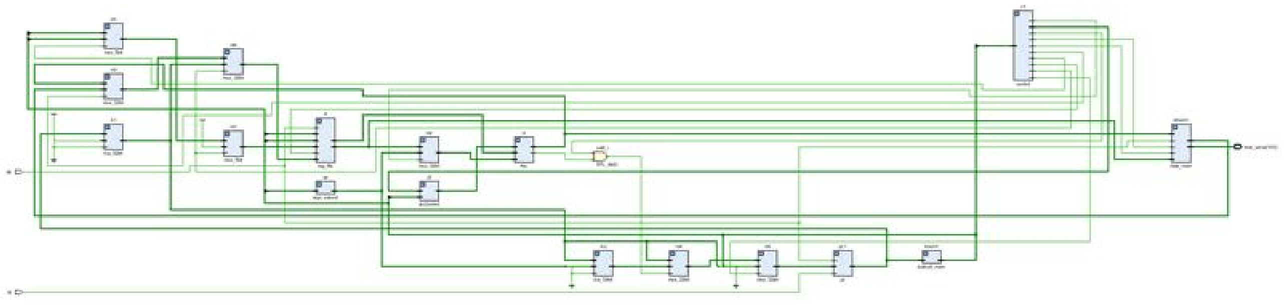

Figure 1: Single-Cycle MIPS Processor Architecture

1. **Program Counter (PC)**: Holds the address of the current instruction.
2. **Instruction Memory**: Stores the program instructions.
3. **Register File**: Contains 32 general-purpose registers.
4. **ALU**: Performs arithmetic and logical operations.
5. **Data Memory**: Stores data for load/store operations.
6. **Control Unit**: Generates control signals based on the instruction opcode.
7. **ALU Control**: Determines ALU operation based on the function code.
8. **Sign Extender**: Extends immediate values from 16 to 32 bits.
9. **Shift Units**: For branch and jump address calculations.
10. **Multiplexers**: For selecting between different paths in the datapath.

3 Module Implementations

1. **Program Counter (PC)**: The program counter is implemented as a 32-bit register that updates on each clock cycle. It stores the address of the current instruction and increments by 4 for the next instruction, unless a branch or jump instruction changes the flow of execution.
2. **Multiplexer (MUX)**: The 2:1 multiplexer is implemented using gate-level logic to select between two 32-bit inputs based on a selection signal.
3. **3:1 RegDst Multiplexer**: A specialized multiplexer that selects between rt, rd, or register \$31 (for jal instruction) as the destination register.

4. **Sign Extender**: The sign extender converts 16-bit immediate values to 32-bit values by extending the sign bit.
5. **Shift Left 2**: This module shifts its input left by 2 bits, effectively multiplying by 4, for branch and jump address calculations.
6. **Control Unit**: The control unit decodes the instruction opcode and generates the appropriate control signals for the datapath components.
7. **ALU Control**: The ALU control unit determines the ALU operation based on the ALU operation code from the control unit and the function code for R-type instructions.
8. **Register File**: The register file contains 32 registers, each 32 bits wide. It supports simultaneous reading of two registers and writing to one register.
9. **Instruction Memory**: The instruction memory is implemented as a read-only memory that stores the program instructions. Each instruction is 32 bits wide.
10. **Data Memory**: The data memory is implemented as a read-write memory that stores data for load and store operations.

4 Gate-Level Implementation Details

1. **Full Adder**: The full adder is implemented using primitive gates (AND, OR, XOR) to add three bits (two inputs and one carry-in) and produce a sum and carry-out.

5 Single-Cycle Datapath

The single-cycle datapath integrates all the components described above to execute MIPS instructions. Each instruction follows these general steps:

1. Fetch: The instruction is fetched from instruction memory using the address in the PC.
2. Decode: The instruction is decoded to determine the operation and operands.
3. Execute: The ALU performs the required operation.
4. Memory Access: For load/store instructions, data memory is accessed.
5. Write Back: For instructions that write to a register, the result is written back.

6 Instruction Execution Details

1. **R-type Instructions (add, sub, and, or, slt)**
   (a) Register values are read from the register file based on rs and rt fields.
   (b) The ALU performs the operation specified by the function code.

