Hardware Lab (CS 224)
Assignment 6

Group No. 3

Team Members:
Vaibhav Singh (230101110)
Toshit Jain (230101106)
Tanmay Pratap Singh (230101102)
Tedla Mahaswin (230101104)

April 22, 2025

Contents

1 Objective

The objective of this assignment is to design and implement a single-cycle 32-bit MIPS processor using Verilog. The processor supports a core subset of MIPS instructions including arithmetic, logic, memory access, and control flow operations. This report elaborates on the datapath design, individual modules, supported instructions, and testing methodology for the processor.

2 Supported Instructions

The processor supports the following MIPS instructions:

- Memory Access: `lw`, `sw`
- Arithmetic/Logical: `add`, `sub`, `and`, `or`, `slt`
- Control Transfer: `beq`, `j`, `jal`, `jr`

R-type: Used for arithmetic/logical operations and `jr`.
Format: `opcode(6) rs(5) rt(5) rd(5) shamt(5) funct(6)`
Example: `add $t0, $s1, $s2` → `000000 10001 10010 01000 00000 100000`

I-type: Used for memory and branch operations.
Format: `opcode(6) rs(5) rt(5) immediate(16)`
Example: `lw $t0, 32($t1)` → `100011 01001 01000 0000000000100000`

J-type: Used for jumps.
Format: `opcode(6) target(26)`
Example: `j 0x00000010` → `000010 00000000000000000000010000`

3 Modules Description

3.1 Program Counter (PC)

Maintains the address of the current instruction. Updated every clock cycle to either `PC + 4`, a branch target, or jump destination. The PC ensures that the processor executes instructions in a sequential manner, unless modified by control flow instructions such as branches or jumps. It is critical for ensuring the proper flow of execution and is usually incremented by 4 to point to the next instruction in memory.

3.2 Instruction Memory

Read-only module storing the instruction set. Accessed using a word-aligned address (`address[11:2]`). This module retrieves the instruction from memory based on the program counter (PC) address. The instruction memory is typically implemented as a simple ROM (Read-Only Memory), where each instruction is fetched in the sequence dictated by the PC. The memory is designed to support fast lookups and word-aligned access for efficient instruction fetching.

3.3 Register File

Contains 32 general-purpose registers. Supports 2 reads and 1 write. Register 0 is hardwired to 0. The register file is responsible for storing values that are used in computations, and it allows two registers to be read simultaneously. It provides an interface for the processor to interact with the data values needed for arithmetic, logic, and memory operations. Register 0 being hardwired to 0 ensures that certain instructions can operate without needing special checks for zero values, making operations more efficient.

3.4 ALU

Performs operations like addition, subtraction, AND, OR, and set-less-than. Outputs a zero flag for conditional branches. The ALU is a key component in performing arithmetic and logic operations. It takes two inputs, applies the selected operation, and produces a result. Additionally, the ALU outputs a zero flag, which is crucial for branch instructions like `beq`, as it helps determine whether a branch should occur based on the result of a comparison.

3.5 ALU Control

Generates specific ALU operation signals based on `ALUOp` and `funct` field. Also detects `jr`. This module is responsible for decoding the ALU operation specified by the instruction. It reads the opcode and funct field to determine the exact operation to be performed (addition, subtraction, etc.). It also handles the `jr` instruction, which is a jump based on the contents of a register, overriding the regular program flow to return from a subroutine.

3.6 Sign Extender

Extends 16-bit immediate values to 32-bit for use in the ALU. The sign extender ensures that the immediate values in I-type instructions, which are only 16 bits, are properly extended to 32 bits. It handles the sign extension by replicating the sign bit (most significant bit) of the immediate value to the left. This is important for ensuring correct arithmetic when working with negative numbers.

3.7 Data Memory

Supports both read and write operations for `lw` and `sw`. Uses `address[11:2]` for word access. Data memory is used for loading and storing data in the processor. The `lw` (load word) instruction reads data from the memory and places it in a register, while the `sw` (store word) instruction writes data from a register to memory.

3.8 Control Unit

Generates control signals based on the `opcode`. Determines the values of `RegDst`, `ALUSrc`, `MemToReg`, `RegWrite`, `MemRead`, `MemWrite`, `Branch`, `Jump`, and `Jal`.

4 PC Logic and Jump Handling

`jal` updates the PC to the target address and saves `PC+4` into register 31. `jr` overrides all and sets `PC = rs`, typically returning from a subroutine. Detected in the `ALUControl` module.

5 Instruction Memory Initialization Example

```
memory[0] = 32'b000000_10001_10010_01000_00000_100000;
                            // add $t0, $s1, $s2

memory[1] = 32'b100011_10011_01101_0000000000000000;
                            // lw $t5, 0($s3)

memory[2] = 32'b000100_01000_01001_1111111111111101;
                            // beq $t0, $t1, -3

memory[3] = 32'b000011_00000000000000000000001000;
                            // jal 0x00000008

memory[4] = 32'b000000_11111_00000_00000_00000_001000;
                            // jr $ra
```

6 Conclusion

The modular design of this single-cycle MIPS processor ensures clarity, reusability, and testability. By supporting all core instructions, including `jal` and `jr`, this design is functionally complete for simple program execution and subroutine handling. Testing with diverse instruction combinations verified correct operation.