Development Phase

Testing Appendices

Dr. Dheya Mustafa

26/10/2024

Appendix A: Suggested Test Plan for MIPS Processor Verification

The goal of the test plan is to verify the functionality and performance of both the single-cycle and pipelined MIPS processors designed during this phase. This involves functional testing, hazard detection, and performance evaluation for all supported instructions, covering both basic and edge cases.

A.1. Test Objectives:

1. **Verify Functional Correctness**:

- Ensure that each instruction in the defined set executes correctly.
- Validate the design against various instruction combinations, edge cases, and incorrect inputs.
- Compare results with the expected outputs to identify any mismatches.

2. **Identify and Handle Pipeline Hazards** (for pipelined design):

- Detect and handle structural, data, and control hazards.
- Implement and test strategies like forwarding, stalling, and flushing.
- Ensure that hazard handling does not introduce errors or unintended delays.

3. **Evaluate Performance**:

- Compare performance metrics such as clock cycles, execution time, and resource usage between the single-cycle and pipelined designs.
- Benchmark both designs against the same test cases for fair comparison.

A.2. Test Environment Setup:

1. **Simulator and Tools:**

- Quartus Project Setup with the full single-cycle and pipelined processor designs.
- Testing scripts for automatic instruction feeding and output validation.
- Cycle Accurate Simulator for step-by-step execution tracking.

2. **Testing Benchmarks:**

- Standard benchmark programs provided by the committee.
- Additional custom benchmarks to cover corner cases and specific hazard scenarios.

3. **Debugging Tools:**

- Signal and waveform analyzers to observe the internal behavior of registers, memory, and the control unit.
- Log files for instruction traces, hazard occurrences, and data forwarding actions.

A.3. Test Cases:

A.3.1 Instruction-Level Functional Testing:

Test each instruction in the expanded set under the following conditions:

- **Simple Execution:**
- Execute each instruction in isolation (e.g., ADD, SUB, AND).
- Verify register and memory updates.
- **Sequential Execution:**
- Run a sequence of different instructions to simulate a typical program.
- Confirm that instruction dependencies are respected, and correct results are produced.
- **Branch and Jump Testing:**
- Test `BEQ`, `BNE`, `BLTZ`, `BGEZ`, `JUMP`, `JR`, and `JAL` instructions.
- Check correct branching behavior and program counter (PC) updates.
- **Arithmetic and Logical Operations:**
- Include tests for overflow, underflow, and zero conditions.
- Validate bitwise operations like `AND`, `OR`, `XOR`, and `NOR`.
- **Memory Operations:**
- Test `LW` and `SW` instructions.
- Validate correct address computation and data storage/retrieval.

A.3.2 Pipeline Hazard Detection and Handling (for Pipelined Design):

Create specific tests to trigger and validate the handling of different types of hazards:

- **Data Hazards:**
- Introduce dependencies between consecutive instructions.
- Validate the effect of data forwarding or stalling on execution.

- **Control Hazards:**
- Test branching and jumping in the middle of a sequence.
- Check that the pipeline flushes or stalls as needed to resolve incorrect PC updates.
- **Structural Hazards:**
- Test concurrent memory accesses and instruction fetches.
- Confirm that the pipeline manages these without data corruption.

A.4 Performance Testing and Evaluation:

Execute various benchmark programs and measure key metrics for both designs:

1. **Execution Time:**
- Measure clock cycles and real-time performance.
- Compare the single-cycle processor against the pipelined design.
2. **Instruction Throughput:**
- Track how many instructions are completed per cycle (IPC).
- Analyze the impact of hazards and forwarding mechanisms on throughput.
3. **Resource Utilization:**
- Evaluate the design size (logic elements, memory usage).
- Compare resource utilization for the single-cycle and pipelined processors.
4. **Critical Path Analysis:**
- Identify critical paths and bottlenecks in each design.
- Measure clock frequency and determine which design can achieve higher operational speeds.

A.5. Testing Procedure:

1. **Simulation and Synthesis:**
- Perform behavioral simulation to verify functionality.
- Synthesize the design in Quartus to check for timing violations and logic errors.

