THE AMERICAN UNIVERSITY IN CAIRO

الجامعة الأمريكية بالقاهرة

School of Sciences and Engineering
Department of Computer Science and Engineering
Spring 2024

CSCE 231: Computer Architecture

Project 02: Tomasulo Algorithm Simulation

Dr. Cherif Salama

Submitted By:

Mark Kyrollos - 900211436
Mohamed Sabry - 900211112

Date: **Dec 7, 2024**

TABLE OF CONTENTS

1. Implementation Description:

This Tomasulo Algorithm Simulator models a simplified CPU pipeline with reservation stations and dynamic scheduling. It handles data and control hazards while simulating out-of-order execution. The code's modularity and configurability make it suitable for exploring different CPU designs, though it simplifies memory, register, and branch prediction models.

2. Design Decisions and Assumptions:

The design emphasizes modularity and clarity by separating responsibilities into distinct classes and structures. For example, the Instruction class focuses on parsing and managing individual instructions, while the FunctionalUnitManager handles reservation stations and functional unit cycles. This modular approach ensures flexibility and maintainability, allowing easy modifications or extensions to the simulator, such as adding new instruction types or functional units.

Dynamic configurability is another critical design choice, enabling the user to specify the number of functional units and their respective cycle times. This allows the simulator to model various CPU designs and evaluate their performance. Furthermore, the simulator tracks dependencies explicitly for each instruction, ensuring accurate resolution of hazards and correct out-of-order execution.

To ensure proper execution sequencing, a priority queue is employed during the write-back phase, which maintains a strict order for completing operations and updating dependencies. Additionally, the code supports detailed tracking of instruction states, logging each phase of the instruction lifecycle (issue, execution, and write-back). This feature is crucial for debugging and performance analysis.

The design also incorporates a basic mechanism for simulating control hazards, including branch prediction for instructions like BNE, CALL, and RET. This addition enhances the realism of the simulation and provides insights into the impact of control flow changes on pipeline efficiency

Assumptions

The simulator assumes a simplified model of CPU execution with eight general-purpose registers (R0 to R7). This fixed register count streamlines dependency tracking and reduces complexity, though it may not accurately reflect architectures with larger register files.

The instruction set is limited to a predefined set of operations such as ADD, ADDI, DIV, NAND, LOAD, STORE, BNE, CALL, and RET. Each operation has a fixed cycle latency, and it is assumed that the user inputs valid positive values for these latencies during configuration.

Memory is modeled as a fixed-size array, and it is assumed that all memory accesses remain within the valid bounds. The simulator also requires the user to provide a starting address for instructions, which must align with the simulated memory layout.

The implementation models a single-core processor and does not account for multithreading or other advanced parallel execution scenarios. Additionally, the branch prediction mechanism is relatively simplistic and may not reflect the complexity of modern branch prediction algorithms.

Overall, these assumptions simplify the implementation, focusing on core Tomasulo algorithm principles while abstracting more complex architectural details.

3. Code Analysis

This code implements a Tomasulo Algorithm Simulator, which is a hardware-based dynamic scheduling method for out-of-order execution in CPUs. The implementation simulates functional units, reservation stations, instruction dependencies, memory, and registers. It tracks instruction execution through issue, execution, and write-back phases while resolving data hazards using a dependency tracking system. As for our modules developed using Verilog programming language, we have developed numerous modules to support our complete pipelined RISC-V processor.

Code Insights

1. **Comprehensive Tomasulo Simulation**
   The code implements a thorough simulation of the Tomasulo algorithm,

including all major components like instruction parsing, reservation stations, register files, memory, and pipeline phases (Issue, Execute, WriteBack). This approach captures the essence of dynamic scheduling and out-of-order execution in modern CPUs.

2. **Dependency Tracking for Hazards**
   The simulator employs detailed dependency tracking for register and reservation station interactions. Fields like **dependency1** and **dependency2** in **UnitEntry** ensure that data hazards are resolved before instruction execution. This mechanism is crucial for maintaining correctness in out-of-order execution.

3. **Flexible Functional Unit Management**
   The **FunctionalUnitManager** provides dynamic configuration of functional unit counts and cycle latencies. This flexibility allows the simulator to model different CPU configurations, making it suitable for exploring architectural trade-offs.

4. **Explicit Instruction Lifecycle Management**
   The instruction lifecycle is carefully managed, with each instruction going through **Issue**, **Execute**, and **WriteBack** phases. The use of queues (e.g., **issuedQueue**, **executingQueue**, **writeBackQueue**) ensures that instructions are handled in the correct order and their statuses are logged accurately.

5. **Priority in Write-Back**
   A priority queue is used during the write-back phase to ensure that instructions are written back in the correct sequence. This avoids overwriting hazards and ensures the proper completion of dependent instructions.

6. **Branch and Control Flow Handling**
   The simulation includes basic support for control flow instructions **(CALL**, **RET**, **BNE**), simulating branch prediction and managing program counter (PC) updates. This demonstrates an understanding of control hazards and their impact on pipeline performance.

7. **Instruction Parsing Robustness**
   The **Instruction** class efficiently parses input into components like **op**, **RD**, **RS1**, and **IMM**. The use of helper functions like **string_split** makes the parsing process modular and reusable.

3. Test Files

For our test files, here are some test cases used for our main program:

1.
ADDI R1, R0, 1
ADDI R2, R0, 5
BNE R5, R2, 2
ADDI R3, R3, 2
ADDI R5, R5, 1
ADDI R6, R0, 6
2.
ADDI R1, R0, 10
ADDI R2, R0, 20
ADD R3, R1, R2
DIV R4, R3, R1
NAND R5, R1, R2
ADDI R6, R0, 5
3.
ADDI R1, R0, 6
ADDI R2, R0, 3
DIV R3, R1, R2
ADDI R4, R3, 10
STORE R4, 0(R0)
LOAD R5, 0(R0)
4.

```
ADDI R1, R0, 1
ADDI R2, R0, 5
BNE R5, R2, 2
ADDI R3, R3, 2
ADDI R5, R5, 1
ADDI R6, R0, 6
```

5. Conclusion

Our Tomasulo Algorithm Simulator is a robust and comprehensive simulation showcasing key concepts of dynamic scheduling, out-of-order execution, and hazard resolution in modern CPUs. By integrating components like reservation stations, dependency tracking, and instruction lifecycle management, it effectively models the behavior of a simplified CPU pipeline. Its modularity and configurability make it a valuable tool for studying architectural trade-offs and pipeline performance. While the implementation achieves its primary objectives, it offers room for enhancements, such as advanced branch prediction, scalability for larger instruction sets, and performance optimization. Overall, this simulator serves as an excellent foundation for understanding and exploring CPU design principles.