Design Specification for a 32-bit 5-Stage Pipelined MIPS Processor

Table of Contents

---

1. Overview

This design specification outlines the implementation of a 32-bit 5-stage pipelined MIPS processor. The processor supports a subset of the MIPS32 instruction set, and its primary design objective is efficient execution using instruction-level parallelism through pipelining. This document covers the processor's architecture, pipeline structure, control logic, hazard detection, and memory management, followed by the testing and verification methodology.

2. Processor Architecture

2.1 Processor Features

- **32-bit word size**: The processor works with 32-bit data and addresses.
- **5-stage pipeline**: Composed of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB) stages.
- **Pipelined design**: Enables instruction-level parallelism for improved performance.
- **Load/store architecture**: Instructions can only access memory through `lw` (load word) and `sw` (store word) operations.
- **MIPS ISA subset**: Supports basic arithmetic, logical, and control flow instructions (add, sub, and, or, beq, j, lw, sw).
- **Hazard management**: Data and control hazards are resolved using forwarding and stalling mechanisms.

2.2 Pipelining Overview

Pipelining splits the execution of instructions into 5 sequential stages, allowing multiple instructions to be executed concurrently at different stages of completion. This approach increases instruction throughput, reducing the overall execution time. However, pipelining introduces hazards (data hazards, control hazards), which must be managed effectively to ensure correct program execution.

2.3 Instruction Set Architecture (ISA)

The processor implements a subset of the MIPS32 ISA. Key instructions include:

- **Arithmetic**: `add`, `sub`, `and`, `or`, `slt`
- **Memory Access**: `lw`, `sw`
- **Branch/Control**: `beq`, `j`

Each instruction has a fixed 32-bit format, with R-type (register), I-type (immediate), and J-type (jump) formats.

3. Pipeline Stages

3.1 Instruction Fetch (IF) Stage

The IF stage fetches the instruction from the instruction memory using the Program Counter (PC). Key components:

- **PC Register**: Holds the current instruction address.
- **Instruction Memory**: A memory array that stores the program instructions.
- **PC + 4 Adder**: Computes the next instruction address by incrementing the current PC by 4 bytes (one word).

Outputs:

- Instruction fetched from memory.
- PC value for the next cycle.

3.2 Instruction Decode (ID) Stage

In the ID stage, the instruction is decoded to determine the operation and required operands. Key components:

- **Control Unit**: Generates control signals based on the instruction opcode.
- **Register File**: Reads the contents of two registers (`rs` and `rt`).
- **Sign/Zero Extension Unit**: Extends immediate values for instructions like `lw`, `sw`, and `beq`.

Outputs:

- Control signals.
- Values of the source registers.
- Immediate value, if applicable.

3.3 Execution (EX) Stage

In the EX stage, the ALU performs arithmetic or logical operations, and memory addresses are computed for load/store instructions. Key components:

- **ALU**: Performs the actual computation based on the control signals (addition, subtraction, bitwise operations).
- **ALU Source Multiplexers**: Select appropriate inputs to the ALU, either register values or immediate values.
- **Branch Target Calculation**: For `beq`, computes the potential branch target address.

Outputs:

- ALU result.
- Branch target address.
- Updated control signals.

3.4 Memory Access (MEM) Stage

The MEM stage handles memory operations for `lw` and `sw` instructions. For other instructions, it simply passes the ALU result to the next stage. Key components:

- **Data Memory**: Handles memory access for load/store instructions.
- **Address Decoder**: Decodes memory addresses for access.

Outputs:

- Data read from memory for `lw` instructions.
- ALU result or data to be written to memory.

3.5 Write Back (WB) Stage

The WB stage writes results back to the register file. This could either be data from memory (`lw`) or an ALU result (arithmetic instructions).

Outputs:

- Data to be written back to the register file.

4. Control and Hazard Management

4.1 Data Hazards and Forwarding

Data hazards occur when an instruction depends on the result of a previous instruction that has not yet completed. The processor uses a **Forwarding Unit** to bypass data from later pipeline stages (EX/MEM/WB) to earlier stages as needed.

- **EX-EX forwarding**: When an instruction in the EX stage needs data from an instruction in the MEM stage.
- **MEM-EX forwarding**: When an instruction in the EX stage needs data from the WB stage.

4.2 Control Hazards and Branch Prediction

Control hazards arise when branch instructions change the program flow. To mitigate this:

- **Branch decision is made in the EX stage**: After the branch target address is calculated and the condition is evaluated.
- **Flush Control**: Ensures that incorrect instructions following a branch are flushed from the pipeline.

4.3 Pipeline Stalls

When forwarding is insufficient to resolve a data hazard (e.g., a load-use hazard), the **Stall Control Unit** inserts a stall in the pipeline, delaying subsequent instructions until the hazard is resolved.

5. Modules and Components

5.1 Instruction Memory

- Holds program instructions.
- Read-only during execution.
- Provides 32-bit instruction words.

5.2 Data Memory

- Holds data for load/store operations.
- Read and write operations are supported.
- 32-bit word addressing.

5.3 ALU (Arithmetic Logic Unit)

- Performs arithmetic (addition, subtraction) and logical (AND, OR) operations.
- Input can be register data or immediate values.
- Generates Zero flag for branch operations.

5.4 Register File

- 32 registers, each 32 bits wide.
- Two read ports and one write port.
- Supports simultaneous read and write operations.

5.5 Control Unit

- Decodes the opcode and generates control signals for each pipeline stage.
- Controls ALU operation, memory access, register writing, and branching.

5.6 Forwarding Unit

- Resolves data hazards by forwarding data from later pipeline stages (MEM or WB) to earlier stages (EX).

5.7 Hazard Detection Unit

- Detects potential data and control hazards.
- Inserts pipeline stalls when forwarding is insufficient.

6. Timing and Clocking

- The processor operates on a single clock cycle per pipeline stage.
- Each stage operates concurrently, with each instruction taking 5 cycles to complete in the pipeline.
- Timing analysis must be performed to ensure that setup and hold times are met.

7. Testing and Verification

7.1 Test Strategy

- **Unit testing**: Each module (ALU, Control Unit, Register File, etc.) is tested individually.
- **Integration testing**: The complete pipeline is tested by running small programs that cover various instruction sequences.
- **Hazard testing**: Programs are designed to create data and control hazards, verifying that forwarding and stalling mechanisms work correctly.

7.2 Simulation and Debugging

- **Testbenches** are written to simulate the behavior of the processor.
- **Waveform analysis** is used to verify the timing of signals and identify any incorrect behavior in the pipeline.

8. Synthesis and FPGA Implementation (Optional)

- For FPGA implementation, synthesis tools (e.g., Xilinx Vivado or Intel Quartus) are used to generate bitstreams for programming the FPGA.
- Constraints files (e.g., `.xdc`) map Verilog module pins to FPGA I/O pins.