2. **Instruction Set Testing:**

- Run the complete set of MIPS instructions and validate outputs.
- Use automated scripts to test each instruction and compare against golden models.

3. **Pipeline Verification:**

- Run step-by-step pipeline simulations.
- Introduce known hazard scenarios and check pipeline behavior at each stage.

4. **Performance Metrics Evaluation:**

- Record cycle counts, execution time, and clock speed.
- Perform detailed comparisons between single-cycle and pipelined designs.

A.6. Expected Deliverables:

1. **Test Report:**

- Detailed documentation of all test cases, test results, and identified issues.
- Performance metrics and analysis comparing the two designs.

2. **Quartus Project Files:**

- Synthesized and verified project files for both single-cycle and pipelined processors.
- Simulation waveforms demonstrating correct instruction execution and hazard handling.

3. **Cycle Accurate Simulator (if applicable):**

- Implementation details and test cases for the simulator.
- Cycle-by-cycle instruction traces and hazard resolution logs.

4. **Performance Comparison Summary:**

- Highlight key advantages and disadvantages of each design.
- Include suggestions for further optimizations and future enhancements.

Appendix B: Coverage Points for MIPS Processor Verification

An extended set of coverage points for verifying the MIPS processor design will include coverage metrics for instruction functionality, pipeline stages, hazard handling, control flow, and performance metrics. These coverage points aim to ensure that the design behaves correctly under all scenarios, including edge cases and corner cases, and that it meets performance expectations.

B1. Instruction Functional Coverage

These coverage points will ensure that each instruction in the ISA is executed correctly, handles various operand configurations, and interacts with other instructions as expected.

1.1 Basic Instruction Coverage:

- Each instruction in the ISA (`ADD`, `SUB`, `AND`, `OR`, `XOR`, etc.) is executed at least once.
- Execution with positive, negative, and zero operands.
- Verify results for basic arithmetic operations with different operand values.

1.2. Memory Operation Coverage:

- `LW` and `SW` instructions are executed with:
  - Different memory addresses (aligned and misaligned addresses).
  - Boundary conditions (first and last addresses of memory).
- Test with both direct and computed addresses.

1.3. Branch and Jump Coverage:

- Test branch instructions (`BEQ`, `BNE`, `BLTZ`, `BGEZ`) under:
  - Taken and not-taken conditions.
  - Conditions with overlapping memory addresses.
- Jump instructions (`J`, `JR`, `JAL`) tested for:

- Jumps forward and backward in the code.

- Jumps to invalid memory locations (error conditions).

1.4. Immediate and Register-Based Instructions:

- Cover different combinations of immediate values and register-based operations.

- Test with maximum, minimum, and zero immediate values.

1.5. Pseudo-Instruction Coverage:

- Implement pseudo-instructions using available instructions.

- Ensure that pseudo-instructions generate expected control signals.

B2. Pipeline Coverage (Pipelined Design Only)

These coverage points ensure that each stage of the pipeline functions correctly and that all inter-stage dependencies are managed properly.

2.1. Pipeline Stage Execution Coverage:

- Each instruction goes through all five-pipeline stages (Fetch, Decode, Execute, Memory, Write-back).

- Track how each instruction type (e.g., arithmetic, memory, control) behaves in each stage.

2.2. Pipeline Stall and Forwarding Coverage:

- Pipeline stalling is triggered and resolved correctly for:

  - Data hazards (e.g., RAW dependencies).

  - Control hazards (e.g., branch misprediction).

- Data forwarding mechanisms are exercised under various scenarios:

  - Forward from Execute to Decode stage.

- Forward from Memory to Decode stage.

- All combinations of source and destination registers.

2.3. Hazard Resolution Coverage:

- Each type of hazard (data, control, structural) is identified and resolved:

- Test each type of hazard individually and in combination.

- Scenarios with multiple hazards occurring simultaneously.

2.4. Flush and Stall Coverage:

- Pipeline flush is triggered correctly on branch misprediction.

- Pipeline stall is triggered when required, and no additional instructions are executed.

2.5. NOP Insertion Coverage:

- Validate that NOP (No Operation) instructions are correctly inserted when stalling is required.

