Hardware Lab (CS 224)
Assignment 8

Group No. 31

Team Members:

Shrey Chandra(230101097)
Sunil (230101099)
Subham (230101098)
Jagrati Mittal(230101047)

April 22, 2025

Contents

1 Problem Statement **2**

2 Logic Behind The Verilog implementation **3**
2.1 Introduction 3
2.2 RISC Architecture 3
2.3 Instruction Execution Cycle 3
2.4 ALU Operations 3
2.5 Register File Usage 4
2.6 Hierarchical Debugging 4

3 Process to Obtain the Design **4**
3.1 Testbench Framework 4
3.2 MIPS R-format instructions 5
3.3 MIPS I-format instructions 5
3.4 MIPS J-format instructions 5

4 Design Methodology **5**

5 Conclusion **7**

1 Problem Statement

The objective of our assignment is to design a single cycle 32-bit MIPS implementation that includes a subset of the core MIPS instruction set

- The memory reference instructions: load word (lw) and store words (sw)
- The arithmetic-logical instructions: add, sub, and, or, and set-less-than (slt)
- Control transfer instructions: branch equal (beq) and jump (j)
- Instruction for supporting subroutine: jump and link (jal)

we need to design the data-path (Instruction memory (only read access), Data memory (read and write), Register file (32 numbers of 32 bit registers), ALUs, MUXes, Shifter, and Sign Extender, Program counter etc), and controller of the processor (executes one instruction in a single cycle)

2 Logic Behind The Verilog implementation

2.1 Introduction

MIPS (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computing (RISC) architecture. This testbench simulates a single-cycle MIPS processor, monitors control and data path signals, and displays register values during execution. The purpose is to ensure the processor correctly executes a sequence of instructions and stores Fibonacci numbers in memory.

2.2 RISC Architecture

MIPS follows a RISC design, which features a small, highly optimized set of instructions for high performance.Fibonacci Algorithm in Assembly:

Registers are used to store intermediate values.

The loop continues computing Fibonacci numbers until the counter equals n.

- **ALU and Control Signals:**

  The ALU control signal dictates the operation: add, subtract, etc.

  Control signals (e.g., RegDst, ALUSrc, MemRead, MemWrite) govern data flow and operations in the processor.

- **Memory Access:**

  Instructions and data are fetched and stored using addresses derived from the Program Counter (PC) and register offsets.

- **Hierarchical Referencing:**

  The testbench accesses internal module signals via hierarchical referencing (e.g., uut.datapath.PC). This facilitates debugging and observation.

2.3 Instruction Execution Cycle

The processor follows the standard instruction cycle: fetch, decode, execute, memory access, and write-back. Each step is handled within the datapath and is controlled by signals generated by the control unit.

2.4 ALU Operations

The Arithmetic Logic Unit performs essential operations such as addition, subtraction, and comparison. Control signals derived from the instruction opcode determine the exact operation.

2.5 Register File Usage

The processor uses a set of general-purpose registers for storing intermediate and final computation values. For instance, during Fibonacci sequence computation, specific registers hold the previous terms, current term, loop counter, and temporary values.

2.6 Hierarchical Debugging

By exposing internal components such as the datapath and control unit to the testbench, internal values can be directly observed, aiding in debugging and verification.

3 Process to Obtain the Design

3.1 Testbench Framework

The testbench includes several critical components to ensure comprehensive testing:

- **Clock Generation:** A periodic clock signal drives the processor. A standard 10ns clock period is used to mimic real-time operation.
- **Reset Control:** A reset signal initializes the processor's internal states before beginning execution. It is activated briefly at the beginning of the simulation.
- **Processor Integration**: The processor module is instantiated and connected to various internal signal probes for observability. This includes connections to control lines, ALU outputs, memory interfaces, and register file outputs.
- **Internal Signal Monitoring:** The testbench accesses key datapath and control signals such as the program counter, current instruction, ALU operation result, and register values. Monitoring these signals helps verify correct instruction flow and data manipulation.
- **Simulation Control:** The simulation runs long enough to allow complete execution of the Fibonacci sequence algorithm. Time-based monitoring and display functions are incorporated to output results to the console.