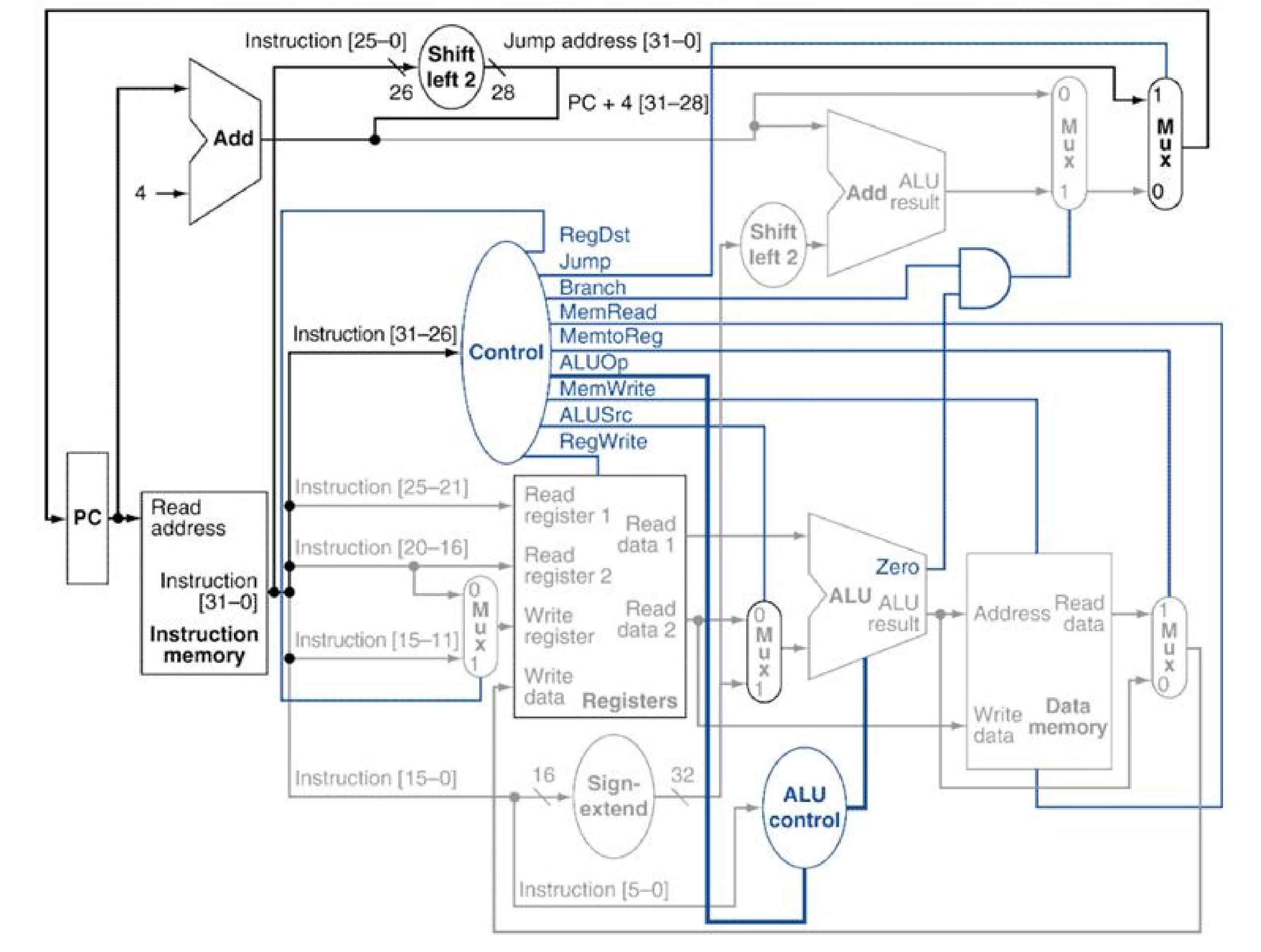

Figure 2: Complete Single-Cycle MIPS Datapath

   (c) The result is written back to the register specified by the rd field.

2. **Load Word (lw)**
   (a) Base register (rs) and offset (immediate) are used to calculate the memory address.
   (b) Data is read from memory at the calculated address.
   (c) The data is written back to the register specified by the rt field.

3. **Store Word (sw)**
   (a) Base register (rs) and offset (immediate) are used to calculate the memory address.
   (b) Data from the register specified by the rt field is written to memory at the calculated address.

4. **Branch Equal (beq)**
   (a) Registers specified by rs and rt are compared.
   (b) If equal, the PC is updated to branch target address (PC + 4 + immediate ¡¡ 2).

5. **Jump (j)**

   (a) The PC is updated to the jump target address.

6. **Jump and Link (jal)**
   (a) The return address (PC + 4) is saved in the register $31.
   (b) The PC is updated to the jump target address.

7 Simulation Results

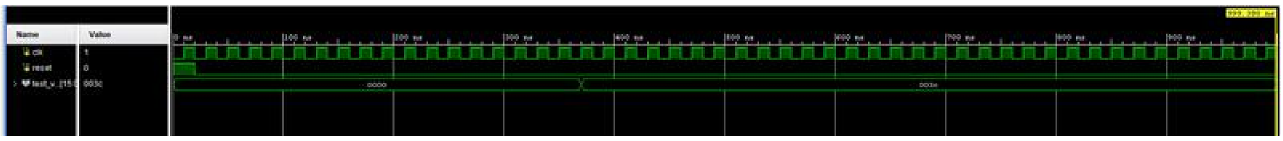

Figure 3: Simulation Waveform of Single-Cycle MIPS Processor

Our simulation tests verified that:

1. The processor correctly fetches and executes all supported instructions.
2. Register file operations (read and write) work as expected.
3. Memory operations (load and store) correctly access data memory.
4. Branch and jump instructions properly update the program counter.
5. ALU operations produce the correct results.

8 Conclusion

We successfully implemented a single-cycle MIPS processor supporting a subset of the core MIPS instruction set. Our implementation is modular, with each component implemented at the gate or RTL level. The processor correctly executes a test program that demonstrates all supported instructions.

The single-cycle design provides a straightforward implementation but has performance limitations compared to more advanced architectures. However, it serves as an excellent foundation for understanding processor design and can be extended to more complex implementations.

This project provided valuable insights into computer architecture, digital design, and Verilog programming. It reinforced concepts from the course and demonstrated the practical application of theoretical knowledge.