- Check that no unintended NOPs are inserted during normal operation.

B3. Control Flow Coverage

Ensure that the control signals generated by the control unit are correct and that the processor executes control flow instructions accurately.

3.1. Control Signal Coverage:

- Verify each control signal (`RegWrite`, `MemRead`, `MemWrite`, `Branch`, etc.) for each instruction.

- Test that the signals change correctly when transitioning between instructions.

3.2. Instruction Decode Coverage:

- All possible combinations of opcode and function fields are decoded correctly.
- Invalid opcode detection and handling.

3.3. Program Counter (PC) Update Coverage:

- PC updates correctly on sequential instruction execution.
- Branch instructions correctly modify the PC when branch conditions are met.
- Jumps, returns, and calls update the PC correctly.

3.4. ALU Control Signal Coverage:

- Test all combinations of ALU control signals for different operations.
- ALU behavior for overflow, underflow, and zero results.

B4. Hazard and Dependency Coverage

Ensure that dependencies between instructions are correctly identified and handled.

4.1. Data Dependency Coverage:

- RAW (Read-After-Write), WAW (Write-After-Write), and WAR (Write-After-Read) dependencies:
  - Cover register dependencies with adjacent and non-adjacent instructions.
- Test data dependencies under various pipeline conditions (stalling, forwarding).

4.2. Control Dependency Coverage:

- Branch instructions followed by dependent instructions.
- Correct handling of instructions dependent on branch outcomes.

4.3. Structural Hazard Coverage:

- Concurrent accesses to the same functional units (e.g., ALU, memory).
- Test for correct resolution (e.g., stalling or serialization).

4.4. Load/Store Hazard Coverage:

- Test **scenarios** where loads and stores access the same memory location.
- Multiple stores to the same address, ensuring the last store takes precedence.

B5. Performance and Timing Coverage

Evaluate the processor's timing and performance characteristics under different workloads.

5.1. Cycle Count Coverage:

- Count the number of cycles taken for each instruction type.
- Ensure that cycle counts match expected values for single-cycle and pipelined designs.

5.2. Critical Path Coverage:

- Measure the timing of critical paths in each design.
- Ensure timing constraints are met under different operating conditions.

5.3. Execution Speed Coverage:

- Test with a range of clock frequencies to measure the impact on performance.
- Track throughput (IPC) and latency for different instruction sequences.

5.4. Power and Area Coverage:

- Measure power consumption for different workloads.

- Monitor resource utilization (e.g., logic elements, memory units).

B6. Error and Edge Case Coverage

Test the design's ability to handle incorrect inputs and abnormal scenarios gracefully.

6.1. Invalid Opcode Coverage:

- Test with invalid opcodes and function codes.
- Check that the control unit generates an error or takes appropriate action.

6.2. Out-of-Bounds Memory Access Coverage:

- Load/store instructions targeting memory outside the defined range.
- Ensure the design detects and flags errors.

6.3. Overflow and Underflow Coverage:

- Arithmetic operations causing overflow/underflow.
- Ensure that these conditions are detected and handled.

6.4. Reset and Initialization Coverage:

- Test behavior under reset and power-up conditions.
- Validate register and memory initializations.

6.5. Pipeline Flush and Restart Coverage:

- Pipeline flush and correct restart after branch misprediction.
- Pipeline restarts after exceptions or reset.

B7. Comprehensive Test Bench Coverage:

Ensure that all test benches exercise the complete functionality of the design

7.1. Test Bench for Functional Verification:

- Implement a test bench that covers all functional instructions.
- Include automatic result checking against a reference model.

7.2. Test Bench for Pipeline Verification:

- Create a separate test bench focusing on pipeline behavior.
- Include stall, flush, and hazard scenarios.

7.3. Test Bench for Performance Metrics:

- Measure clock cycles and IPC for various test cases.
- Include benchmarks that stress different parts of the processor.

By defining these extended coverage points, the MIPS processor implementation can be thoroughly tested, ensuring that both the single-cycle and pipelined designs meet functional and performance requirements under all scenarios.