3.2 MIPS R-format instructions

These instructions perform operations using register operands only. They typically include arithmetic, logical, and shift operations. The result is stored in a destination register. Instruction fields:

- op: operation code (opcode)
- rs: first source register number
- rt: second source register number
- rd: destination register number
- shamt: shift amount
- funct: function code (extends opcode)

3.3 MIPS I-format instructions

These instructions use one register and an immediate value (constant). They are used for arithmetic with constants, memory access, and conditional branching.

3.4 MIPS J-format instructions

These instructions are used for unconditional jumping to a new address. They alter the control flow by updating the program counter directly.

4 Design Methodology

These figure illustrate the single cycle datapaath path with control and different types of instruction. The following is the main control unit:

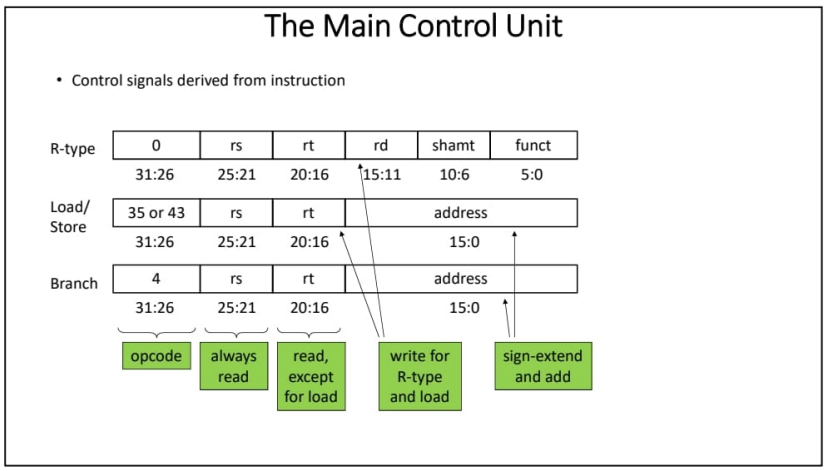
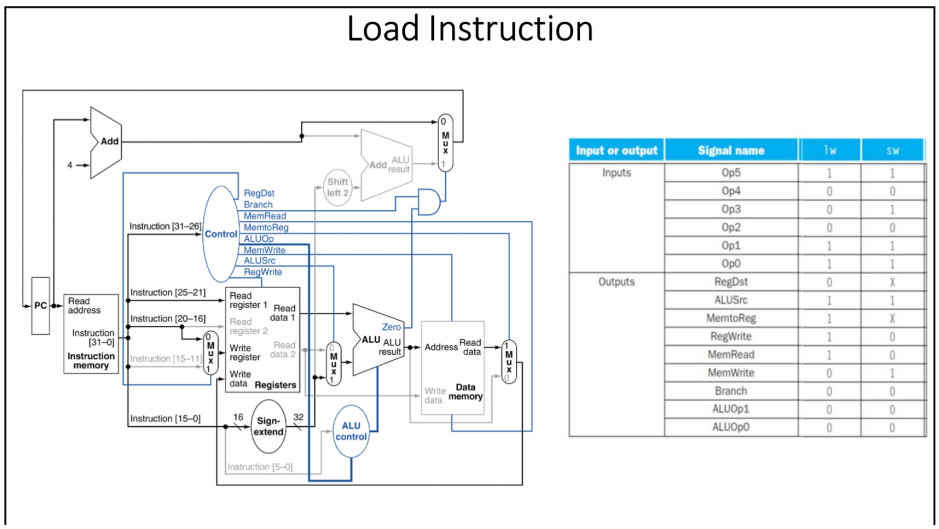


Figure 1:



Input or output	Signal name	lw	sw
Inputs	Op5	1	1
	Op4	0	0
	Op3	0	1
	Op2	0	0
	Op1	1	1
	Op0	1	1
Outputs	RegDst	0	X
	ALUSrc	1	1
	MemtoReg	1	X
	RegWrite	1	0
	MemRead	1	0
	MemWrite	0	1
	Branch	0	0
	ALUOp1	0	0
	ALUOp0	0	0

Figure 2:

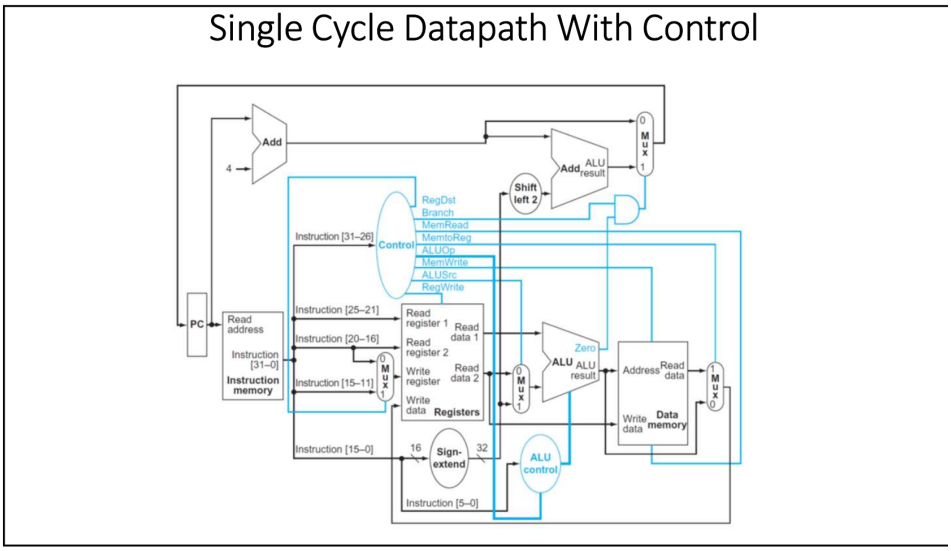
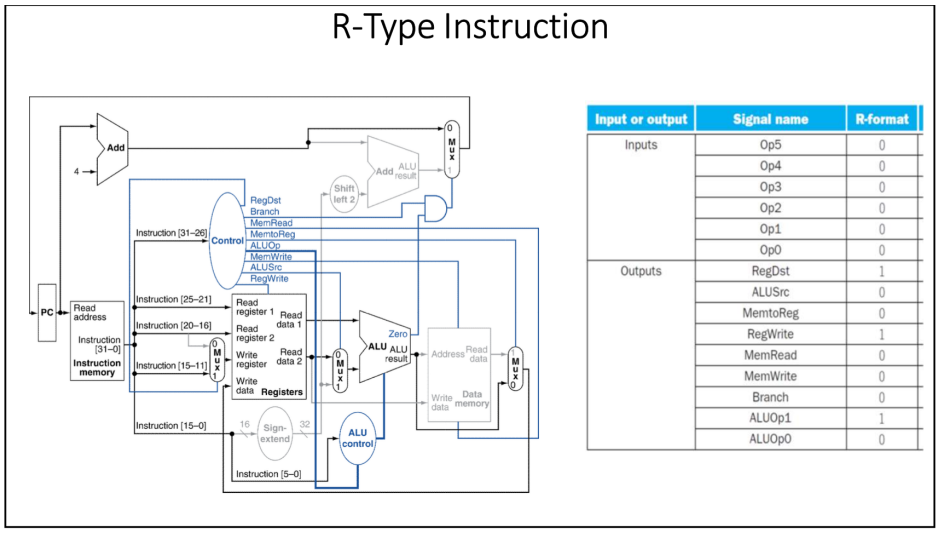


Figure 3:



Input or output	Signal name	R-format
Inputs	Op5	0
	Op4	0
	Op3	0
	Op2	0
	Op1	0
	Op0	0
Outputs	RegDst	1
	ALUSrc	0
	MemtoReg	0
	RegWrite	1
	MemRead	0
	MemWrite	0
	Branch	0
	ALUOp1	1
	ALUOp0	0

Figure 4:

5 Conclusion

- The testbench effectively validates the operation of a MIPS processor by simulating the computation of the Fibonacci sequence.
- It demonstrates proper execution of different instruction types (R, I, J), highlights the operation of the ALU and control unit, and confirms correct memory and register interaction. The detailed visibility into processor internals provides a robust environment for functional verification, and serves as a foundational tool for future enhancements such as pipeline support, hazard detection, and exception handling.
- Overall, this project provided a comprehensive view of how to design, simulate, and implement arithmetic circuits in hardware using Verilog. It also highlights performance comparisons between different design choices, guiding future designs in balancing resource usage and speed.